Night Session
Best of Java 17 + 18

https://github.com/Michaeli71/NightSession-Best-Of-Java-17+18

Michael Inden

Freiberuflicher Consultant, Buchautor und Trainer

Speaker Intro

- **Michael Inden**, Jahrgang 1971
- Diplom-Informatiker, C.v.O. Uni Oldenburg
- ~8 ¼ Jahre **SSE** bei Heidelberger Druckmaschinen AG in Kiel
- ~6 ¾ Jahre **TPL, SA** bei IVU Traffic Technologies AG in Aachen
- ~4 ¼ Jahre **LSA / Trainer** bei Zühlke Engineering AG in Zürich
- ~3 Jahre **TL / CTO** bei Direct Mail Informatics / ASMIQ in Zürich
- Freiberuflicher **Consultant**, **Trainer** und **Konferenz-Speaker**
- **Seit Januar 2022 Head of Development bei Adcubum in Zürich**
- Autor und Gutachter beim dpunkt.verlag / APress

E-Mail: michael_inden@hotmail.com
Blog: https://jaxenter.de/author/minden

Kurse: Bitte sprecht mich an!

https://github.com/Michaeli71/NightSession-Best-Of-Java-17+18

Agenda

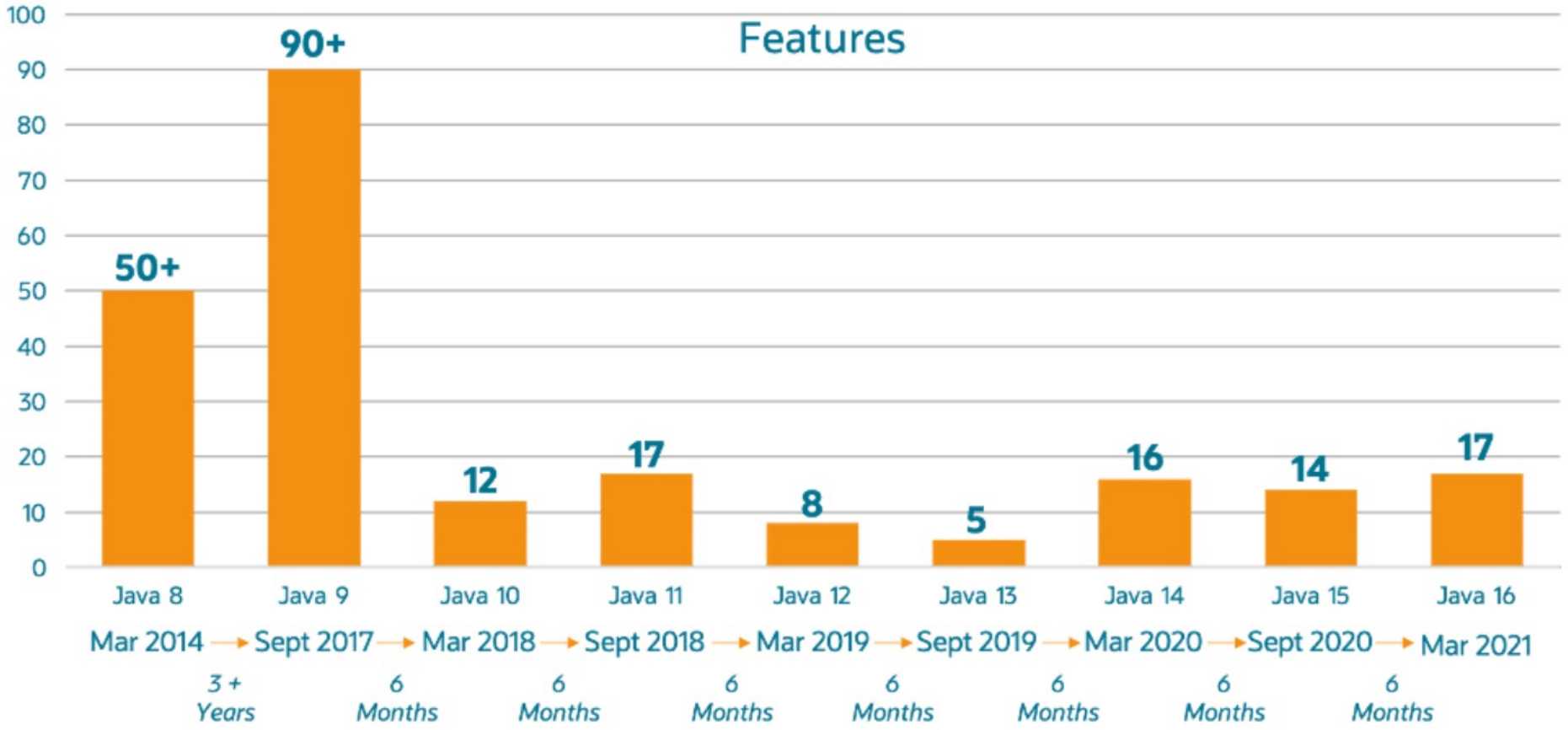
Workshop Contents

- **Vorbemerkungen / Build Tools & IDEs**
- **PART 1:** Syntax-Erweiterungen in Java 12 bis 17
- **PART 2:** Neuheiten und Änderungen in APIs und JVM in Java 12 bis 17
- **PART 3:** Neuheiten in Java 18 im Überblick

Long-Term Support-Modell

- **alle drei -> zwei Jahre Long Term Support (LTS) Release**
  - erhalten über längere Zeit Updates
  - Produktionsversionen
  - derzeit Java 8, 11 und 17
  - aktuelles LTS-Release ist Java 17 (September 2021)
  - **Seit Java 17 wieder ALLE 2 Jahre UND FOR FREE ☺**
- **andere Versionen sind "nur" Zwischenversionen**
  - erhalten nur 6 Monate Updates
  - Previews
  - Ideal um neue Features kennenzulernen und zum Experimentieren (vor allem privat)

Einordnung 6 monatiger Releasezyklus

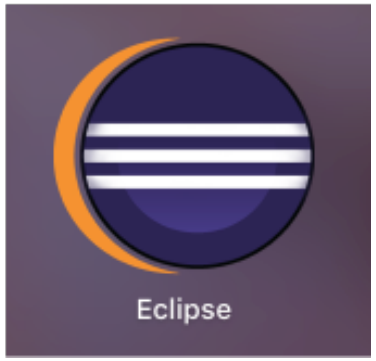


https://www.techzine.eu/news/devops/57122/oracle-releases-version-16-of-java/

Build-Tools und IDEs

IDE & Tool Support für Java 18 (17)

- **Aktuelle IDEs & Tools grundsätzlich gut**
- **Eclipse: Version 2022-03 mit zusätzlichem Plugin**
- **IntelliJ: Version 2022.1 EAP**
- **Maven: 3.8.5, Compiler-Plugin: 3.8.1**
- **Gradle: 7.4.1**
- **Aktivierung von Preview-Features nötig**
  - In Dialogen
  - Im Build-Skript

