Workshop: Best of Java 17 bis 23

Java 21 Übungen

Ablauf

Dieser Workshop gliedert sich in mehrere Vortragsteile, die den Teilnehmern die Thematik Java 11 bis 23 sowie die dortigen Neuerungen überblicksartig näherbringen. Im Anschluss daran sind jeweils einige Übungsaufgaben von den Teilnehmern – idealerweise in Gruppenarbeit – am Rechner zu lösen.

Voraussetzungen

1) Aktuelles JDK 21 LTS (21.0.4 oder neuer) // sowie aktuelles JDK 23 installiert
2) Aktuelles Eclipse 2024-09 oder IntelliJ IDEA 2024.2.2 oder neuer installiert

Teilnehmer

- Entwickler mit Java-Erfahrung sowie
- SW-Architekten, die Java 11 bis 23 kennenlernen/evaluieren möchten

Kursleitung und Kontakt

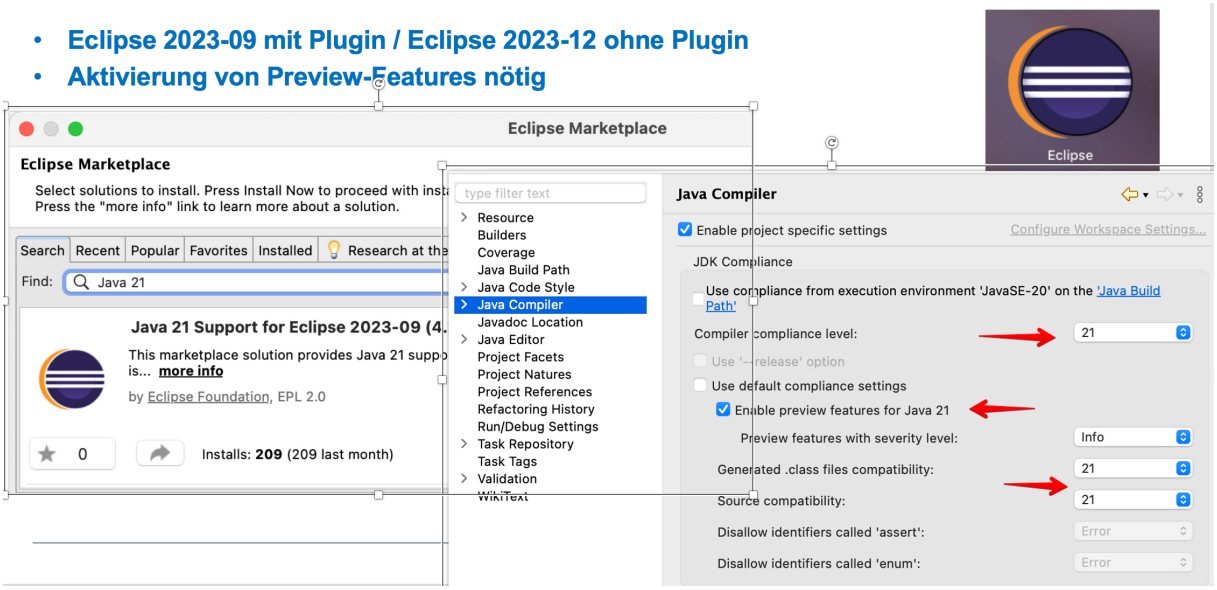
Michael Inden
Head of Development, freiberuflicher Buchautor, Trainer und Konferenz-Speaker
E-Mail: michael_inden@hotmail.com

Weitere Kurse (Java, Unit Testing, Design Patterns, JPA, Spring) biete ich gerne auf Anfrage als Online- oder Inhouse-Schulung an.

