

Workshop: Best of Java 11 bis 22

Übungen speziell für Java 22

Ablauf

Dieser Workshop gliedert sich in mehrere Vortragsteile, die den Teilnehmern die Thematik Java 11 bis 22 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 22 und JDK 21 LTS (21.0.2 oder neuer) installiert
- 2) Aktuelles Eclipse 2024-03 oder IntelliJ IDEA 2024.1 installiert

Teilnehmer

- Entwickler mit Java-Erfahrung sowie
- SW-Architekten, die Java 11 bis 22 kennenlernen/evaluieren möchten

Kursleitung und Kontakt

Michael Inden

Head of Development, freiberuflicher Buchautor, Trainer und Konferenz-Speaker

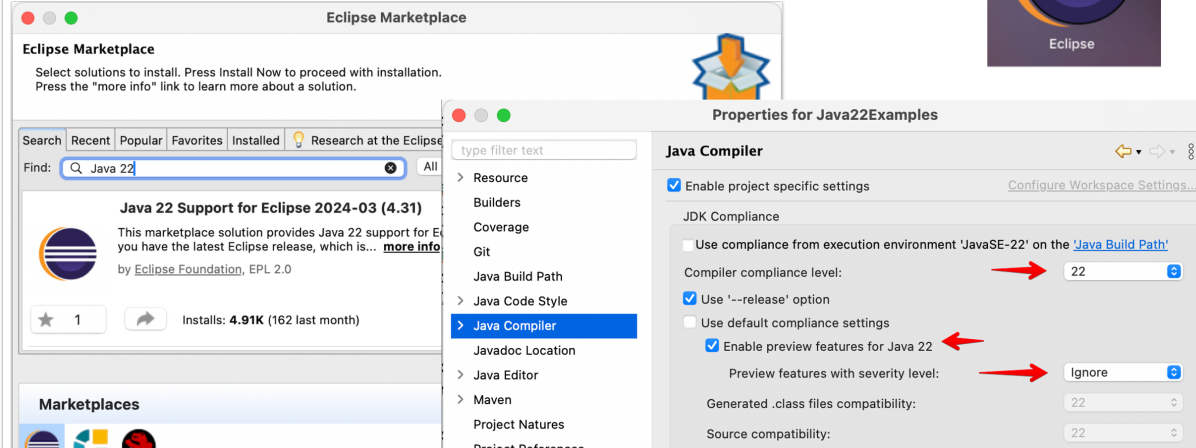
E-Mail: michael_inden@hotmail.com

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 22

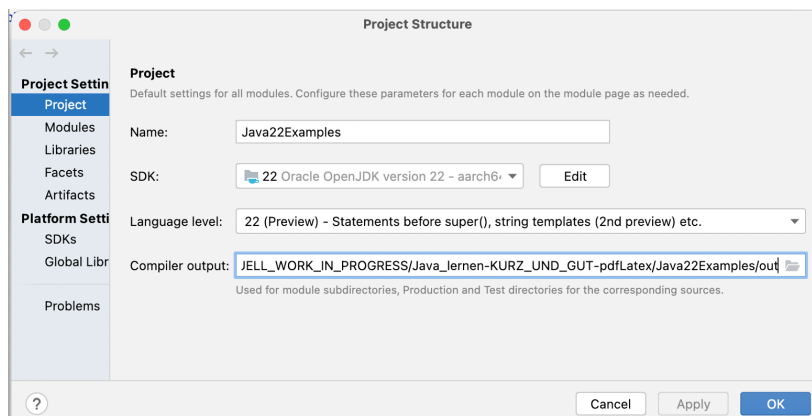
Bedenken Sie bitte, dass wir vor den Übungen noch einige Kleinigkeiten bezüglich Java/JDK und Compiler-Level konfigurieren müssen.