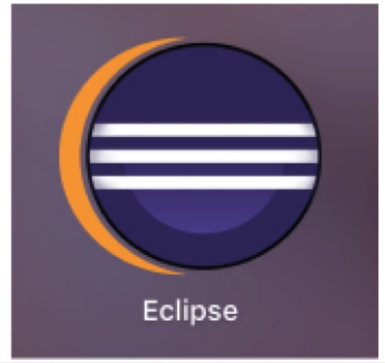


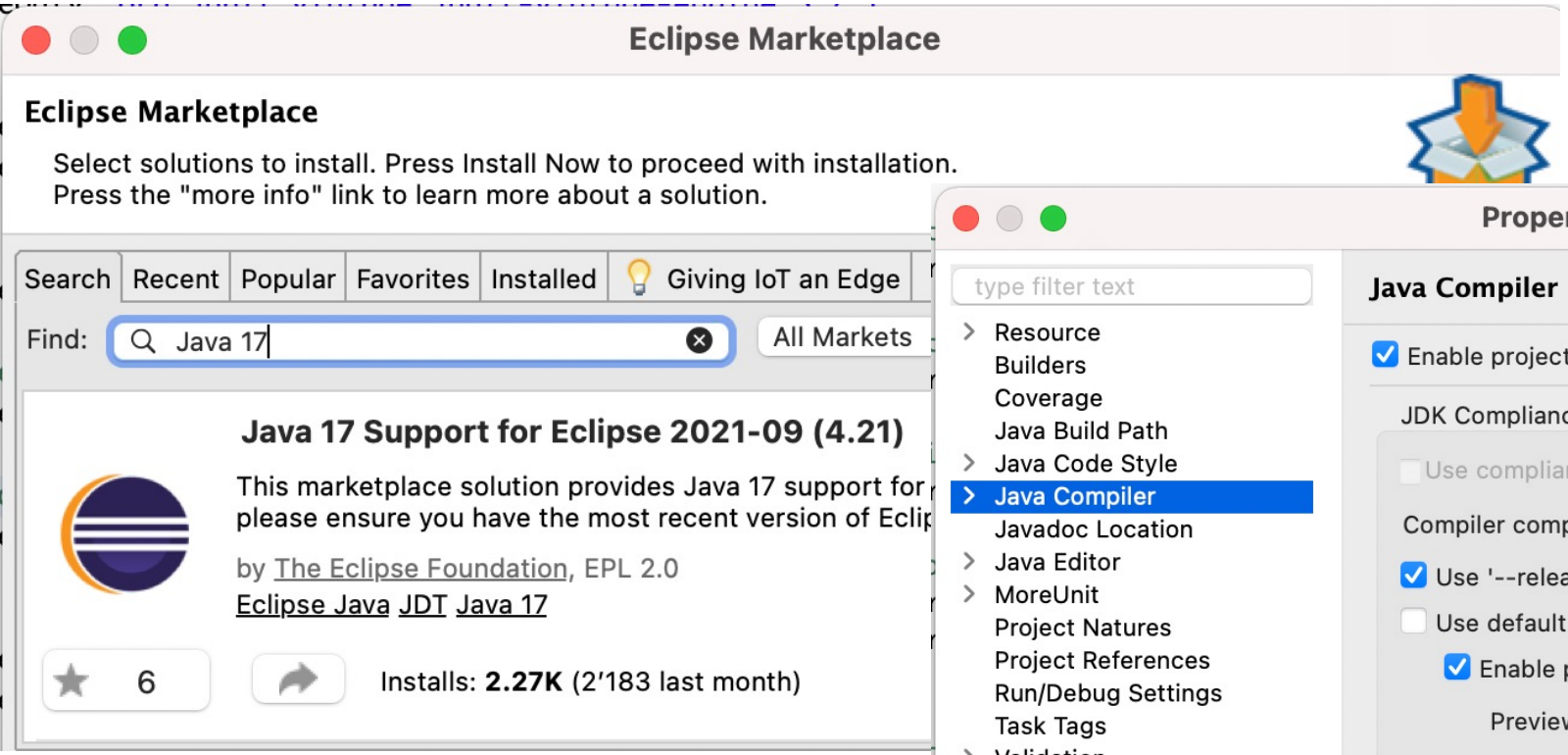
Maven™

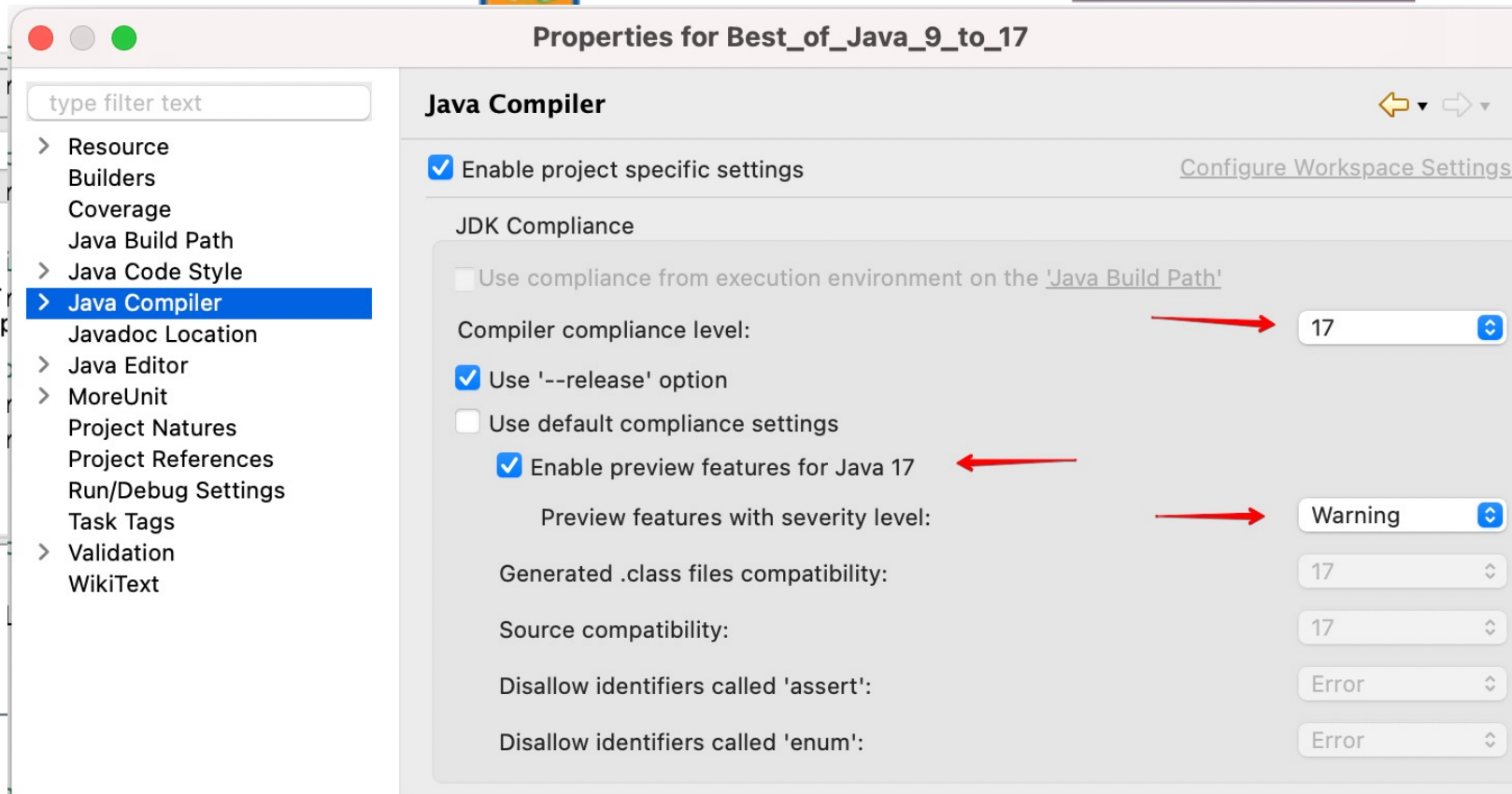
Gradle

IDE & Tool Support Java 17

- Eclipse 2021-09 mit Plugin
- Aktivierung von Preview-Features nötig

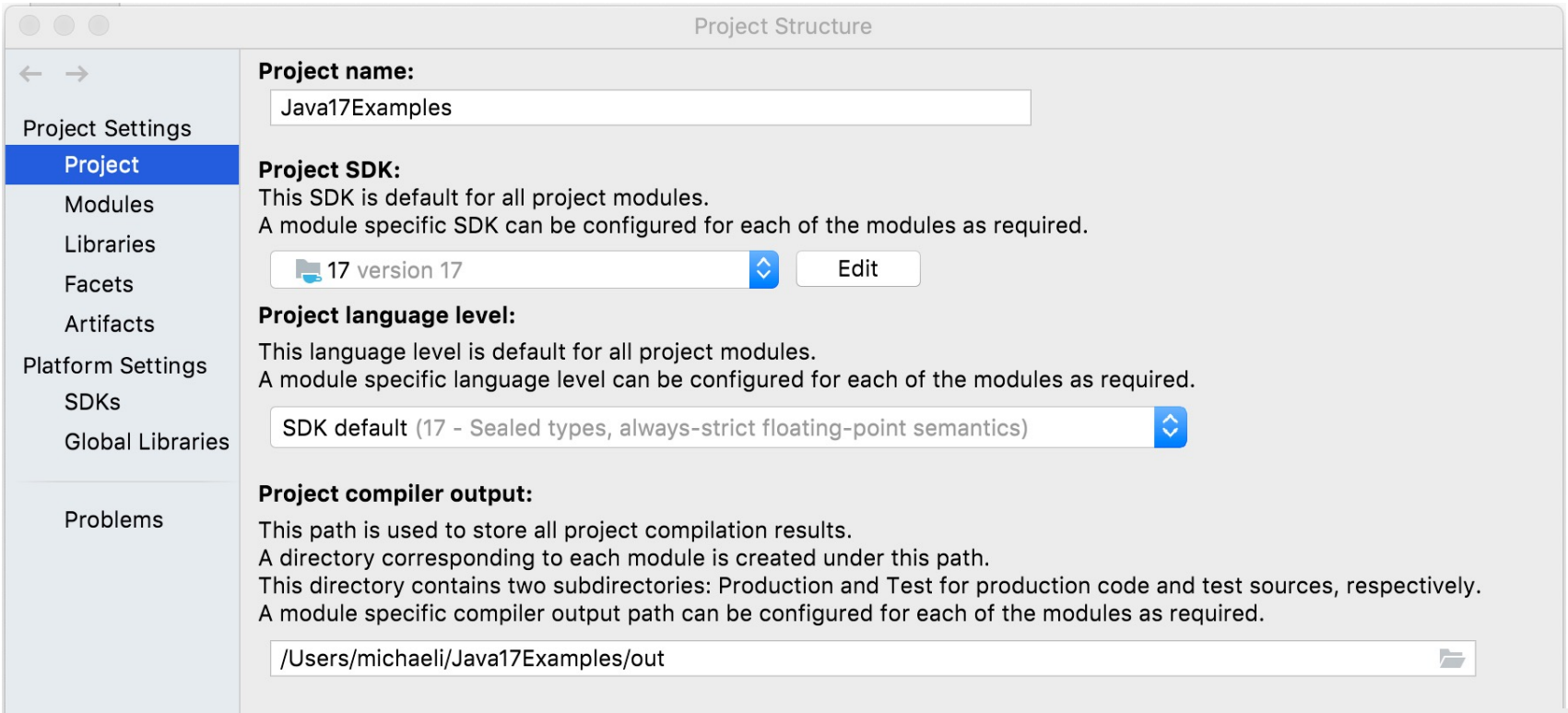


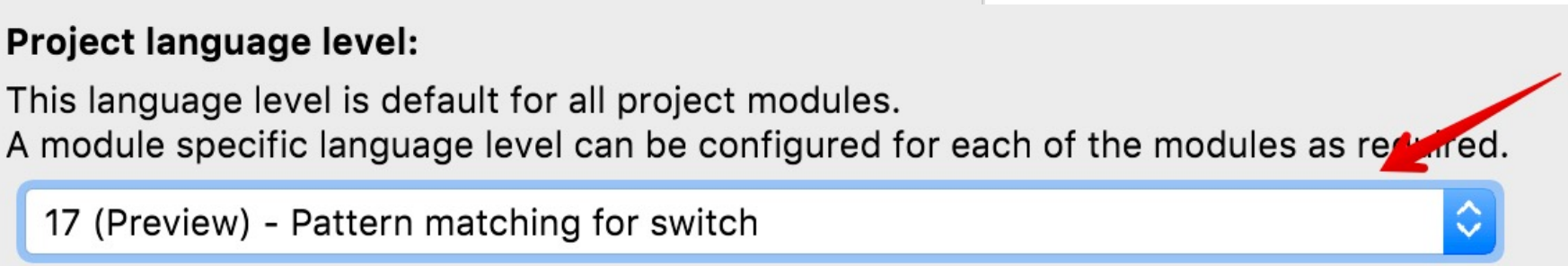




IDE & Tool Support Java 17

- **Aktivierung von Preview-Features nötig**





IDE & Tool Support Java 17 / 18

- **Aktivierung von Preview-Features nötig**

```
sourceCompatibility=17
targetCompatibility=17

// Aktivierung von Switch Expressions Preview
tasks.withType(JavaCompile) {
   options.compilerArgs += ["--enable-preview"]
}
```

- **Java 18**

```
sourceCompatibility=18
targetCompatibility=18

// Aktivierung von Switch Expressions Preview
tasks.withType(JavaCompile) {
   options.compilerArgs += ["--enable-preview"]
}
```

IDE & Tool Support Java 17 / 18

- **Aktivierung von Preview-Features nötig**

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
            <source>17</source>
            <target>17</target>
            <!-- Wichtig für Java Syntax-Neuerungen -->
            <compilerArgs>--enable-preview</compilerArgs>
        </configuration>
    </plugin>
</plugins>
```

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
            <source>18</source>
            <target>18</target>
            <!-- Wichtig für Java Syntax-Neuerungen -->
            <compilerArgs>--enable-preview</compilerArgs>
        </configuration>
    </plugin>
</plugins>
```

PART 1: Syntax-Erweiterungen

- Syntaxerweiterungen bei `switch`
- Syntaxerweiterung Hilfreiche `NullPointerExceptions`
- Syntaxerweiterung `Text Blocks`
- Syntaxerweiterung `Records`
- Syntaxerweiterung bei `instanceof`
- Syntaxerweiterung `Local Enums und Interfaces`
- Syntaxerweiterung `Sealed Types / Classes`

Switch Expressions

Switch Expressions

- switch-case-Konstrukt noch bei den uralten Wurzeln aus der Anfangszeit von Java
- Kompromisse im Sprachdesign, die C++-Entwicklern den Umstieg erleichtern sollten.
- Relikte wie das `break` und beim Fehlen des solchen das Fall-Through
- Flüchtigkeitsfehler kamen immer wieder vor
- Zudem war man beim `case` recht eingeschränkt bei der Angabe der Werte.
- Das alles ändert sich glücklicherweise mit Java 13. Dazu wird die Syntax leicht verändert und erlaubt nun die Angabe einer Expression sowie mehrerer Werte beim case:

Switch Expressions: Blick zurück

- **Abbildung von Wochentagen auf deren Länge …**

```
DayOfWeek day = DayOfWeek.FRIDAY;
int numOfLetters = -1;

switch (day)
{
    case MONDAY:
    case FRIDAY:
    case SUNDAY:
        numOfLetters = 6;
        break;
    case TUESDAY:
        numOfLetters = 7;
        break;
    case THURSDAY:
    case SATURDAY:
        numOfLetters = 8;
        break;
    case WEDNESDAY:
        numOfLetters = 9;
        break;
}
```

Switch Expressions als Abhilfe

- **Abbildung von Wochentagen auf deren Länge … elegant mit modernem Java:**

```
DayOfWeek day = DayOfWeek.FRIDAY;
int numLetters = -1;

switch (day)
{
   case MONDAY, FRIDAY, SUNDAY -> numLetters = 6;
   case TUESDAY                -> numLetters = 7;
   case THURSDAY, SATURDAY     -> numLetters = 8;
   case WEDNESDAY              -> numLetters = 9;
};
```

Switch Expressions als Abhilfe

- **Abbildung von Wochentagen auf deren Länge … elegant mit modernem Java:**

Return-Value

```
DayOfWeek day = DayOfWeek.FRIDAY;
int numLetters = switch (day)
{
    case MONDAY, FRIDAY, SUNDAY -> 6;
    case TUESDAY                -> 7;
    case THURSDAY, SATURDAY     -> 8;
    case WEDNESDAY              -> 9;
};
```

Switch Expressions als Abhilfe

- **Abbildung von Wochentagen auf deren Länge … elegant mit modernem Java 1**

```
DayOfWeek day = DayOfWeek.FRIDAY;
int numLetters = switch (day)
{
    case MONDAY, FRIDAY, SUNDAY -> 6;
    case TUESDAY                -> 7;
    case THURSDAY, SATURDAY     -> 8;
    case WEDNESDAY              -> 9;
};
```

- **Elegantere Schreibweise beim case:**
  - Neben dem offensichtlichen Pfeil statt des Doppelpunkts
  - auch mehrere Werte
  - Keim break nötig, auch kein Fall-Through
  - switch kann nun einen Wert zurückgeben, vermeidet künstliche Hilfsvariablen

Switch Expressions: Blick zurück … Fallstricke

- **Abbildung von Monat auf deren Namen …**

```
// ACHTUNG: Mitunter ganz übler Fehler: default mitten zwischen den cases
String monthString = "";
switch (month)
{
    case JANUARY:
        monthString = "January";
        break;
    default:
        monthString = "N/A";  // hier auch noch Fall Through
    case FEBRUARY:
        monthString = "February";
        break;
    case MARCH:
        monthString = "March";
        break;
    case JULY:
        monthString = "July";
        break;
}

System.out.println("OLD: " + month + " = " + monthString); // February
```

Switch Expressions als Abhilfe

- **Abbildung von Monaten auf deren Namen … elegant mit modernem Java:**

```
static String monthToName(final Month month)
{
    return switch (month)
    {
        case JANUARY -> "January";
        default -> "N/A";  // hier KEIN Fall Through
        case FEBRUARY -> "February";
        case MARCH -> "March";
        case JULY -> "JULY";
    };
}
```

Switch Expressions: switch-old Fallstrick mit break

- **Gegeben sei eine Aufzählung von Farben:**

```
enum Color { RED, GREEN, BLUE, YELLOW, ORANGE };
```

- **Bilden wir diese auf die Anzahl an Buchstaben ab:**

```
Color color = Color.GREEN;
int numOfChars;

switch (color)
{
    case RED: numOfChars = 3; break;
    case GREEN: numOfChars = 5; /* break; UPS: FALL-THROUGH */;
    case YELLOW: numOfChars = 6; break;
    case ORANGE: numOfChars = 6; break;
    default: numOfChars = -1;
}
```