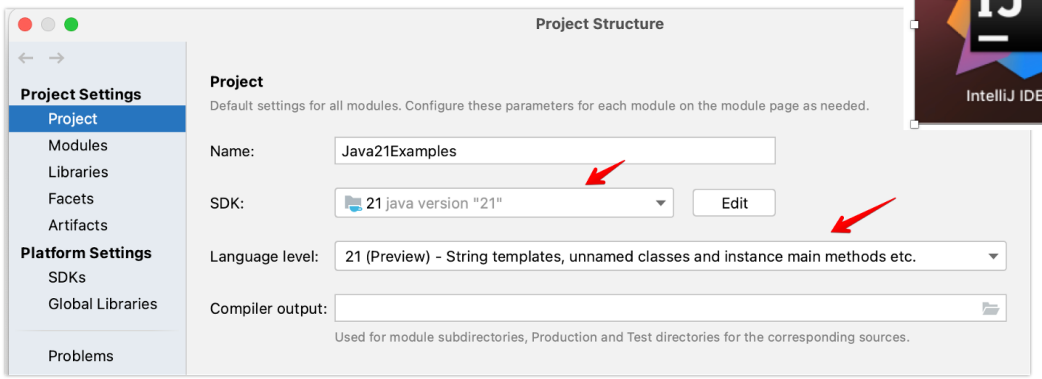
Konfiguration Eclipse / IntelliJ für Java 21 LTS

Bedenken Sie bitte, dass wir vor den Übungen noch einige Kleinigkeiten bezüglich Java/JDK und Compiler-Level konfigurieren müssen.

- **Eclipse 2023-09 mit Plugin / Eclipse 2023-12 ohne Plugin**
- **Aktivierung von Preview-Features nötig**



- **Aktivierung von Preview-Features nötig**



PART 1: Syntax-Erweiterungen bis Java 17 LTS

Lernziel: In diesem Abschnitt beschäftigen wir uns mit Syntax-Erweiterungen in Java 11 bis 17.

Aufgabe 1 – Syntaxänderungen bei switch

Vereinfache folgenden Sourcecode mit einem herkömmlichen switch-case durch die neue Syntax.

```
private static void dumbEvenOddChecker(int value)
{
      String result;

      switch (value)
      {
      case 1:
      case 3:
      case 5:
      case 7:
      case 9:
            result = "odd";
            break;

      case 0, 2, 4, 6, 8, 10:
            result = "even";
            break;

      default:
            result = "only implemented for values <= 10";
      }

      System.out.println("result: " + result);
}
```

Aufgabe 1a

Nutze zunächst nur die Arrow-Syntax, um die Methode kürzer und übersichtlicher zu schreiben.

Aufgabe 1b

Verwende nun noch die Möglichkeit, Rückgaben direkt zu spezifizieren und ändere die Signatur in `String dumbEvenOddChecker(int value)`

Aufgabe 1c

Wandle das Ganze so ab, dass du die Spezialform «yield mit Rückgabewert» verwendest.

Aufgabe 2 – Text Blocks

Vereinfache folgenden Sourcecode mit einem herkömmlichen String, der über mehrere Zeilen geht und nutze die neu eingeführte Syntax.

```
String multiLineStringOld = "THIS IS\n" +
            "A MULTI\n" +
            "LINE STRING\n" +
            "WITH A BACKSLASH \\\n";

String multiLineHtmlOld = "<html>\n" +
           "    <body>\n" +
           "        <p>Hello, world</p>\n" +
           "    </body>\n" +
           "</html>";

String javaScriptObjOld = ""
              + "{\n"
              + "    \"version\": \"Java13\",\n"
              + "    \"feature\": \"text blocks\",\n"
              + "    \"attention\": \"preview!\"\n"
              + "}\n";
```

Aufgabe 3 – Text Blocks mit Platzhaltern

Vereinfache folgenden Sourcecode mit einem herkömmlichen String, der über mehrere Zeilen geht und nutze die neu eingeführte Syntax:

```
String multiLineStringWithPlaceHoldersOld =
       String.format("HELLO \"%s\"!\n" +
                     "  HAVE %s\n" +
                     "  NICE \"%s\"!",
                     new Object[]{"WORLD", "A", "DAY"});

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

Produziere folgende Ausgaben mit der neuen Syntax:

```
HELLO "WORLD"!
  HAVE A
  NICE "DAY"!
```

Aufgabe 4 – Record-Grundlagen

Gegeben seien zwei einfache Klassen, die reine Datencontainer darstellen und somit lediglich ein öffentliches Attribut bereitstellen. Wandle diese in Records um:

```
class Square
{
     public final double sideLength;

     public Square(final double sideLength)
     {
          this.sideLength = sideLength;
     }
}

class Circle
{
     public final double radius;

     public Circle(final double radius)
     {
          this.radius = radius;
     }
}
```

Welche Vorteile ergeben sich – außer der kürzeren Schreibweise – durch den Einsatz von Records statt eigener Klassen?

Aufgabe 5 – Record

Erstelle auf Basis des nachfolgend gezeigten Records zwei Methoden, die eine JSON- und eine XML-Ausgabe erzeugen. Ergänze eine Gültigkeitsprüfung, sodass Name und Vorname mindestens 3 Zeichen lang sind und der Geburtstag nicht in der Zukunft liegt.

```
record Person(String firstName, String lastName,
              LocalDate birthday) {}
```

```
<Person>
     <firstName>Michael</firstName>
     <lastName>Inden</lastName>
     <birthday>1971-02-07</birthday>
</Person>
```

```
{
     "firstName" : "Michael",
     "lastName" : "Inden",
     "birthday" : "1971-02-07"
}
```

Aufgabe 6 – instanceof-Grundlagen

Gegeben seien folgende Zeilen mit einem `instanceof` sowie einem Cast. Vereinfache das Ganze mit den Neuerungen aus modernem Java.

```
Object obj = "BITTE ein BIT";

if (obj instanceof String)
{
     final String str = (String)obj;
     if (str.contains("BITTE"))
     {
          System.out.println("It contains the magic word!");
    }
}
```

Aufgabe 7 – `instanceof` und `record`

Vereinfache den Sourcecode mithilfe der Syntaxneuerungen bei `instanceof` und danach mithilfe der Besonderheiten bei Records.

```
record Square(double sideLength) {
}

record Circle(double radius) {
}

public double computeAreaOld(final Object figure)
{
     if (figure instanceof Square)
     {
          final Square square = (Square) figure;
          return square.sideLength * square.sideLength;
     }
     else if (figure instanceof Circle)
     {
          final Circle circle = (Circle) figure;
          return circle.radius * circle.radius * Math.PI;
     }
     throw new IllegalArgumentException("figure is not a
recognized figure");
}
```

Zwar haben wir durch `instanceof` sicher eine Verbesserung bezüglich Lesbarkeit und Anzahl Zeilen erzielt, jedoch deuten mehrere derartige Prüfungen auf einen Verstoß gegen das Open-Closed-Prinzip, eines der SOLID-Prinzipien guten Entwurfs, hin. Was wäre ein objektorientiertes Design? Die Antwort ist in diesem Fall einfach: Oftmals lassen sich `instanceof`-Prüfungen vermeiden, wenn man einen Basistyp einführt. **Vereinfache das Ganze durch ein Interface `BaseFigure` und nutze dieses passend.**

Bonus
Führe mit Rechtecken oder Polygonen einen oder zwei weitere(n) Typ(en) von Figuren ein. Das sollte aber keine Modifikationen in der Methode `computeArea()` erfordern.

PART 2: API- und JVM-Neuerungen bis 17 LTS

Lernziel: In diesem Abschnitt beschäftigen wir uns mit JVM-Erweiterungen und API-Neuerungen bis Java 17 LTS.

Aufgabe 1 – HTTP/2

Gegeben sei folgende HTTP-Kommunikation, die auf die Webseite von Oracle zugreift und diese textuell aufbereitet.

```
private static void readOraclePageJdk8() throws MalformedURLException, IOException
{
    final URL oracleUrl = new URL("https://www.oracle.com");
    final URLConnection connection = oracleUrl.openConnection();

    System.out.println(readContent(connection.getInputStream());
}

public static String readContent(final InputStream is) throws IOException
{
    try (final InputStreamReader isr = new InputStreamReader(is);
         final BufferedReader br = new BufferedReader(isr))
    {
        final StringBuilder content = new StringBuilder();
        String line;
        while ((line = br.readLine()) != null)
        {
            content.append(line + "\n");
        }
        return content.toString();
    }
}
```

Aufgabe 1a

Wandle den Sourcecode so um, dass das neue HTTP/2-API zum Einsatz kommt. Nutze die Klassen `HttpRequest` und `HttpResponse` und erstelle eine Methode `printResponseInfo(HttpResponse)`, die analog zu der obigen Methode `readContent(InputStream)` den Body ausliest und ausgibt. Zusätzlich soll noch der HTTP-Statuscode ausgegeben werden. Starte mit folgendem Programmfragment:

```
private static void readOraclePageJdk9() throws URISyntaxException,
                                                IOException,
                                                InterruptedException
{
    final URI uri = new URI("https://www.oracle.com");
    final HttpClient httpClient = // TODO
    final HttpRequest request = // TODO

    printResponseInfo(response);
}
```

Aufgabe 1b

Starte die Abfragen durch Aufruf von `sendAsync()` asynchron und verarbeite das erhaltene `CompletableFuture<HttpResponse>`.

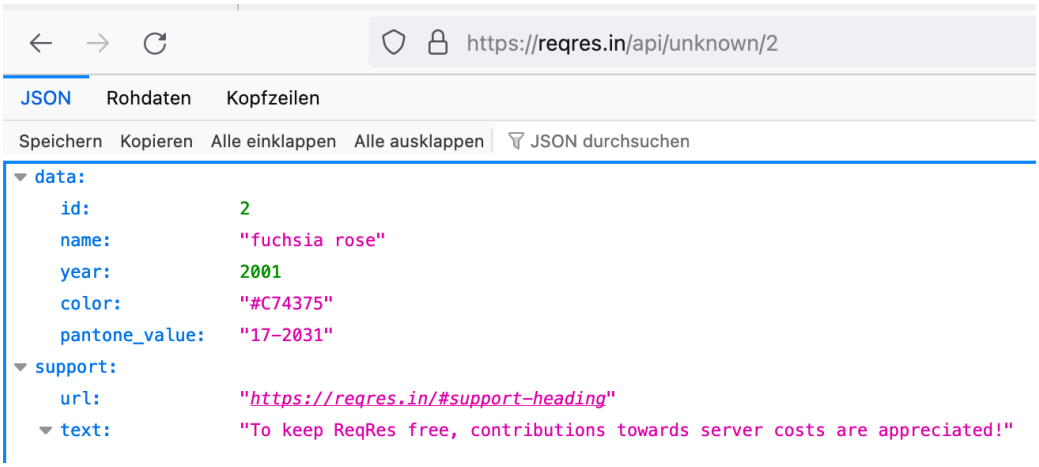
Aufgabe 2 – Direct Compilation

Aufgabe 2a

Schreibe eine Klasse `HelloWorld` zur Konsolenausgabe eines Grusses und speichere diese in einer gleichnamigen Java-Datei. Führe diese direkt mit dem Kommando `java` aus.

Aufgabe 2b

Schreibe eine Klasse `PerformGetWithHttpClient` und, um einen REST-Call etwa GET `https://reqres.in/api/unknown/2` auszuführen. Speichere das Programm in einer gleichnamigen Java-Datei. Führe diese direkt mit dem Kommando `java` aus. Dabei sollten in etwa die Daten wie bei dem Aufruf aus dem Browse geliefert werden.



Als Alternative kann man auch von der Seite `xkcd`, etwa die Grafik mit dieser Adresse `https://imgs.xkcd.com/comics/modern_tools.png`

Schreibe eine Klasse, die diese Grafik herunterlädt und diese als PNG-File speichert.

Bonus

Erstelle ein bash-Skript `exec_rest.sh` zum direkten Ausführen, denke an die korrekten Rechte (`chmod u+x`). Nutze für Windows den Umweg über eine `.bat`-Datei.

Aufgabe 3 – JPackage

Experimentiere mit dem PackagingDemo-Projekt und wandle dieses so ab, dass noch eine weitere Abhängigkeit verwendet wird, etwa auf Apache Commons.

