Workshop: Best of Modern Java 21 Übungen

Ablauf

Dieser Workshop stellt die Neuerungen aus Java 18 bis 21 überblicksartig vor. Zum Vertiefen des Erlernten sind ergänzend jeweils einige Übungsaufgaben von den Teilnehmern – idealerweise in Gruppenarbeit – am Rechner zu lösen.

Voraussetzungen

1) Aktuelles JDK 21 installiert
2) Aktuelles Eclipse oder IntelliJ IDEA installiert

Teilnehmer

- Entwickler mit Java-Erfahrung sowie
- SW-Architekten, die Java 21 kennenlernen/evaluieren möchten

Kursleitung und Kontakt

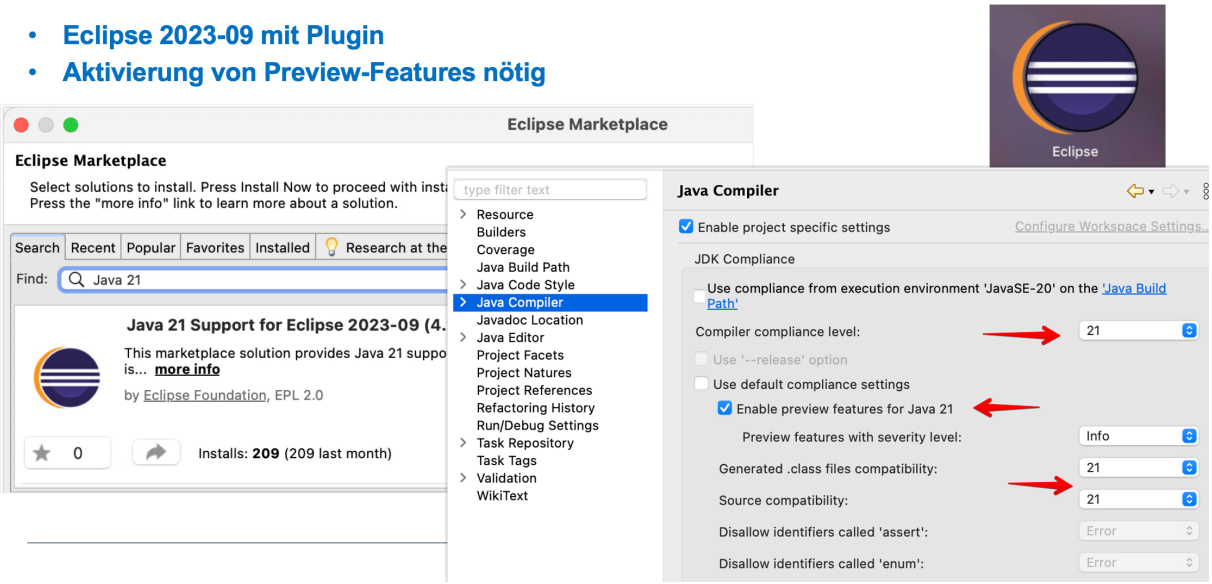
Michael Inden

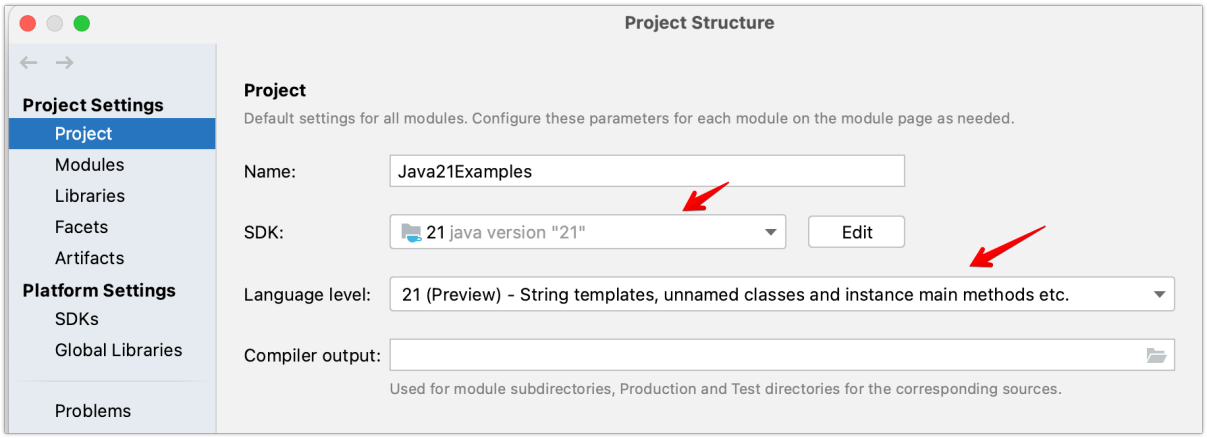
Head of Development, freiberuflicher Buchautor, Trainer und Konferenz-Speaker
E-Mail: michael_inden@hotmail.com