Switch Expressions: `yield` mit Rückgabe

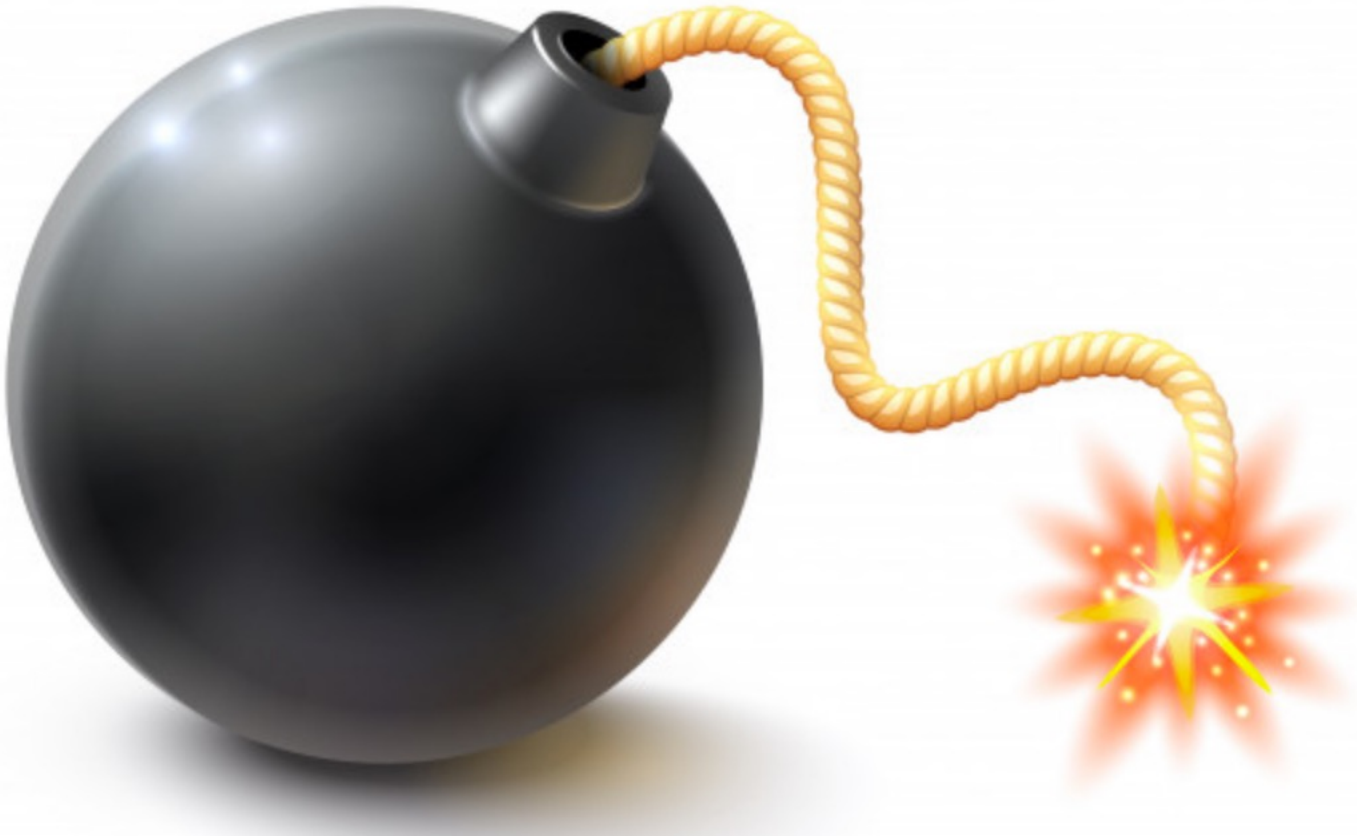
Mit modernem Java wird wieder alles sehr klar und einfach:

```
public static void switchBreakReturnsValue(Color color)
{
    int numOfChars = switch (color)
    {
        case RED: yield 3;
        case GREEN: yield 5;
        case YELLOW, ORANGE: yield 6;
        default: yield -1;
    };

    System.out.println("color: " + color + " ==> " + numOfChars);
}
```

Switch Expressions

```
public static void main(final String[] args)
{
    DayOfWeek day = DayOfWeek.SUNDAY;

    int numOfLetters = switch (day)
    {
        case MONDAY, FRIDAY, SUNDAY -> {
            if (day == DayOfWeek.SUNDAY)
                System.out.println("SUNDAY is FUN DAY");
            yield 6;
        }
        case TUESDAY                -> 7;
        case THURSDAY, SATURDAY     -> 8;
        case WEDNESDAY              -> 9;
    };
    System.out.println(numOfLetters);
}
```

```
SUNDAY is FUN DAY
6
```



NPE

https://www.freepik.com/free-vector/realistic-bomb-illustration_3886825.htm

Hilfreiche NullPointerExceptions

```
public static void main(final String[] args)
{
    SomeType a = null;
    a.value = "ERROR";
}
```

```
Exception in thread "main" java.lang.NullPointerException
at java14.NPE_Example.main(NPE_Example.java:8)
```

Hilfreiche NullPointerExceptions

```
public static void main(final String[] args)
{
    SomeType a = null;
    a.value = "ERROR";
}
```

```
Exception in thread "main" java.lang.NullPointerException
at java14.NPE_Example.main(NPE_Example.java:8)
```

-XX:+ShowCodeDetailsInExceptionMessages

```
Exception in thread "main" java.lang.NullPointerException: Cannot assign field
"value" because "a" is null
at java14.NPE_Example.main(NPE_Example.java:8)
```

Hilfreiche NullPointerExceptions

```
public static void main(final String[] args)
{
    try
    {
        final String[] stringArray = { null, null, null };
        final int errorPos = stringArray[2].lastIndexOf("ERROR");
    }
    catch (final NullPointerException e) { e.printStackTrace(); }
    try
    {
       final Integer value = null;
       final int sum = value + 3;
    }
    catch (final NullPointerException e) { e.printStackTrace(); }
}
```

```
java.lang.NullPointerException: Cannot invoke
"String.lastIndexOf(String)" because "stringArray[2]" is null
at java14.NPE_Second_Example.main(NPE_Second_Example.java:10)
java.lang.NullPointerException: Cannot invoke
"java.lang.Integer.intValue()" because "value" is null
at java14.NPE_Second_Example.main(NPE_Second_Example.java:20)
```

Text Blocks

Text Blocks

- **langersehnte Erweiterung, nämlich mehrzeilige Strings ohne mühselige Verknüpfungen definieren zu können und auf fehlerträchtiges Escaping zu verzichten.**

- **Erleichtert unter anderem den Umgang mit SQL-Befehlen, regulären Ausdrücken oder der Definition von JavaScript in Java-Sourcecode.**

- **ALT**

```
String javaScriptCodeOld = "function hello() {\n" +
        "   print(\"Hello World\");\n" +
        "}\n" +
        "\n" +
        "hello();\n";
```

Text Blocks

- **NEU**

```
String javascriptCode = """
                        function hello()
                        {
                            print("Hello World");
                        }

                        hello();
                        """;


String multiLineString = """
                         THIS IS
                         A MULTI
                         LINE STRING
                         WITH A BACKSLASH \\
                         """;
```

Text Blocks

- https://openjdk.java.net/jeps/326

Traditional String Literals

```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello World.</p>\n" +
              "    </body>\n" +
              "</html>\n";
```

```
String multiLineHtml = """
                       <html>
                           <body>
                               <p>Hello, world</p>
                           </body>
                       </html>
                       """;
```

Text Blocks

- **NEU**

```
String multiLineSQL = """
                      SELECT `ID`, `LAST_NAME` FROM `CUSTOMER`
                      WHERE `CITY` = 'ZÜRICH'
                      ORDER BY `LAST_NAME;
                      """;


String multiLineStringWithPlaceHolders = """
                        SELECT %s
                          FROM %s
                          WHERE %s
                       """.formatted(new Object[]{"A", "B", "C"});
```

Text Blocks

- **NEU**

```
String jsonObj = """
          {
              "name": "Mike",
              "birthday": "1971-02-07",
              "comment": "Text blocks are nice!"
          }
          """;
```

Besonderheit Ausrichtung bei `Text Blocks`

```
jshell> var multiLine = """
   ...>                    Eins
   ...>                     Zwei
   ...>                      Drei
   ...>                       Vier
   ...>                  """
multiLine ==> "   Eins\n    Zwei\n     Drei\n      Vier\n"

jshell> var multiLine = """
   ...>                  _  Eins
   ...>                   _  Zwei
   ...>                    _  Drei
   ...>                   _ _ _Vier
   ...>                  """
multiLine ==> "_  Eins\n _  Zwei\n  _  Drei\n _ _ _Vier\n"
```

Besonderheit bei `Text Blocks`

```
String text = """
          This is a string splitted \
          in several smaller \
          strings.\
          """;

System.out.println(text);
```

```
This is a string splitted in several smaller strings.
```

Records

Wäre es nicht cool, auf einfache Weise DTOs usw. zu definieren?

Erweiterung Record

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

- simplifizierte Form von Klassen für einfache Datencontainer
- Sehr kurze, kompakte Schreibweise
- API ergibt sich implizit aus den als Konstruktorparameter definierten Attributen

```
MyPoint point = new MyPoint(47, 11);
System.out.println("point x:" + point.x());
System.out.println("point y:" + point.y());
```

- Implementierungen von Accessor-Methoden sowie equals() und hashCode() automatisch und vor allem kontraktkonform

Erweiterung Record

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

```
Michaels-iMac:java14 michaeli$ javap MyPoint
Compiled from "MyPoint.java"
final class java14.MyPoint extends java.lang.Record {
  public java14.MyPoint(int, int);
  public int x();
  public int y();
  public java.lang.String toString();
  public int hashCode();
  public boolean equals(java.lang.Object);
}
```

```
public final class MyPoint
{
    private final int x;
    private final int y;

    public MyPoint(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    @Override
    public boolean equals(Object o)
    {
        if (this == o)
            return true;
        if (o == null || getClass() != o.getClass())
            return false;

        MyPoint point = (MyPoint) o;
        return x == point.x && y == point.y;
    }

    @Override
    public int hashCode()
    {
        return Objects.hash(x, y);
    }

    @Override public String toString()
    {
        return "MyPoint[x=" + x + ", y=" + y + "]";
    }

    // Zugriffsmethoden auf x und y
}
```

Records und zusätzliche Konstruktoren und Methoden

```
record MyPoint(int x, int y)
{
    public MyPoint(String values)
    {
        this(Integer.parseInt(values.split(",")[0]),
             Integer.parseInt(values.split(",")[1]));
    }

    public String shortForm()
    {
        return "[" + x + ", " + y + "]";
    }
}
```

```
var topLeft = new MyPoint(17, 19);
System.out.println(topLeft);
System.out.println(topLeft.shortForm());
```

```
MyPoint[x=10, y=10]
[10, 10]
```

Records für DTO / Parameter Value Objects

```
record TopLeftWidthAndHeight(MyPoint topLeft, int width, int height) { }

record ColorAndRgbDTO(String name, int red, int green, int blue) { }

record PersonDTO(String firstname, String lastname, LocalDate birthday) { }



record Rectangle(int x, int y, int width, int height)
{
	Rectangle(TopLeftWidthAndHeight pointAndDimension)
	{
		this(pointAndDimension.topLeft().x, pointAndDimension.topLeft().y,
			pointAndDimension.width, pointAndDimension.height);
	}
}
```

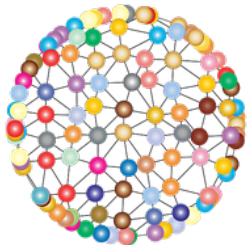
Records für komplexere Rückgabewerte und Parameter

```
record IntStringReturnValue(int code, String info) { }
record IntListReturnValue(int code, List<String> values) { }

record ReturnTuple(String first, String last, int amount) { }
record CompoundKey(String name, int age) { }


IntStringReturnValue calculateTheAnswer()
{
    // Some complex stuff here
    return new IntStringReturnValue(42, "the answer");
}

IntListReturnValue calculate(CompoundKey inputKey)
{
    // Some complex stuff here
    return new IntListReturnValue(201,
                                  List.of("This", "is", "a", "complex", "result"));
}
```

Records für Tupel? – Ausflug Pair<T>

- Was ist an diesem self made Pair falsch?

```
static class Pair<T>
{
	public T first;
	public T second;

	public Pair(T first, T second)
	{
		this.first = first;
		this.second = second;
	}

	@Override
	public String toString()
	{
	    return "Pair [first=" + first + ", second=" + second + "]";
	}
}
```



Was fehlt da eigentlich?
Was stört da vielleicht?

Records für Pairs und Tupel

```
record IntIntPair(int first, int second) {};

record StringIntPair(String name, int age) {};

record Pair<T1, T2>(T1 first, T2 second) {};

record Top3Favorites(String top1, String top2, String top3) {};

record CalcResultTuple(int min, int max, double avg, int count) {};
```

- **Exterm wenig Schreibaufwand**
- **Sehr praktisch für Pair, Tuples usw.**
- **Records funktionieren prima mit primitiven Typen und auch mit Generics**
- **Implementierungen von Accessor-Methoden sowie equals() und hashCode() automatisch und vor allem kontraktkonform**

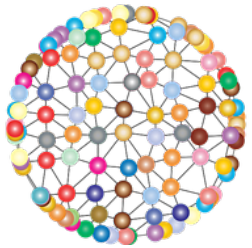
Ist ja cool … ABER: Wie kann ich denn Gültigkeitsprüfungen integrieren?