```
jpackage --input target/ --name JPackageDemoApp
         --main-jar PackagingDemoExamples-1.0.0-SNAPSHOT.jar
         --main-class de.java17.ApplicationExample
         --type <dmg / msi / …>
```

Bei Bedarf kann man Folgendes ergänzen:

```
--java-options '--enable-preview'
```

PART 3: Neuerungen in Java 18 bis 21 LTS

Lernziel: In diesem Abschnitt beschäftigen wir uns mit Erweiterungen in Java 18 bis 21 LTS.

Aufgabe 1 – Wandle in Record Pattern um

Gegeben ist eine Definition einer Reise durch folgende Records:

```
record Person(String firstname, String lastname, LocalDate birthday) {
}

record TravelInfo(LocalDate start, Duration maxTravellingTime) {
}

record City(Integer zipCode, String name) {
}

record Journey(Person person,
               TravelInfo travelInfo,
               City from,
               City to) {
}
```

Zur Gültigkeitsprüfung werden verschiedene Konsistenz-Checks und Prüfungen ausgeführt. Dabei werden verschachtelte Bestandteile wie Person oder City eines Journey-Objekts auf `!= null` geprüft und somit deren Existenz für eine nachfolgende Abfrage abgesichert. Dazu sieht man mitunter – vor allem in Legacy-Code – Implementierungen, die tief verschachtelte `if`s und diverse `null`-Prüfungen enthalten, etwa wie folgt:

```
static boolean checkFirstNameTravelTimeAndDestZipCode(final Object obj) {
    if (obj instanceof Journey journey) {
        if (journey.person() != null) {
            var person = journey.person();

            if (journey.travelInfo() != null) {
                var travelInfo = journey.travelInfo();

                if (journey.to() != null) {
                    var to = journey.to();
```

Die Aufgabe besteht nun darin, das Ganze mithilfe von Record Patterns verständlicher und kompakter zu realisieren.

Bonus: Vereinfache die Angaben in den Record Patterns mit `var`.

Aufgabe 2 – Nutze Record Patterns für rekursive Aufrufe

Gegeben sind Definitionen einiger Figuren durch folgende Records:

```
sealed interface Figure {}

record Point(int x, int y) implements Figure {}

record Line(Point start, Point end) implements Figure {}


record Triangle(Point pointA,
                Point pointB,
                Point PointC) implements Figure {
}
```

Zudem ist die folgende Methode definiert, die die x- und y-Koordinate eines Punkts multipliziert. Das ist für `Point` bereits realisiert. Das `switch` soll so ergänzt werden, dass `case`s für `Line` und `Triangle` hinzugefügt werden. Als Berechnung sollen die jeweiligen Teilkomponenten in Form von Points addiert werden, indem die bisherige Methode `process()` aufgerufen wird:

```
static int process(Figure figure) {
    return switch (figure) {
        case Point(int x, int y) -> x * y;
        // TODO
        default -> throw new IllegalStateException("Unexpected value: " +
                                                   figure);
    };
}
```

Aufgabe 3 – Wandle in virtuelle Threads um

Gegeben ist eine Ausführung verschiedener Tasks mithilfe eines klassischen `ExecutorService` und einer vorgegebenen Pool-Size von 100:

```
try (var executor = Executors.newFixedThreadPool(100)) {
    IntStream.range(0, 10_000).forEach(i -> {
        executor.submit(() -> {
            Thread.sleep(Duration.ofSeconds(5));

            System.out.println("Task " + i + " finished!");
            return i;
        });
    });
}
```

Wandle das Ganze so um, dass virtuelle Threads genutzt werden, und prüfe dies nach. Nutze dazu eine passende Methode in `Thread`.

Aufgabe 4 – Experimentiere mit Sequenced Collections

Gegeben sei folgende Methode mit einigen TODO-Kommentaren, die die ersten Primzahlen als Liste aufbereiten soll. Zudem sollen vorne und hinten Elemente eingefügt sowie eine umgekehrte Reihenfolge aufbereitet werden.