- **Eclipse 2024-03 mit Plugin**
- **Aktivierung von Preview-Features nötig**



The screenshot shows the Eclipse Marketplace interface on the left, where the 'Java 22 Support for Eclipse 2024-03 (4.31)' plugin is highlighted. On the right, the 'Properties for Java22Examples' dialog is open, showing the 'Java Compiler' tab. Red arrows point to the 'Compiler compliance level' set to '22' and the 'Enable preview features for Java 22' checkbox, which is checked. The 'Preview features with severity level' is set to 'Ignore'.

- **Aktivierung von Preview-Features nötig**



The screenshot shows the 'Project Structure' dialog in IntelliJ IDEA. The 'Project' tab is selected. The 'Name' field is 'Java22Examples'. The 'SDK' is set to '22 Oracle OpenJDK version 22 - aarch64'. The 'Language level' is set to '22 (Preview) - Statements before super(), string templates (2nd preview) etc.'. The 'Compiler output' field is set to 'JELL_WORK_IN_PROGRESS/Java_lernen-KURZ_UND_GUT-pdfLatex/Java22Examples/out'.



PART 4: Erweiterungen in Java 22

Lernziel: Kennenlernen von Syntax-Neuerungen und verschiedenen API- sowie JVM-Erweiterungen in Java 22 anhand von Beispielen.

Aufgabe 1 – Kennenlernen von Statements before super(...)

Entdecke die Eleganz durch die neue Syntax, Aktionen von dem Aufruf von `super()` ausführen zu können. Es soll eine Gültigkeitsprüfung von Parametern von der Konstruktion der Basisklasse erfolgen.

```
public Rectangle(Color color, int x, int y, int width, int height)
{
    super(color, x, y);

    if (width < 1 || height < 1) throw
        new IllegalArgumentException("width and height must be positive");

    this.width = width;
    this.height = height;
}
```

Aufgabe 2 – Kennenlernen von Statements before super(...)

Nutze Aktionen von dem Aufruf von `super()` ausführen zu können. Hier nimmt die Basisklasse einen anderen Typ entgegen als die Subklasse. Dabei kommt der Trick mit der Hilfsmethode ins Spiel. Neben einer Prüfung und Konvertierung erfolgen dort noch aufwändige Aktionen. Die Aufgabe ist nun, das Ganze lesbarer mit der neuen Syntax umzuwandeln – was sind die weiteren Vorteile dieser Variante?

```
public StringMsgOld(String payload)
{
    super(convertToByteArray(payload));
}

private static byte[] convertToByteArray(final String payload)
{
    if (payload == null)
        throw new IllegalArgumentException("payload should not be null");

    String transformedPayload = heavyStringTransformation(payload);
    return switch (transformedPayload) {
        case "AA" -> new byte[]{1, 2, 3, 4};
        case "BBBB" -> new byte[]{7, 2, 7, 1};
        default -> transformedPayload.getBytes();
    };
}

private static String heavyStringTransformation(String input) {
    return input.repeat(2);
}
```

Aufgabe 3 – Kennenlernen der Standard-Gatherer

Lerne das neue Interface `Gatherer` als Grundlage für Erweiterungen von `Intermediate Operations` mit seinen Möglichkeiten kennen.

Ergänze den folgenden Programmschnipsel, um das Produkt aller Zahlen im Stream zu bilden. Nutze dazu eine der neuen vordefinierten `Gatherer` aus der Klasse `Gatherers`.

```
var crossMult = Stream.of(1, 2, 3, 4, 5, 6, 7);
                        // TODO
// crossMult ==> Optional[5040]
```

Außerdem sollen folgende Eingabedaten in jeweils Gruppen von 3 Elementen aufgeteilt werden:

```
var values = Stream.of(1, 2, 3, 10, 20, 30, 100, 200, 300);
// TODO
// [[1, 2, 3], [10, 20, 30], [100, 200, 300]]
```

Aufgabe 4 – Besonderheiten der Structured Concurrency

Die neu eingeführte `Structured Concurrency` bietet nicht nur die bereits (bestens) bekannte Strategie `ShutdownOnFailure`, die beim Auftreten eines Fehlers alle anderen Berechnungen stoppt, sondern auch die für einige Anwendungsfälle praktische Strategie `ShutdownOnSuccess`. Damit lassen sich mehrere Berechnungen beginnen und nachdem eine ein Ergebnis geliefert hat, alle anderen Teilaufgaben stoppen. Wozu kann das nützlich sein? Stellen wir uns verschiedene Suchanfragen vor, bei der die Schnellste gewinnen soll.