Records mit Gültigkeitsprüfung

```
record ClosedInterval(int lower, int upper)
{
    public ClosedInterval(int lower, int upper)
    {
        if (lower >= upper)
        {
            String errorMsg = String.format("invalid: %d (lower) >= %d (upper)",
                                            lower, upper);
            throw new IllegalArgumentException(errorMsg);
        }


        this.lower = lower;
        this.upper = upper;
    }
}
```

Records mit Gültigkeitsprüfung (Kurzschreibweise)

```
record ClosedInterval(int lower, int upper)
{
    public ClosedInterval
    {
        if (lower >= upper)
        {
            String errorMsg = String.format("invalid: %d (lower) >= %d (upper)",
                                            lower, upper);
            throw new IllegalArgumentException(errorMsg);
        }
    }
}
```



Letzte Frage für Records: Ist das alles kombinierbar?

All in Beispiel

```
record MultiTypes<K, V, T>(Map<K, V> mapping, T info)
{
    public void printTypes()
    {
        System.out.println("mapping type: " + mapping.getClass());
        System.out.println("info type: " + info.getClass());
    }
}


record ListRestrictions<T>(List<T> values, int maxSize)
{
    public ListRestrictions
    {
        if (values.size() > maxSize)
            throw new IllegalArgumentException(
                "too many entries! got: " + values.size() +
                ", but restricted to " + maxSize);
    }
}
```

Pattern Matching bei `instanceof`

Pattern Matching bei instanceof

- **ALT**

```
final Object obj = new Person("Michael", "Inden");
if (obj instanceof Person)
{
    final Person person = (Person) obj;
    // ... Zugriff auf person...
}
```

Immer diese Casts…
Geht es nicht einfacher?

Pattern Matching bei instanceof

- **ALT**

```
final Object obj = new Person("Michael", "Inden");
if (obj instanceof Person)
{
    final Person person = (Person) obj;
    // ... Zugriff auf person...
}
```

- **NEU**

```
if (obj instanceof Person person)
{
    // Hier kann man auf die Variable person direkt zugreifen
}
```

Pattern Matching bei instanceof

```java
Object obj2 = "Hallo Java 14";


if (obj2 instanceof String str)
{
    // Hier kann man str nutzen
    System.out.println("Länge: " + str.length());
}
else
{
    // Hier kein Zugriff auf str
    System.out.println(obj.getClass());
}
```

Pattern Matching bei instanceof

```
if (obj2 instanceof String str2 && str2.length() > 5)
{
    System.out.println("Länge: " + str2.length());
}
```

Lokale Enums und Interfaces

Lokale Enums und Interfaces

```
public class LocalEnumsAndInterfacesExamples
{
    public static void main(String[] args)
    {
        // Erst ab Java 15, vorher Compile-Error:
        // Local enums are not supported at language level '14'
        enum LocalEnumState
        {
            BAD, GOOD, UNKNOWN
        }

        // Erst ab Java 15, vorher Compile-Error:
        // Local interfaces are not supported at language level '14'
        interface Evaluationable
        {
            LocalEnumState evaluate(String info);
        }
        ...
```

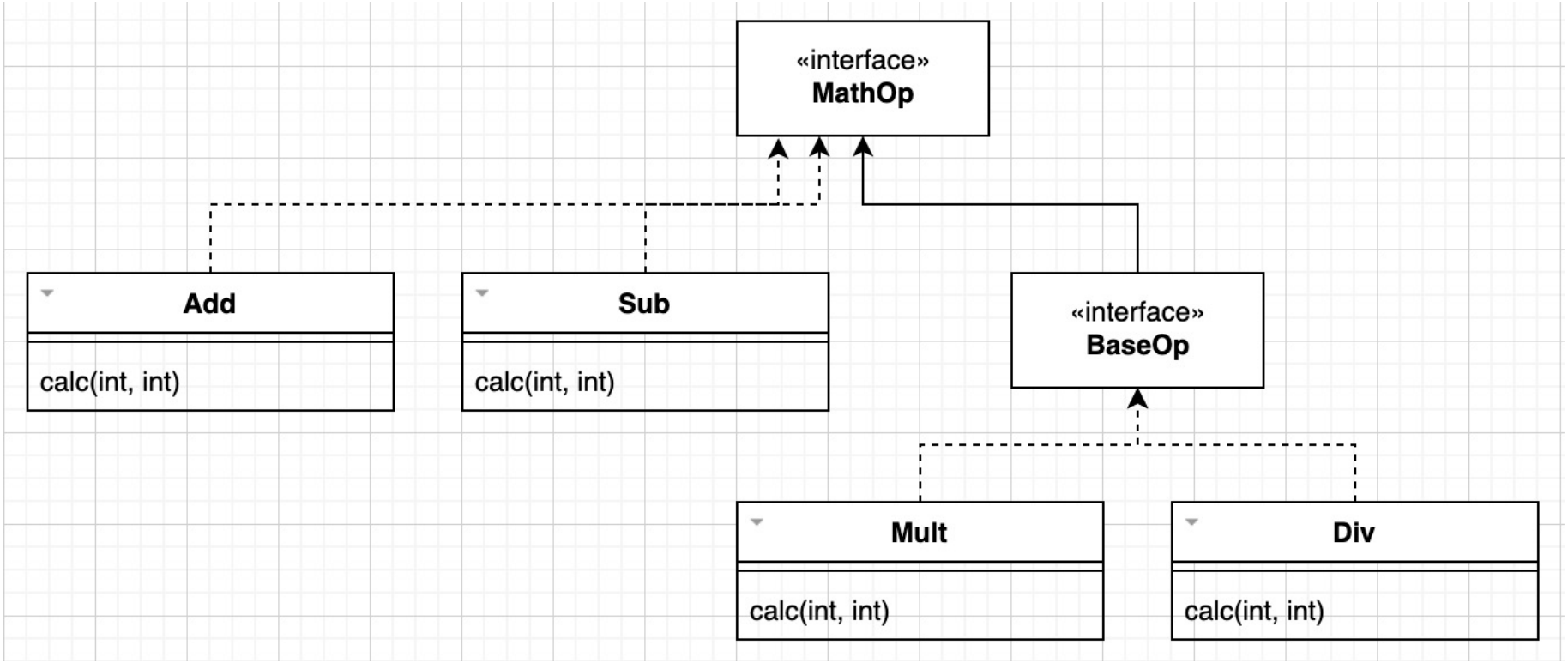
Lokale Enums und Interfaces

```
public class LocalEnumsAndInterfacesExamples
{

    ...

    class AlwaysBad implements Evaluationable
    {
        @Override
        public LocalEnumState evaluate(String info)
        {
            return LocalEnumState.BAD;
        }
    }

    System.out.println(new AlwaysBad().evaluate("DOES NOT MATTER"));
  }
}
```

Sealed Types

Sealed Types

- **Vererbung steuern** und spezifizieren, welche Klassen eine Basisklasse erweitern können, also welche anderen Klassen oder Interfaces davon Subtypen bilden dürfen.

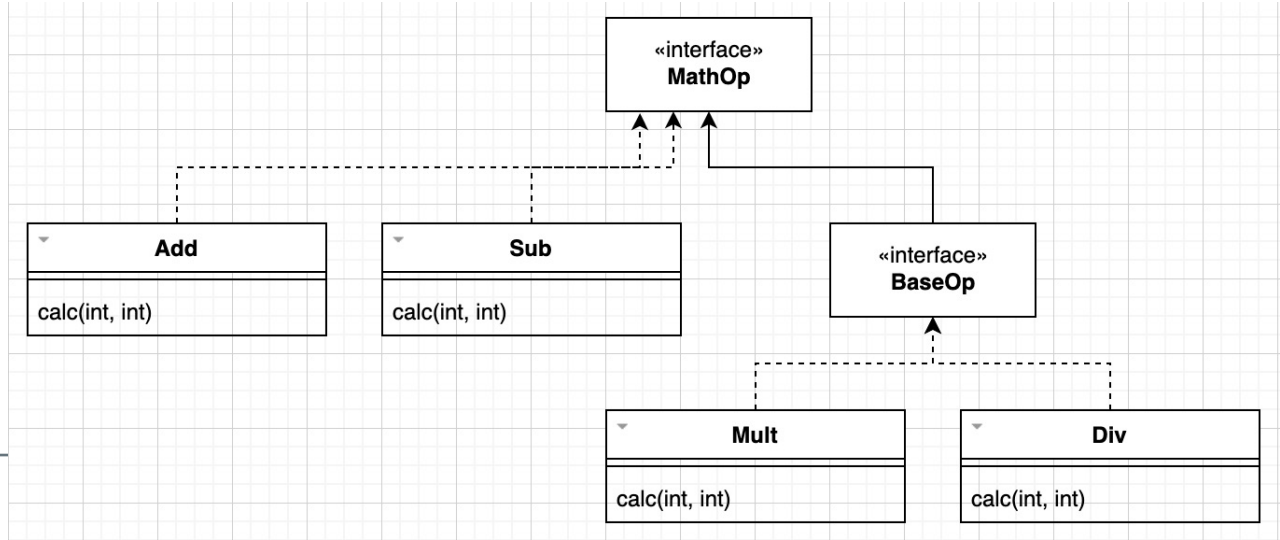


Sealed Types – Vererbung steuern

- **Spezifizieren, welche anderen Klassen oder Interfaces davon Subtypen bilden dürfen.**

```
public class SealedTypesExamples
{
    sealed interface MathOp
        permits BaseOp, Add, Sub  // <= erlaubte Subtypen
    {
        int calc(int x, int y);
    }

    // Mit non-sealed kann man innerhalb der Vererbungshierarchie Basisklassen bereitstellen
    non-sealed class BaseOp implements MathOp // <= Basisklasse nicht versiegeln
    {
        @Override
        public int calc(int x, int y)
        {
            return 0;
        }
    }
    ...
```

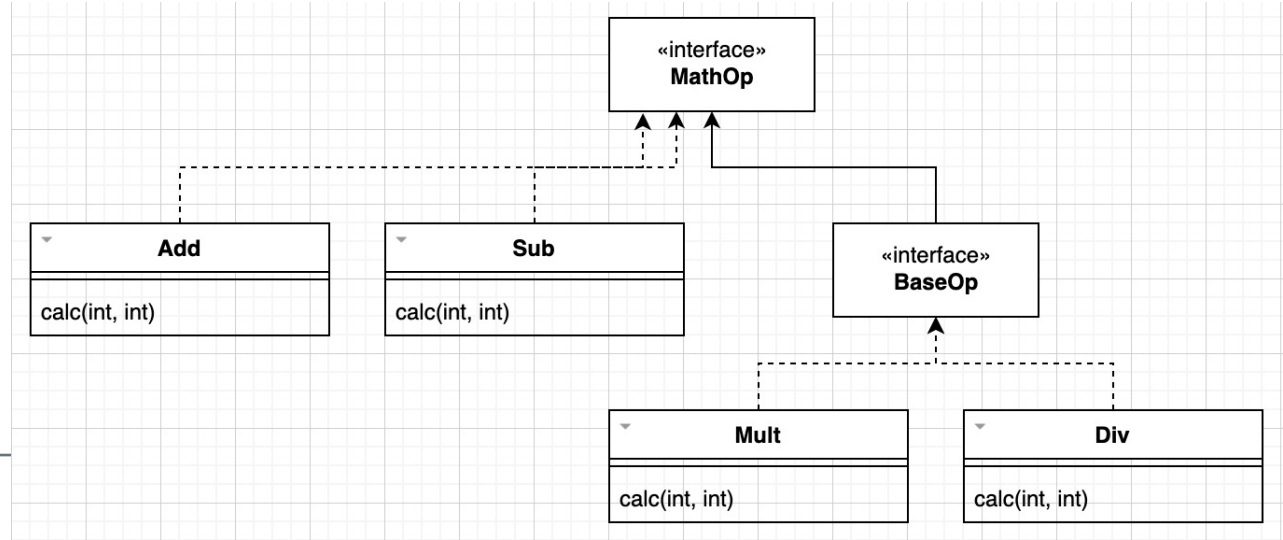
Mit `sealed` können wir eine Vererbungshierarchie versiegeln und nur die explizit angegebenen Typen erlauben. Diese müssen `sealed`, `non-sealed` oder `final` sein.



Sealed Types

```
..
   // direkte Implementierung muss final sein
   final class Add implements MathOp
   {
      @Override
      public int calc(int x, int y)
      {
         return x + y;
      }
   }
   final class Sub implements MathOp
   {
      ...
   }
   // Ableitung aus Basisklasse muss final sein
   final class Mult extends BaseOp
   {
   }
   final class Div extends BaseOp
   {
   }
}
```

- Eine als `sealed` markierte Klasse muss Subklassen besitzen, die wiederum hinter `permits` aufgeführt werden
- Eine als `non-sealed` markierte Klasse kann als Basisklasse fungieren und von dieser können Klassen abgeleitet werden.
- Eine als `final` markierte Klasse bildet – wie gewohnt – den Endpunkt einer Ableitungshierarchie.