Konfiguration Eclipse / IntelliJ für Java 21

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

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





PART 1: Neuerungen in Java 18 bis 21

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

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(String zipCode, String name) {
}

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

Zudem werden verschiedene Konsistenz-Checks und Prüfungen ausgeführt, die auf verschachtelte Bestandteile zugreifen. Dazu sieht man mitunter – vor allem in Legacy-Code – Implementierungen, die tief verschachtelte `if`s und diverse `null`-Prüfungen enthält.

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

Bonus: Vereinfache die Angabe 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 ergänzt werden soll, sodass für die beiden Figuren `Line` und `Triangle` die Punkte addiert werden sollen:

```
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 50:

```
try (var executor = Executors.newFixedThreadPool(50)) {
    IntStream.range(0, 1_000).forEach(i -> {
        executor.submit(() -> {
            Thread.sleep(Duration.ofSeconds(1));

            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 Klasse 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.

```
public static void main(String[] args)
{
    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
}
```

PART 2: Neuerungen in Java 18 bis 21

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

Aufgabe 5 – Wandle mit Structured Concurrency um

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

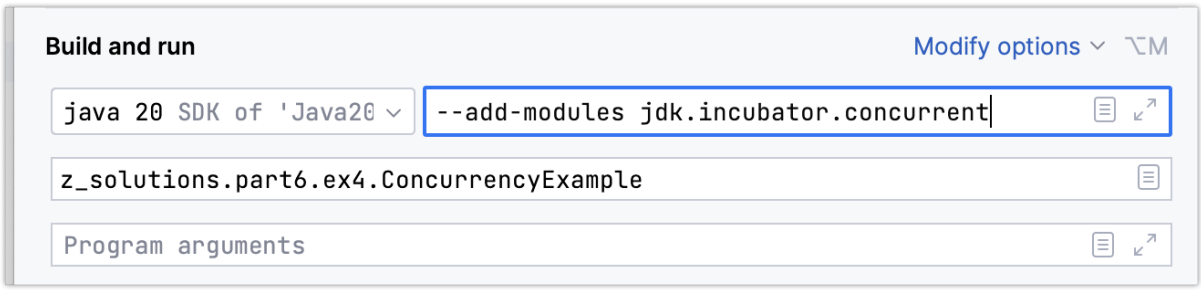
```
static void executeTasks(boolean forceFailure) throws InterruptedException,
                                                      ExecutionException
{
    try (var executor = Executors.newFixedThreadPool(50)) {
        Future<String> task1 = executor.submit(() -> {
            return "1";
      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

Tipp: Es handelt sich in Java 20 noch um ein Incubator-Feature, weshalb man beim Kompilieren und Start passende Konfigurationen vornehmen muss. In Java 21 ist es bereits ein Preview Feature.



Aufgabe 6 – Experimentiere mit Template Processor

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

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

=>

Hello, [[Michael]]! Next year, you'll be [[53]].
```

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 und Variables

Vereinfache die folgende Methode durch Einsatz von Unnamed Pattern uns Variables, um die Lesbarkeit und Verständlichkeit zu steigern – nutze aus, dass die IDEs unbenutzte Variablen anzeigen:

```
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;
}
```