

Best of Modern Java 21 & 22

Meine Lieblingsfeatures

Übungen speziell für Java 22

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 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

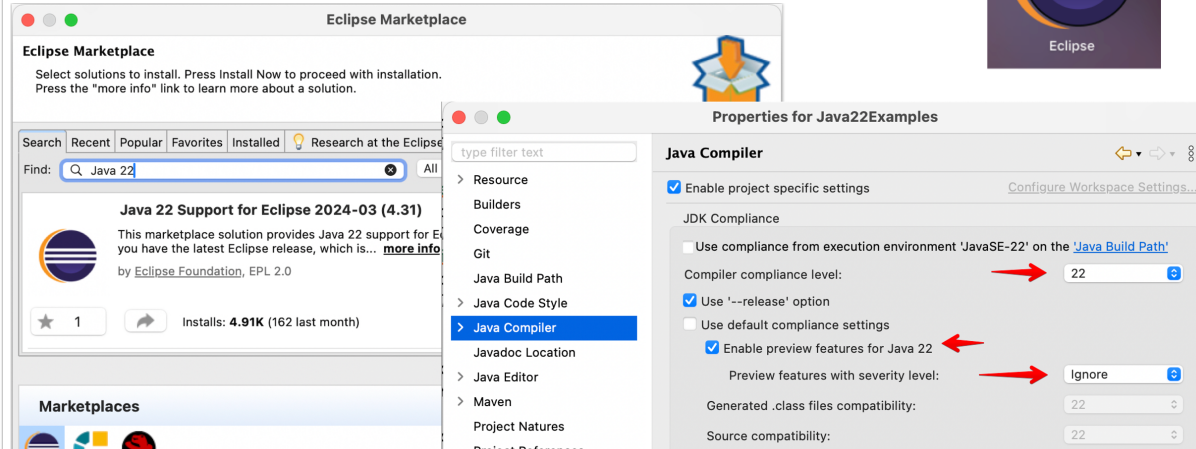
E-Mail: [michael inden@hotmail.com](mailto:michael.inden@hotmail.com)

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

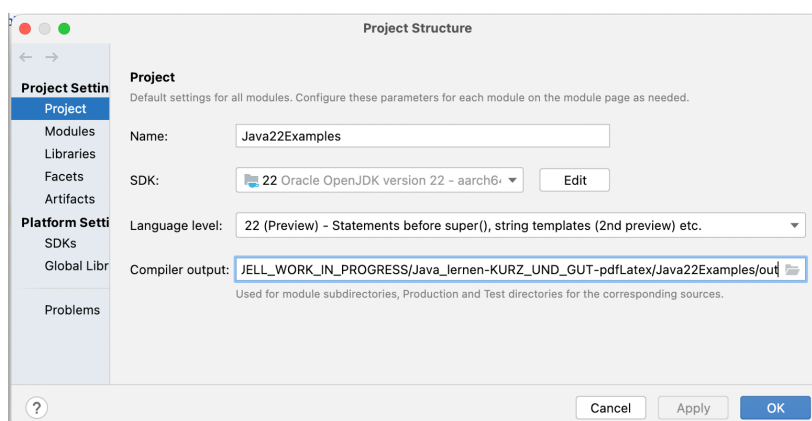
Bedenken Sie bitte, dass wir vor den Übungen noch einige Kleinigkeiten bezüglich Java/JDK und Compiler-Level konfigurieren müssen.

- **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 selected. 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**



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 vor dem Aufruf von `super()` ausführen zu können. Es soll eine Gültigkeitsprüfung von Parametern vor 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-Gatherers

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 einen 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?