Wissenswerts zu Sealed Types

- **festlegen, welche andere Klassen oder Interfaces davon Subtypen bilden dürfen.**
- **Sealed Types können bei der Entwicklung von Bibliotheken:**
  Verhalten über Interfaces exponieren, aber Kontrolle über mögliche Implementierungen
- **Sealed Types schränken bezüglich der Erweiterbarkeit von Klassenhierarchien ein und sollten daher** mit Bedacht verwendet **werden. Bei der Implementierung nicht vorhergesehene Flexibilität kann nachträglich störend sein.**

PART 2:
Neuerungen und Änderungen in den APIs und der JVM

- `CompactNumberFormat`
- `Teeing()-Kollektor`
- Tooling `jpackage`

Erweiterung CompactNumberFormat

Utility-Klasse CompactNumberFormat

- **CompactNumberFormat** ist eine Subklasse von **NumberFormat**
- Formattiert eine Dezimalzahl in kompakter Schreibweise, also 10K statt 10.000
- Beachtet Locales
- **Es gibt zwar nen Konstruktor, aber einfacher durch Factory-Methode**

```
NumberFormat compactFormat =
             NumberFormat.getCompactNumberInstance(Locale.US,
                                                   NumberFormat.Style.SHORT);
```

CompactNumberformat Style

```
public static void main(final String args[])
{
    var shortFormat = getUsCompactNumberFormat(NumberFormat.Style.SHORT);
    formatNumbers("SHORT", shortFormat);

    var longFormat = getUsCompactNumberFormat(NumberFormat.Style.LONG);
    formatNumbers("LONG", longFormat);
}

private static NumberFormat getUsCompactNumberFormat(NumberFormat.Style style)
{
    return NumberFormat.getCompactNumberInstance(Locale.US, style);
}

private static void formatNumbers(final String style,
                                  final NumberFormat shortFormat)
{
    System.out.println("\nNumberFormat " + style);
    System.out.println("Result: " + shortFormat.format(10_000));
    System.out.println("Result: " + shortFormat.format(123_456));
    System.out.println("Result: " + shortFormat.format(1_234_567));
    System.out.println("Result: " + shortFormat.format(1_950_000_000));
}
```

CompactNumberformat Style

```
public static void main(final String args[])
{
    var shortFormat = getUsCompactNumberFormat(NumberFormat.Style.SHORT);
    formatNumbers("SHORT", shortFormat);

    var longFormat = getUsCompactNumberFormat(NumberFormat.Style.LONG);
    formatNumbers("LONG", longFormat);
}

private static NumberFormat getUsCompactNumberFormat(NumberFormat.Style style)
{
    return NumberFormat.getCompactNumberInstance(Locale.US, style);
}

private static void formatNumbers(final String style,
                                  final NumberFormat shortFormat)
{
    System.out.println("\nNumberFormat " + style);
    System.out.println("Result: " + shortFormat.format(10_000));
    System.out.println("Result: " + shortFormat.format(123_456));
    System.out.println("Result: " + shortFormat.format(1_234_567));
    System.out.println("Result: " + shortFormat.format(1_950_000_000));
}
```

```
NumberFormat SHORT
Result: 10K
Result: 123K
Result: 1M
Result: 2B


NumberFormat LONG
Result: 10 thousand
Result: 123 thousand
Result: 1 million
Result: 2 billion
```

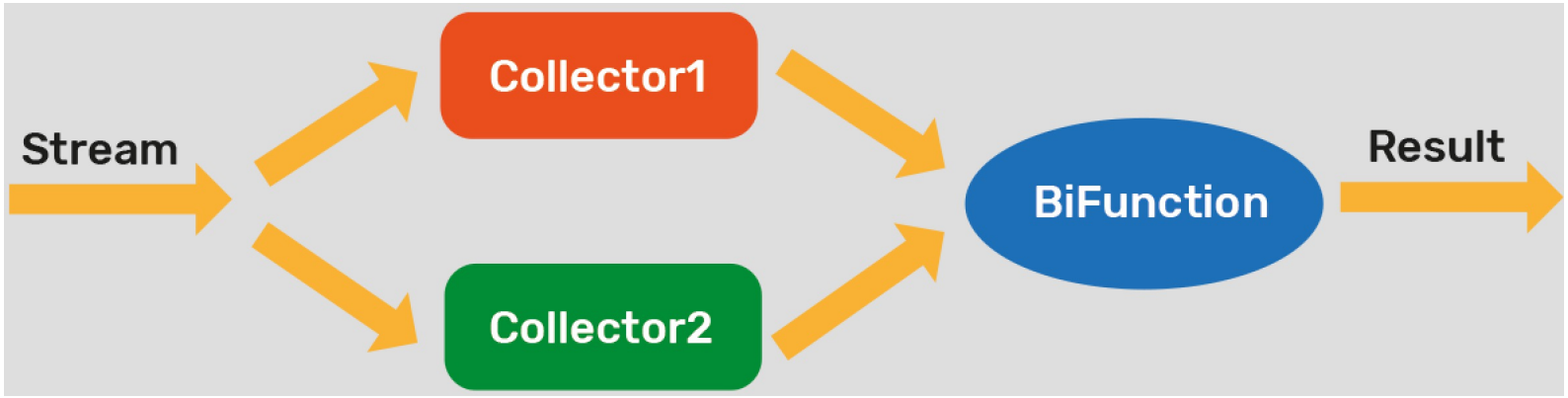
Teeing()-Kollektor

Kollektoren

Wie sieht es denn mit dem Zusammenfassen von Streams aus?

"...returns a Collector that is a composite of two downstream collectors. Every element passed to the resulting collector is processed by both downstream collectors, then their results are merged using the specified merge function into the final result."

```
static <T, R1, R2, R> Collector<T, ?, R> teeing(Collector<? super T, ?, R1> downstream1,
                                                Collector<? super T, ?, R2> downstream2,
                                                BiFunction<? super R1, ? super R2, R> merger)
```



Kollektoren

```
public static void main(String[] args)
{
	var firstSixNumbers = Stream.of(1, 2, 3, 4, 5, 6);
	System.out.println(calcCountAndSum(firstSixNumbers));

	var primeNumbers = Stream.of(2, 3, 5, 7, 11, 13, 17);
	System.out.println(calcCountAndSum(primeNumbers));
}

private static Pair<Long> calcCountAndSum(Stream<Integer> numbers)
{
	return numbers.collect(Collectors.teeing(
				Collectors.counting(),
				Collectors.summingLong(n -> n),
				// (count, sum) -> new Pair<Long>(count, sum))
				Pair<Long>::new));
}
```

Kollektoren

```
public static void main(String[] args)
{
        var firstSixNumbers = Stream.of(1, 2, 3, 4, 5, 6);
        System.out.println(calcCountAndSum(firstSixNumbers));

        var primeNumbers = Stream.of(2, 3, 5, 7, 11, 13, 17);
        System.out.println(calcCountAndSum(primeNumbers));
}

private static Pair<Long> calcCountAndSum(Stream<Integer> numbers)
{
        return numbers.collect(Collectors.teeing(
                                        Collectors.counting(),
                                        Collectors.summingLong(n -> n),
                                        Pair<Long>::new));
}
```

```
Pair<T> [first=6, second=21]
Pair<T> [first=7, second=58]
```

Ausflug Pair<T> (Simpelste Form, nicht allgemeingültig)

```
static class Pair<T>
{
        public T first;
        public T second;

        public Pair(T first, T second)
        {
            this.first = first;
            this.second = second;
        }

        @Override
        public String toString()
        {
            return "Pair [first=" + first + ", second=" + second + "]";
        }
}
```

Wie wäre es mit Map.Entry als Ersatz für ein eigenes Pair?

NICHT SO GUT!!

Warum? Map.Entry ist für Maps gedacht und sollte nur in deren Kontext genutzt werden. In modernem Java besser Records nutzen!

Kollektoren

Aufgabe: Aus einem Stream von Strings, sollen unterschiedliche Elemente herausgefiltert und dann die Ergebnisse kombiniert werden.

- **Hier hilft der `filtering()`-Kollektor aus Java 9 sowie `teeing()`:**

```
var names = Stream.of("Michael", "Tim", "Tom", "Mike", "Bernd");

final Predicate<String> startsWithMi = text -> text.startsWith("Mi");
final Predicate<String> endsWithM = text -> text.endsWith("m");
final BiFunction<List<String>, List<String>, List<List<String>>> combineLists =
                                        (list1, list2) -> List.of(list1, list2);

var result = names.collect(teeing(filtering(startsWithMi, toList()),
                                  filtering(endsWithM, toList()),
                                  // (list1, list2) -> List.of(list1, list2)
                                  combineLists));
System.out.println(result);
```

Kollektoren

Aufgabe: Aus einem Stream von Strings, sollen unterschiedliche Elemente herausgefiltert und dann die Ergebnisse kombiniert werden.

```
var names = Stream.of("Michael", "Tim", "Tom", "Mike", "Bernd");

final Predicate<String> startsWithMi = text -> text.startsWith("Mi");
final Predicate<String> endsWithM = text -> text.endsWith("m");

var result = names.collect(teeing(filtering(startsWithMi, toList()),
                                  filtering(endsWithM, toList()),
                                  (list1, list2) -> List.of(list1, list2));
System.out.println(result);
```

[Michael, Mike]

Kollektoren

Aufgabe: Aus einem Stream von Strings, sollen unterschiedliche Elemente herausgefiltert und dann die Ergebnisse kombiniert werden.

```
var names = Stream.of("Michael", "Tim", "Tom", "Mike", "Bernd");

final Predicate<String> startsWithMi = text -> text.startsWith("Mi");
final Predicate<String> endsWithM = text -> text.endsWith("m");

var result = names.collect(teeing(filtering(startsWithMi, toList()),
                                  filtering(endsWithM, toList()),
                                  (list1, list2) -> List.of(list1, list2));
System.out.println(result);
```

[Michael, Mike]

[Tim, Tom]

Kollektoren

Aufgabe: Aus einem Stream von Strings, sollen unterschiedliche Elemente herausgefiltert und dann die Ergebnisse kombiniert werden.

```
var names = Stream.of("Michael", "Tim", "Tom", "Mike", "Bernd");

final Predicate<String> startsWithMi = text -> text.startsWith("Mi");
final Predicate<String> endsWithM = text -> text.endsWith("m");

var result = names.collect(teeing(filtering(startsWithMi, toList()),
                                  filtering(endsWithM, toList()),
                                  (list1, list2) -> List.of(list1, list2));

System.out.println(result);
```

[Michael, Mike]

[Tim, Tom]

[[Michael, Mike], [Tim, Tom]]

Java 16

Stream => List ... es war so umständlich ...

```
List<String> namesMi = Stream.of("Tim", "Tom", "Mike", "Michael").
                              filter(str -> str.startsWith("Mi")).
                              collect(Collectors.toList());
```

FINALLY… toList()

```
List<String> namesMi = Stream.of("Tim", "Tom", "Mike", "Michael").
                              filter(str -> str.startsWith("Mi")).
                              toList();
```

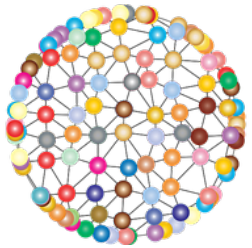
mapMulti()

Warum `mapMulti()`?

```
// OLD
Stream.of(Optional.of("0"), Optional.empty(), Optional.of("1"),
          Optional.empty(), Optional.of("2"), Optional.empty(), Optional.of("3")).
       flatMap(Optional::stream).
       forEach(System.out::print);


// NEW
Stream.of(Optional.of("0"), Optional.empty(), Optional.of("1"),
          Optional.empty(), Optional.of("2"), Optional.empty(), Optional.of("3")).
       mapMulti(Optional::ifPresent). // !!!
       forEach(System.out::print);
```

Warum `mapMulti()`?

```
// NEW
Stream.of(Optional.of("0"), Optional.empty(), Optional.of("1"),
          Optional.empty(), Optional.of("2"), Optional.empty(), Optional.of("3")).
       mapMulti(Optional::ifPresent). // !!!
       forEach(System.out::print);


// Mehr der Gedanke wie imperative for-Schleife
var elements = List.of(Optional.of("0"), Optional.empty(), Optional.of("1"),
                       Optional.empty(), Optional.of("2"), Optional.empty(), Optional.of("3"));
for (Optional optElem : elements )
{
    optElem.ifPresent(System.out::print);
}
```



Wie bauen wir das?

```
Stream.of(List.of(1, 2, 3),
          "ABC", null,
          Set.of("X", "Y", "Z"));
```

=>

```
[1, 2, 3, ABC, null, Z, Y, X]
```

Warum `mapMulti()`? Beispiel `expandIterables()`

```
// OLD
var stream = Stream.of(List.of(1, 2, 3), "ABC", null, Set.of("X", "Y", "Z"));


Stream<Object> expandedStream = stream.flatMap(e -> {
    if (e instanceof Iterable<?> iterable) {
        // Trick to convert Iterable to Stream
        return StreamSupport.stream(iterable.spliterator(), false);
    }
    return Stream.of(e);
});


System.out.println(expandedStream.toList());
```

```
[1, 2, 3, ABC, null, Z, Y, X]
```

Warum mapMulti()? Beispiel expandIterables()