Als Aufgabe sollen wir den Verbindungsaufbau zum Mobilnetz in den Varianten 5G, 4G, 3G und WiFi modellieren. Befülle nachfolgendes Programmstück mit Leben:

```
public static void main(final String[] args) throws ExecutionException,
                                                    InterruptedException
{
    try (var scope =
        new StructuredTaskScope.ShutdownOnSuccess<NetworkConnection>())
    {
        // TODO
        StructuredTaskScope.Subtask<NetworkConnection> result1 = null;
        StructuredTaskScope.Subtask<NetworkConnection> result2 = null;
        StructuredTaskScope.Subtask<NetworkConnection> result3 = null;
        StructuredTaskScope.Subtask<NetworkConnection> result4 = null;
        // TODO

        System.out.println(STR."Wifi \{result1.state()}/5G \{result2.state()}" +
                           STR."/4G \{result3.state()}/3G \{result4.state()}");
        System.out.println("found connection: " + scope.result());
    }
}
```

BONUS: Was passiert, wenn in einer der Methoden eine Exception ausgelöst wird?

Aufgabe 5 – Scoped Values als ThreadLocal-Alternative

In dieser Aufgabe soll ein herkömmlich per Parameterübergabe realisiertes Programm auf Scoped Values umgestellt werden. Dabei wird eine Request-Verarbeitung mit einer mehrschichtige Aufrufhierarchie mit simplifizierten Controller- und Service-Klassen nachempfunden. Den Ausgangspunkt bildet die Klasse `ScopedValuesExample`, die zwei Scoped Values definiert, um Informationen zum eingeloggten Benutzer sowie zum Zeitpunkt des Requests bereitstellen zu können. Derzeit sind diese noch ungenutzt und die Informationen werden jeweils als Parameter weitergegeben:

```
public static void main(String[] args) throws Exception
{
    // Simuliere Requests
    for (String name : List.of("ATTACKER", "ADMIN"))
    {
        var user = new User(name, name.toLowerCase());
        controller.consumingMethod(user, ZonedDateTime.now());

        controller.consumingMethod(user, null); // no time passed

        String answer = controller.process(user, ZonedDateTime.now());
        System.out.println(answer);

        String answer2 = controller.process(user, null); // no time passed
        System.out.println(answer2);
    }
}
```

Ihre Aufgabe ist es, dass auf Scoped Values umzustellen und diese passend mit Werten zu befüllen und deren Propagation in der Aufrufkette von `ScopedValuesExample` über Controller und Service. In den beiden Klassen soll dann auf die Informationen zugegriffen werden, ohne dass diese bei den Methodenaufrufen als Parameter übergeben werden müssen.

Aufgabe 6 – Kennenlernen des und Experimentieren mit dem Vector API

Das Vector API erlaubt es, SIMD-Berechnungen durchzuführen. In dieser Aufgabe soll eine als Skalarberechnung vorgegebene Implementierung auf des Vector API umgestellt werden.

Gegeben ist folgende Skalarberechnung der Formel $a_i * b_i + a_i * b_i - (a_i + b_i)$ mit dieser Implementierung:

```
static float[] scalarComputation(float[] a, float[] b)
{
    float[] c = new float[a.length];

    for (int i = 0; i < a.length; i++)
    {
        c[i] = a[i] * b[i] + a[i] * b[i] - (a[i] + b[i]);
    }

    return c;
}
```

Wandle das in passende Aufrufe mit dem Vector API um. Experimentiere ein wenig:

- 1) Was passiert, wenn man die Schleife nicht korrekt abbildet?
- 2) Wie kann man die Formel vereinfachen? Lässt sich dort vor einer Skalarmultiplikation, also mit einem fixen Wert, profitieren?