```
private static void primeNumbers()
{
    List<Integer> primeNumbers = new ArrayList<>();
    primeNumbers.add(3); // [3]
    // TODO: add 2
    primeNumbers.addAll(List.of(5, 7, 11));
    // TODO: add 13

    System.out.println(primeNumbers); // [2, 3, 5, 7, 11, 13]
    // TODO print first and last element
    // TODO print reverser order

    // TODO: add 17 as last
    System.out.println(primeNumbers); // [2, 3, 5, 7, 11, 13, 17]
    // TODO print reverser order
}
```

Bonus

Experimentiere mit dem Interface `SequencedSet<E>` und erstelle mit den passenden Methoden eine sortierte Menge, bestehend aus den Buchstaben A, B und C:

```
private static void createABCSet()
{
    Set<String> numbers = new LinkedHashSet<>();

    // TODO

    // TODO print first and last element
    // TODO print reverser order
}
```

PART 3: Neuerungen in Java 18 bis 21 LTS Preview + Incubator

Lernziel: In diesem Abschnitt beschäftigen wir uns mit Erweiterungen in Java 18 bis 21 LTS, die noch nicht final sind.

Aufgabe 5 – Wandle mit Structured Concurrency um

Gegeben ist eine Ausführung verschiedener Tasks mithilfe eines klassischen `ExecutorService` und einer Zusammenführung der Berechnungsergebnisse:

```
static void executeTasks(boolean forceFailure) throws InterruptedException,
                                                      ExecutionException
{
    try (var executor = Executors.newFixedThreadPool(50)) {
        Future<String> task1 = executor.submit(() -> {
            return "1";
        });
        Future<String> task2 = executor.submit(() -> {
            if (forceFailure)
                throw new IllegalStateException("FORCED BUG");

            return "2";
        });
        Future<String> task3 = executor.submit(() -> {
            return "3";
        });

        System.out.println(task1.get());
        System.out.println(task2.get());
        System.out.println(task3.get());
    }
}
```

Mithilfe von Structured Concurrency soll der `ExecutorService` ersetzt werden und die Strategie `ShutdownOnFailure` die Verarbeitung im Fehlerfall klarer machen. Analysiere die Abarbeitungen im Fehlerfall.

Führen wir die Methode einmal mit beiden Wertebelegungen aus:

```
jshell> import java.util.concurrent.*

jshell> executeTasks(false)
 result: 1 / 2 / 3

jshell> executeTasks(true)
| Ausnahme java.util.concurrent.ExecutionException:
java.lang.IllegalStateException: FORCED BUG
   at FutureTask.report (FutureTask.java:122)
   at FutureTask.get (FutureTask.java:191)
   at executeTasks (#19:16)
   at (#21:1)
|  Verursacht von: java.lang.IllegalStateException: FORCED BUG
|        at lambda$executeTasks$1 (#19:8)
```

Aufgabe 6 – Experimentiere mit Template Processor

Schreibe einen eigenen Template Processor, der die Werte mit [[und]] oder alternativ jeweils vorne und hinten einem ' umschließt:

```
var name = "Michael";
int age = 53;
System.out.println(DOUBLE_BRACES.
                   "Hello, \{name}! Next year, you'll be \{age + 1}.");

=>

Hello, [[Michael]]! Next year, you'll be [[54]].
```

a) Verwende dazu `fragments()` und `values()` und eine Schleife.
b) Vereinfache das Ganze durch `interpolate()` und das Stream-API.
c) Nutze einen parametrierbaren Lambda, um die Start- und Endsequenz frei wählbar zu machen.
d) Erzeuge einen Template Processor der ähnlich zu den f-Strings in Python (`f"Berechnung: {x} + {y} = {x + y}"`) arbeitet, ohne direkt STR zu referenzieren.

Aufgabe 7 – Experimentiere mit Unnamed Patterns and Variables

Vereinfache die folgende Methode durch Einsatz von Unnamed Patterns and Variables, um die Lesbarkeit und Verständlichkeit zu steigern – nutze aus, dass die IDEs unbenutzte Variablen anzeigen – verzichte auf automatisierte Hilfen der IDE und forme schrittweise um.

```
static boolean checkFirstNameAndCountryCodeAgainImproved(Object obj)
{
    if (obj instanceof Journey(
          Person(var firstname, var lastname, var birthday),
          TravelInfo(var start, var maxTravellingTime), var from,
          City(var zipCode, var name)))
    {
        if (firstname != null && maxTravellingTime != null
                              && zipCode != null)
        {
            return firstname.length() > 2
                   && maxTravellingTime.toHours() < 7
                   && zipCode >= 8000 && zipCode < 8100;
        }
    }
    return false;
}
```