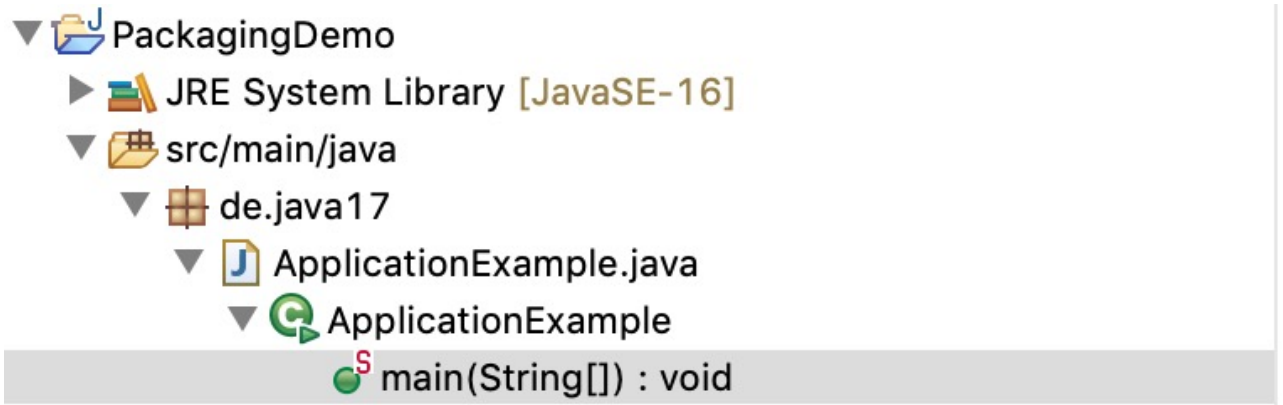
```
// NEW
var stream2 = Stream.of(List.of(1, 2, 3), "ABC", null, Set.of("X", "Y", "Z"));

Stream<Object> expandedStream2 = stream2.mapMulti(MapMultiExample2::expandIterable);

System.out.println(expandedStream2.toList());


static void expandIterable(Object e, Consumer<Object> c) {
    if (e instanceof Iterable<?> iterable) {
        for (Object elem : iterable) {
            expandIterable(elem, c);
        }
    } else {
        c.accept(e);
    }
}
```

```
[1, 2, 3, ABC, null, Z, Y, X]
```

~~Nashorn Java Script Engine~~

~~(seit Java 11 deprecated, mit Java 15 entfernt)~~

JPackage




```java
public class ApplicationExample {

    public static void main(String[] args) {

        System.out.println("First JPackage");

        JOptionPane.showConfirmDialog(null, "Generated by jpackage", "DEMO",
                JOptionPane.OK_CANCEL_OPTION, JOptionPane.INFORMATION_MESSAGE, null)
    }
}
```

Wie bauen wir das?

- PackagingDemo
  - JRE System Library [JavaSE-16]
  - src/main/java
    - de.java17
      - ApplicationExample.java
        - ApplicationExample
          - main(String[]) : void
  - src
  - build.gradle
  - pom.xml

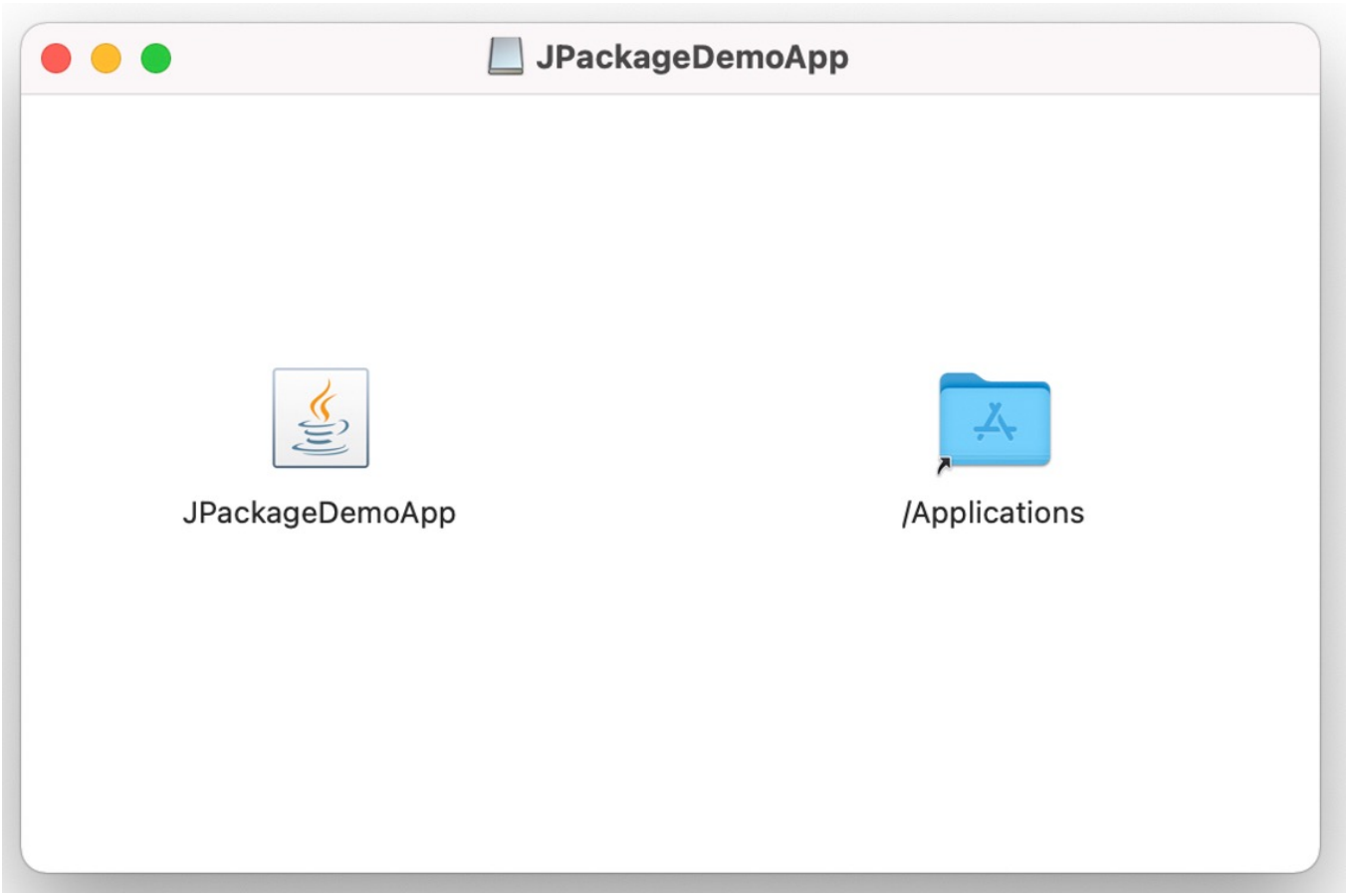
Maven™

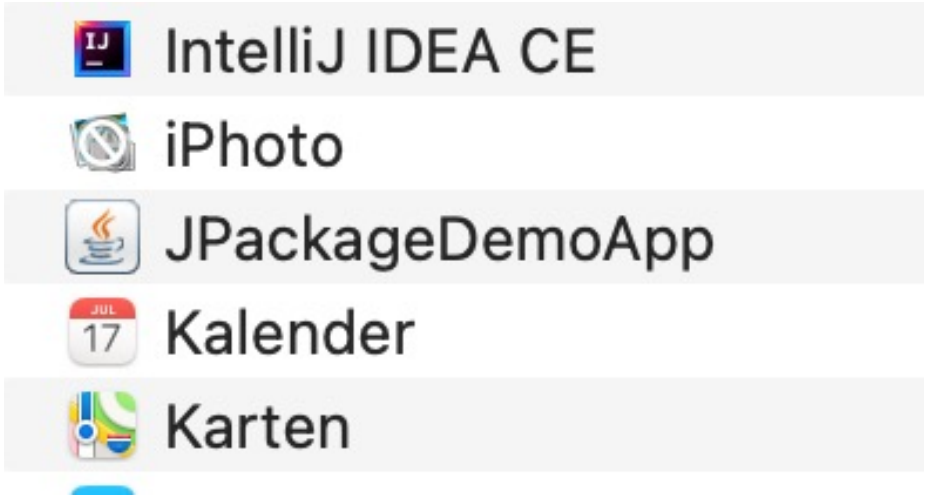
Gradle

```
mvn clean install
```

- target
  - classes
  - generated-sources
  - maven-archiver
  - maven-status
  - PackagingDemoExamples-1.0.0-SNAPSHOT.jar

```
jpackage --input target/ --name JPackageDemoApp --main-jar PackagingDemoExamples-1.0.0-SNAPSHOT.jar --main-class de.java17.ApplicationExample --type dmg --java-options '--enable-preview'
```





Aber was machen wir mit 3rd Party Libraries?

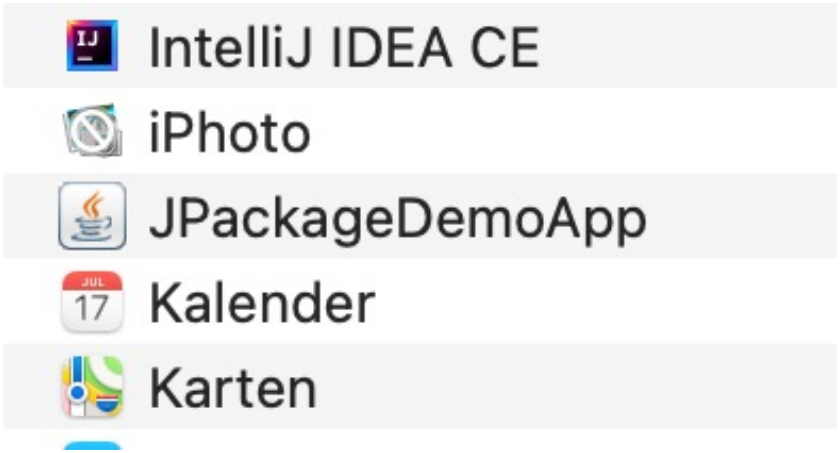
```java
public class ApplicationExample {

    public static void main(String[] args) {
        System.out.println("First JPackage");

        Joiner joiner = Joiner.on(":");
        String result = joiner.join(List.of("Michael", "mag", "Python"));
        System.out.println(result);
        JOptionPane.showConfirmDialog(null, result);
    }
}
```

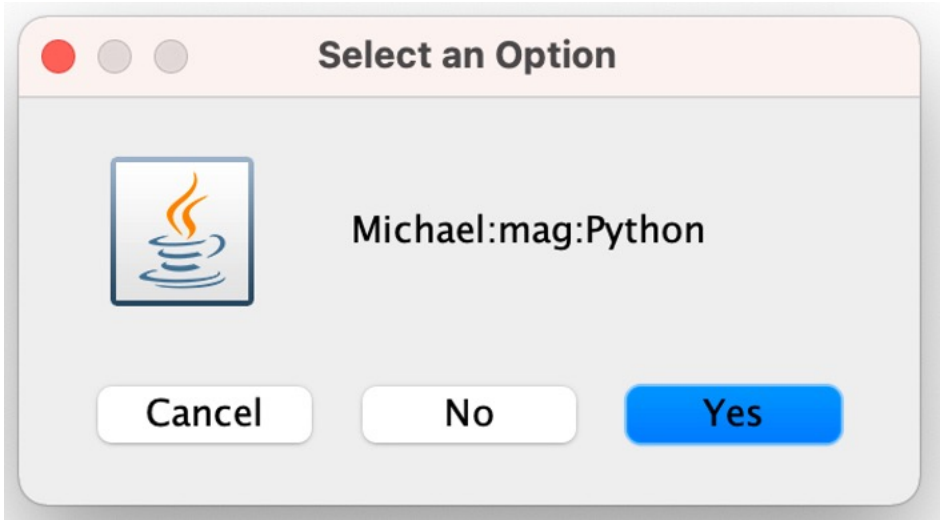
```
<!-- https://mvnrepository.com/artifact/com.google.guava/guava -->
<dependency>
<groupId>com.google.guava</groupId>
<artifactId>guava</artifactId>
<version>31.0.1-jre</version>
</dependency>
```

Wie wird die Bibliothek in den Anwendung eingebunden?

```
<plugin>
    <artifactId>maven-dependency-plugin</artifactId>
    <executions>
        <execution>
            <phase>process-sources</phase>
            <goals>
                <goal>copy-dependencies</goal>
            </goals>

            <configuration>
                <outputDirectory>target</outputDirectory>
                <includeScope>runtime</includeScope>
                <excludeScope>test</excludeScope>
            </configuration>
        </execution>
    </executions>
</plugin>
```

- target
  - classes
  - generated-sources
  - maven-archiver
  - maven-status
  - checker-qual-3.12.0.jar
  - error_prone_annotations-2.7.1.jar
  - failureaccess-1.0.1.jar
  - guava-31.0.1-jre.jar
  - j2objc-annotations-1.3.jar
  - jsr305-3.0.2.jar
  - listenablefuture-9999.0-empty-to-avoid-conflict-with-guav
  - PackagingDemoExamples-1.0.0-SNAPSHOT.jar


IntelliJ IDEA CE
iPhoto
JPackageDemoApp
Kalender
Karten


Select an Option
Michael:mag:Python
Cancel
No
Yes

DEMO & Hands on

Recap: Neuerungen in Java 17 im Überblick

- Überblick: Was ist drin?
- Syntaxerweiterungen bei `switch (PREVIEW)`

Java 17 – Was ist drin?

- JEP 306: Restore Always-Strict Floating-Point Semantics
- JEP 356: Enhanced Pseudo-Random Number Generators
- JEP 382: New macOS Rendering Pipeline
- JEP 391: macOS/AArch64 Port
- JEP 398: Deprecate the Applet API for Removal
- JEP 403: Strongly Encapsulate JDK Internals
- JEP 406: Pattern Matching for switch (Preview)
- JEP 407: Remove RMI Activation
- JEP 409: Sealed Classes
- JEP 410: Remove the Experimental AOT and JIT Compiler
- JEP 411: Deprecate the Security Manager for Removal
- JEP 412: Foreign Function & Memory API (Incubator)
- JEP 414: Vector API (Second Incubator)
- JEP 415: Context-Specific Deserialization Filters

Was ist davon wirklich wichtig für uns?

- JEP 406: Pattern Matching for switch (Preview)
- ?? JEP 409: Sealed Classes ??
- ?? JEP 391: macOS/AArch64 Port ??

Versionsüberblick – https://javaalmanac.io/

The Java Version Almanac

Collection of information about the history and future of Java.

Details	Status	Documentation	Download	Compare API to										
Java 18	DEV	API | Notes	JDK | JRE	17	16	15	14	13	12	11	10	9	8	...
Java 17	LTS	API | Lang | VM | Notes	JDK | JRE	16	15	14	13	12	11	10	9	8	7	...
Java 16	EOL	API | Lang | VM | Notes	JDK | JRE	15	14	13	12	11	10	9	8	7	6	...
Java 15	EOL	API | Lang | VM | Notes	JDK | JRE	14	13	12	11	10	9	8	7	6	5	...
Java 14	EOL	API | Lang | VM | Notes	JDK | JRE	13	12	11	10	9	8	7	6	5	1.4	...
Java 13	EOL	API | Lang | VM | Notes	JDK | JRE	12	11	10	9	8	7	6	5	1.4	1.3	...
Java 12	EOL	API | Lang | VM | Notes	JDK | JRE	11	10	9	8	7	6	5	1.4	1.3	1.2	...
Java 11	LTS	API | Lang | VM | Notes	JDK | JRE	10	9	8	7	6	5	1.4	1.3	1.2	1.1	
Java 10	EOL	API | Lang | VM | Notes	JDK | JRE	9	8	7	6	5	1.4	1.3	1.2	1.1		
Java 9	EOL	API | Lang | VM | Notes	JDK | JRE	8	7	6	5	1.4	1.3	1.2	1.1			
Java 8	LTS	API | Lang | VM | Notes	JDK | JRE	7	6	5	1.4	1.3	1.2	1.1				
Java 7	EOL	API | Lang | VM | Notes	JDK | JRE	6	5	1.4	1.3	1.2	1.1					
Java 6	EOL	API | Lang | VM | Notes	JDK | JRE	5	1.4	1.3	1.2	1.1						
Java 5	EOL	API | Lang | VM | Notes		1.4	1.3	1.2	1.1							
Java 1.4	EOL	API		1.3	1.2	1.1								
Java 1.3	EOL	API		1.2	1.1									
Java 1.2	EOL	API | Lang		1.1										
Java 1.1	EOL	API												
Java 1.0	EOL	API | Lang | VM												

Data Source

Pattern Matching bei `switch` (PREVIEW)

Pattern Matching bei `switch`

- Mit Java 16 wurde Pattern Matching für `instanceof` eingeführt. Damit ist es möglich, ein sogenanntes Typmuster zu akzeptieren und einen Mustervergleich durchzuführen. Dadurch kann man sich lästige Casts sparen.

- Diese Syntax-Neuerung ist mit Java 17 auch für `switch` möglich

- Allerdings ist das Ganze zunächst in Form von Preview Features umgesetzt und zum Nachvollziehen müssen Sie diese geeignet aktivieren, etwa wie folgt in der JShell:

```
$ jshell --enable-preview
|  Welcome to JShell -- Version 17-ea
|  For an introduction type: /help intro
```

Pattern Matching bei `switch`

- Nehmen wir an, wir müssten eine generische Methode zum Formatieren von Werten schreiben. Sofern wir Java 16 nutzen, können wir dies mithilfe von Pattern Matching und `instanceof` einigermaßen lesbar wie folgt schreiben -- das Ganze wäre ohne Java 16 deutlich schlimmer, da wir für jeden Typ eine Zeile mit einem Cast ergänzen müssten:

```
static String formatterJdk16instanceof(Object obj) {
    String formatted = "unknown";
    if (obj instanceof Integer i) {
        formatted = String.format("int %d", i);
    } else if (obj instanceof Long l) {
        formatted = String.format("long %d", l);
    } else if (obj instanceof Double d) {
        formatted = String.format("double %f", d);
    } else if (obj instanceof String s) {
        formatted = String.format("String %s", s);
    }
    return formatted;
}
```

Pattern Matching bei `switch`

- Bislang war es nicht möglich, in den `cases` eines `switchs` den Wert `null` zu behandeln, sondern dazu war folgende Spezialbehandlung nötig:

```
static void switchSpecialNullSupport(String str) {
    if (str == null) {
        System.out.println("special handling for null");
        return;
    }

    switch (str) {
        case "Java", "Python" -> System.out.println("cool language");
        default -> System.out.println("everything else");
    }
}
```

Pattern Matching bei `switch`

- **Mit Java 17 und aktivierten Preview Features können wir nun auch `null`-Werte angeben und das ganze Konstrukt vereinheitlichen:**

```
static void switchSupportingNull(String str) {
    switch (str) {
        case null -> System.out.println("null is allowed in preview");
        case "Java", "Python" -> System.out.println("cool language");
        default -> System.out.println("everything else");
    }
}
```

Pattern Matching bei `switch`

- Analog zu `instanceof` sind in den `cases` nun auch Abfragen wie die folgenden möglich:

```
static void processData(Object obj) {
    switch (obj) {
        case String str && str.startsWith("V1") -> System.out.println("Processing V1");
        case String str && str.startsWith("V2") -> System.out.println("Processing V2");
        case Integer i && i > 10 && i < 100 -> System.out.println("Processing ints");
        default -> throw new IllegalArgumentException("invalid input");
    }
}
```

Deprecations

Deprecations

- **Konstruktion von Wrappern:**

```
/Users/michaelinden/java8-
workspace/Java17Examples/src/main/java/primitives/PrimitiveCtorDeprecati
onExample.java:6: warning: [removal] Long(long) in Long has been
deprecated and marked for removal
                Long myLong = new Long(1234L);
                              ^
/Users/michaelinden/java8-
workspace/Java17Examples/src/main/java/primitives/PrimitiveCtorDeprecati
onExample.java:8: warning: [removal] Integer(int) in Integer has been
deprecated and marked for removal
                Integer myInt = new Integer(1234);
                                ^
```

PART 3:
Neuerungen in Java 18

Java 18 – Was ist drin?

- JEP 400: UTF-8 by Default
- **JEP 408: Simple Web Server**
- **JEP 413: Code Snippets in Java API Documentation**
- JEP 416: Reimplement Core Reflection with Method Handles
- JEP 417: Vector API (Third Incubator)
- **JEP 418: Internet-Address Resolution SPI**
- JEP 419: Foreign Function & Memory API (Second Incubator)
- **JEP 420: Pattern Matching for switch (Second Preview)**
- **JEP 421: Deprecate Finalization for Removal**

Versionsüberblick – https://javaalmanac.io/

Feedback on this page?

The Java Version Almanac

Collection of information about the history and future of Java.

Details	Status	Documentation	Download	Compare API to										
Java 19	DEV	API | Notes	JDK | JRE	18	17	16	15	14	13	12	11	10	9	...
Java 18	DEV	API | Notes	JDK | JRE	17	16	15	14	13	12	11	10	9	8	...
Java 17	LTS	API | Lang | VM | Notes	JDK | JRE	16	15	14	13	12	11	10	9	8	7	...
Java 16	EOL	API | Lang | VM | Notes	JDK | JRE	15	14	13	12	11	10	9	8	7	6	...
Java 15	EOL	API | Lang | VM | Notes	JDK | JRE	14	13	12	11	10	9	8	7	6	5	...
Java 14	EOL	API | Lang | VM | Notes	JDK | JRE	13	12	11	10	9	8	7	6	5	1.4	...
Java 13	EOL	API | Lang | VM | Notes	JDK | JRE	12	11	10	9	8	7	6	5	1.4	1.3	...
Java 12	EOL	API | Lang | VM | Notes	JDK | JRE	11	10	9	8	7	6	5	1.4	1.3	1.2	...
Java 11	LTS	API | Lang | VM | Notes	JDK | JRE	10	9	8	7	6	5	1.4	1.3	1.2	1.1	
Java 10	EOL	API | Lang | VM | Notes	JDK | JRE	9	8	7	6	5	1.4	1.3	1.2	1.1		
Java 9	EOL	API | Lang | VM | Notes	JDK | JRE	8	7	6	5	1.4	1.3	1.2	1.1			
Java 8	LTS	API | Lang | VM | Notes	JDK | JRE	7	6	5	1.4	1.3	1.2	1.1				
Java 7	EOL	API | Lang | VM | Notes	JDK | JRE	6	5	1.4	1.3	1.2	1.1					
Java 6	EOL	API | Lang | VM | Notes	JDK | JRE	5	1.4	1.3	1.2	1.1						
Java 5	EOL	API | Lang | VM | Notes		1.4	1.3	1.2	1.1							
Java 1.4	EOL	API		1.3	1.2	1.1								
Java 1.3	EOL	API		1.2	1.1									
Java 1.2	EOL	API | Lang		1.1										
Java 1.1	EOL	API												
Java 1.0	EOL	API | Lang | VM												

Data Source

Sandbox – https://javaalmanac.io/

Java 18

Status	In Development
Release Date	2022-03-15
EOL Date	2022-09-15
Bytecode Version	62.0
API Changes	Compare to **17** - **16** - **15** - **14** - **13** - **12** - **11** - **10** - **9** - **8** - **7** - **6** - **5** - **1.4** - **1.3** - **1.2** - **1.1**
Documentation	**Release Notes**, **JavaDoc**
SCM	**git**

Data Source

New Features

JVM

- UTF-8 by Default (**JEP 400**)

Language

- Pattern Matching for switch (Second Preview) Preview (**JEP 420**)

API

- Reimplement Core Reflection with Method Handles (**JEP 416**)
- Internet-Address Resolution SPI (**JEP 418**)
- Deprecate Finalization for Removal (**JEP 421**)

Tools

- Simple Web Server (**JEP 408**)
- Code Snippets in Java API Documentation (**JEP 413**)

Sandbox – https://javaalmanac.io/

Sandbox

Instantly compile and run Java 18 snippets without a local Java installation.

Java18.java ▶ Run 18+36-2087

```
import javax.lang.model.SourceVersion;

public class Java18 {

    public static void main(String[] args) {
        System.out.println("Runtime required for this: " + SourceVersion.RELEASE_18.runtimeVersion());

        switchSupportingNull(null);
        switchSupportingNull("JAVA");
    }

    static void switchSupportingNull(String str) {
        switch (str.toUpperCase()) {
        case null -> System.out.println("null is allowed in preview");
        case "Java", "Python" -> System.out.println("cool language");
        default -> System.out.println("everything else");
        }
    }
}
```

Sandbox – https://javaalmanac.io/

Sandbox

Instantly compile and run Java 18 snippets without a local Java installation.

Java18.java ▶ Run 18+36-2087

```
Java18.java:14: warning: [preview] null in switch cases is a preview feature and may be removed in a future r
                case null -> System.out.println("null is allowed in preview");
                     ^
Runtime required for this: 18
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.toUpperCase()" because "<par
        at Java18.switchSupportingNull(Java18.java:13)
        at Java18.main(Java18.java:8)
```

JEP 408: Simple Web Server

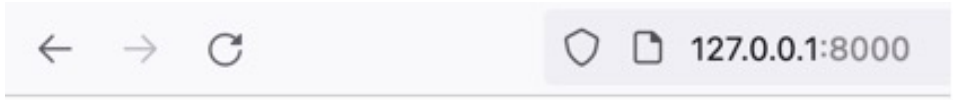
JEP 408: Simple Web Server

- **Ziel bei diesem JEP war es, einen minimalistischen Webserver ohne viel Konfiguration ins JDK zu integrieren, der lediglich statische Dateien bereitstellen kann. Dies erleichtert die Erstellung von Prototypen, das File Sharing oder kann auch für einfache Tests hilfreich sein.**

```
michaelinden@MBP-von-Michael ~ % jwebserver
Binding to loopback by default. For all interfaces use "-b 0.0.0.0" or "-b ::".
Serving /Users/michaelinden and subdirectories on 127.0.0.1 port 8000
URL http://127.0.0.1:8000/
```

```
jshell> import com.sun.net.httpserver.SimpleFileServer;
   ...> import com.sun.net.httpserver.SimpleFileServer.OutputLevel;

jshell> var server = SimpleFileServer.createFileServer(new
InetSocketAddress(8000), Path.of("/Users/michaelinden"), OutputLevel.VERBOSE)
server ==> sun.net.httpserver.HttpServerImpl@6659c656

jshell> server.start()
```

JEP 408: Simple Web Server



Directory listing for /

- Music/
- JavaDateAndTimeOverview_RO.asta
- spring-boot-test-workspace/
- JDK_SET.txt
- spacerocks.dat
- java8-workspace/
- h2
- spring-boot-persistence-data-workspace/
- Pictures/
- zshrc.textClipping
- JavaDateAndTimeOvervie_IFs.png
- Desktop/
- Library/
- eclipse-workspace/
- mongodb-macos-x86_64-5.0.3/
- spring-framework-intro-workspace/
- Sites/
- splash.png
- my.properties
- PycharmProjects/

DEMO & Hands on

JEP 413: Code Snippets in Java API Documentation

JEP 413: Code Snippets in Java API Documentation

- **Bislang gibt es keinen Standard, um Codefragmente in die mit Java-Doc generiert HTML-Dokumentation aufzunehmen. Im Rahmen dieses JEPs wird das Inline-Tag `@snippet` eingeführt. Damit lassen sich Codefragmente in einen Java-Doc-Kommentar einfügen:**

```
/**
 * The following code shows how to use {@code Optional.isPresent}:
 * {@snippet :
 * if (optValue.isPresent())
 * {
 *     System.out.println("value: " + optValue.get());
 * }
 * }
 */
public static void method1(String[] args) {
    //
    //
    //
    //
}
```

void java18.misc.JavaDocExample.method1(String[] args)

The following code shows how to use `Optional.isPresent`:

```
if (optValue.isPresent())
{
    System.out.println("value: " + optValue.get());
}
```

JEP 418: Internet-Address Resolution SPI

JEP 418: Internet-Address Resolution SPI

- Im Rahmen dieses JEP wurde ein SPI (Service Provider Interface) für die Auflösung von Host- und Namensadressen implementiert. Die Adressauflösung verwendet bislang eine hartcodierte Umsetzung innerhalb der Klasse `java.net.InetAddress`.

```
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Arrays;

public class InetAddressExmaple
{
    public static void main(String[] args) throws UnknownHostException
    {
        InetAddress[] addresses = InetAddress.getAllByName("www.dpunkt.de");
        System.out.println("addresses = " + Arrays.toString(addresses));
    }
}
```

JEP 418: Internet-Address Resolution SPI

```
public class OwnInetAddressResolver implements InetAddressResolver {
    @Override
    public Stream<InetAddress> lookupByName(String host, LookupPolicy lookupPolicy)
                               throws UnknownHostException {

        if (host.equals("www.dpunkt.de"))
            return Stream.of(InetAddress.getByAddress(new byte[] { 127, 0, 0, 1 }));

        throw new UnsupportedOperationException();
    }

    @Override
    public String lookupByAddress(byte[] addr) {
        throw new UnsupportedOperationException();
    }
}
```

JEP 418: Internet-Address Resolution SPI

```
import java.net.spi.InetAddressResolver;
import java.net.spi.InetAddressResolverProvider;

public class OwnInetAddressResolverProvider extends InetAddressResolverProvider
{
    @Override
    public InetAddressResolver get(Configuration configuration) {
        return new OwnInetAddressResolver();
    }

    @Override
    public String name() {
        return "Own Internet Address Resolver Provider";
    }
}
```

DEMO & Hands on

JEP 420: Pattern Matching bei `switch` (PREVIEW II)

JEP 420: Pattern Matching bei `switch`

- **JEP 420 führt zu zwei Änderungen bei der Auswertung der `case` innerhalb `switch` beim Kompilieren:**
  - zum einen ändert sich die sogenannte **Dominanzprüfung**,
  - zum anderen wurde die **Vollständigkeitsanalyse** korrigiert.

- **Allerdings ist das Ganze wieder in Form eines Preview Features umgesetzt und zum Nachvollziehen müssen Sie diese geeignet aktivieren, etwa wie folgt in der JShell:**

```
michaelinden@MBP-von-Michael ~ % jshell --enable-preview
|  Welcome to JShell -- Version 18-ea
|  For an introduction type: /help intro
```

JEP 420: Pattern Matching bei `switch`: Dominanzprüfung

- **Es können mehrere Pattern auf eine Eingabe matchen. Dasjenige, das «am besten» passt, wird dominierendes Pattern genannt. Im Beispiel dominiert das kürzere Pattern `String s` das längere davor angegebene.**

```
public static void main(String[] args) {
    multiMatch("Python");
    multiMatch(null);
}

static void multiMatch(Object obj) {
    switch (obj) {
        case null -> System.out.println("null");
        case String s && s.length() > 5 -> System.out.println(s.toUpperCase());
        case String s                   -> System.out.println(s.toLowerCase());
        case Integer i                  -> System.out.println(i * i);
        default -> {}
    }
}
```

JEP 420: Pattern Matching bei `switch`: Dominanzprüfung

- Es können mehrere Pattern auf eine Eingabe matchen. Dasjenige, das «am besten» passt, wird dominierendes Pattern genannt. Im Beispiel dominiert das kürzere Pattern `String s` das längere davor angegebene.

```
static void dominaceExample(Object obj) {
    switch (obj) {
      case null -> System.out.println("null");
      case String s                      -> System.out.println(s.toLowerCase());
      case String s && s.length() > 5 -> System.out.println(s.toUpperCase());
      case Integer i
      default -> {}
    }
}
```

This case label is dominated by one of the preceding case label

Press 'F2' for focus

- Mit Java 17 gab das noch keinen Fehler!

JEP 420: Pattern Matching bei `switch`: Dominanzprüfung

- **Probleme mit Konstanten (wurde mit Java 17 nicht entdeckt)**

```
static void dominaceExampleWithConstant(Object obj) {
    switch (obj.toString()) {
      case String s && s.length() > 5 -> System.out.println(s.toUpperCase());
      case "Sophie"                   -> System.out.println("My lovely daughter");
      defaul
    }
}
```

This case label is dominated by one of the preceding case label

Press 'F2' for focus

- **Korrektur:**

```
static void dominaceExampleWithConstant(Object obj) {
    switch (obj.toString()) {
      case "Sophie"                   -> System.out.println("My lovely daughter");
      case String s && s.length() > 5 -> System.out.println(s.toUpperCase());
      default -> System.out.println("FALLBACK");
    }
}
```

JEP 420: Pattern Matching bei `switch`: Vollständigkeitsanalyse

- Bug Fix im Bereich der Vollständigkeitsanalyse = der Prüfung, ob alle möglichen Pfade von den `cases` im `switch` abgedeckt werden

- mitunter fälschlicherweise Fehlermeldung «switch statement does not cover all possible input values»

```
static void performAction(BaseOp op) {
    switch (op) {
        case Add a -> System.out.println(a);
        case Sub s -> System.out.println(s);
        // default -> System.out.println("FALLBACK")
    }
}
```

```
static void performAction(BaseOp op) {
    switch (op) {
        case
        case
        // de
    }
}
```

'switch' statement does not cover all possible input values

Insert 'default' branch ⌥⇧⏎ More actions... ⌥⏎

```
@NotNull ↗
SwitchPreviewExample.BaseOp op
```

main

JEP 420: Pattern Matching bei `switch`: Vollständigkeitsanalyse

- **Bug Fix im Bereich der Vollständigkeitsanalyse = der Prüfung, ob alle möglichen Pfade von den `cases` im `switch` abgedeckt werden**

```
static sealed abstract class BaseOp permits Add, Sub {
}

static final class Add extends BaseOp {
}

static final class Sub extends BaseOp {
}

static void performAction(BaseOp op) {
    switch (op) {
        case Add a -> System.out.println(a);
        case Sub s -> System.out.println(s);
    }
}
```

DEMO & Hands on

JEP 421: Deprecate Finalization for Removal

JEP 421: Deprecate Finalization for Removal

- **Ursprünglich sollte die Finalization als Sicherheitsnetz dienen und Aufräumarbeiten oder Ressourcenfreigaben beim Ableben eines Objekts ermöglichen.**
- **Dazu wird während der Garbage Collection der jeweilige Finalizer in Form einer Methode `finalize()` aufgerufen, bevor der Speicher des Objekts zurückgegeben wird.**

- **Allerdings gibt es dabei diverse Hürden und Schwierigkeiten:**
  - nicht definiert, wann der Aufruf genau geschieht (möglicherweise einige Zeit nachdem das Objekt schon längst nicht mehr referenziert wird).
  - das Objekt können wieder in einen aktiven Zustand versetzt wird, quasi wiederbelebt werden.

- **Im Rahmen dieses JEPs wird die Finalization als deprecated sowie forRemoval. Zwar bleibt die Finalization standardmäßig noch aktiv, lässt sich nun aber auch deaktiviere.**

- **Die Hintergründe diesen JEPs beleuchtet Nicolai Parlog ausführlich online unter https://www.youtube.com/watch?v=eDgBnjOid-g**

Other Deprecations

Other Deprecations

- Durch den JEP 421 sind bereits alle `finalize()`-Methoden deprecated.
- In der Klasse `Runtime` wurden diverse überladene `exec()`-Methoden deprecated. Ebenso die Methode `runFinalization()`.
- In der Klasse `Thread` ist nun die Methode `stop()` nicht nur deprecated, sondern for removal.

Fazit

Positives

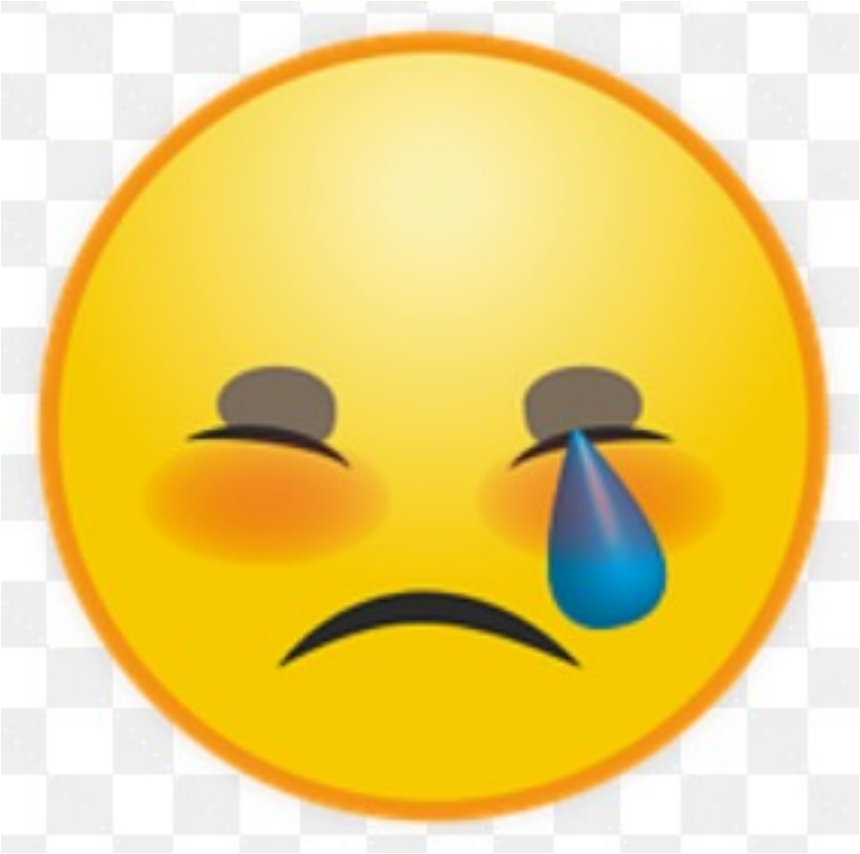
- eine paar Dinge aus Project COIN
- Switch / Records
- diverse praktische Erweiterungen in den APIs
- JPackage
- HTTP 2 (Java 11)
- Modularisierung mit Project JIGSAW (Java 9)--
  aber leider (immer noch) kein wirklich gutes Tooling

Negatives

- Mehrmalige Verschiebung von Java 9
  Sept. 2016 => März 2017 => Juli 2017 => Sept. 2017

- Folge-Release waren zwar pünktlich, aber manchmal (außer Java 14) ziemlich dünn bezüglich wichtigen Neuerungen

- Immer Mal wieder weniger als geplant
  - keine Versionierung bei JIGSAW
  - kein JSON-Support
  - statt Collection-Literals nur Convenience-Methods
  - Statt ZIP nur TEE (ing)-Kollektor



Hilfe


HELP
IS ON THE
WAY


Michael Inden
Java
Die Neuerungen in
Java 17 LTS und 18
dpunkt.verlag


Michael Inden
Java
Die Neuerungen in
Version 9 bis 14
Modularisierung, Syntax- und
API-Erweiterungen
dpunkt.verlag


5.
Auflage
Michael Inden
Der Weg zum
Java-Profi
Konzepte und Techniken für die
professionelle Java-Entwicklung
dpunkt.verlag

Questions?

Thank You