Best of Modern Java 21 – 25 LTS
My Favorites

Procedure

Some exercises are to be solved by the participants.

Requirements

1) Current JDK 25 LTS and JDK 21 LTS are installed
2) Current Eclipse 2025-09 with Java 25 Plugin or IntelliJ IDE2025.2.1 is installed

Attendees

- Developers with Java experience, as well as
- SW-Architects, who would like to learn/evaluate Java 21 to 25 LTS.

Course management and contact

Michael Inden
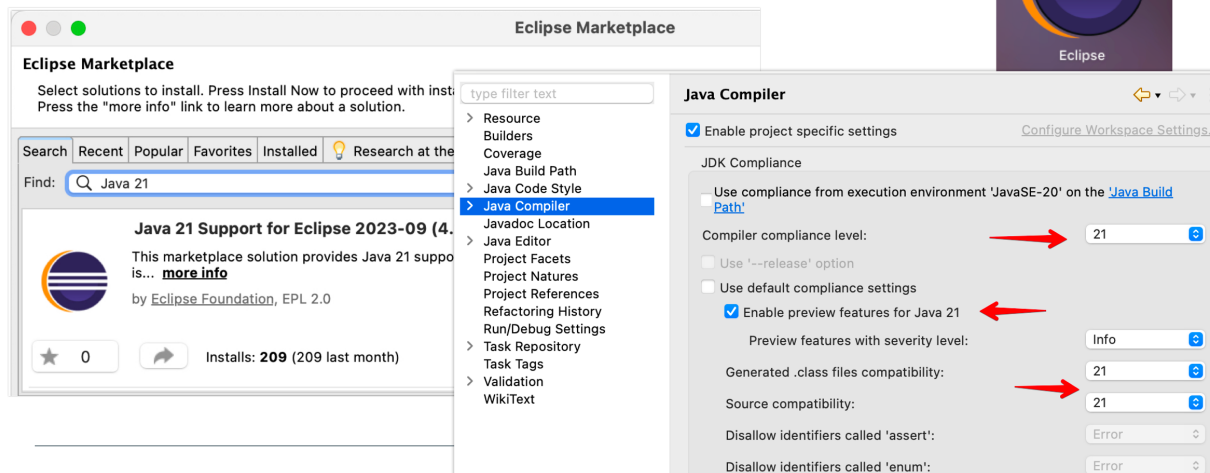
Independent SW Consultant, Author, and Trainer
E-Mail: michael_inden@hotmail.com

Please do not hesitate to ask: Further courses (Java, Unit Testing, Design Patterns) are available on request as in-house training.
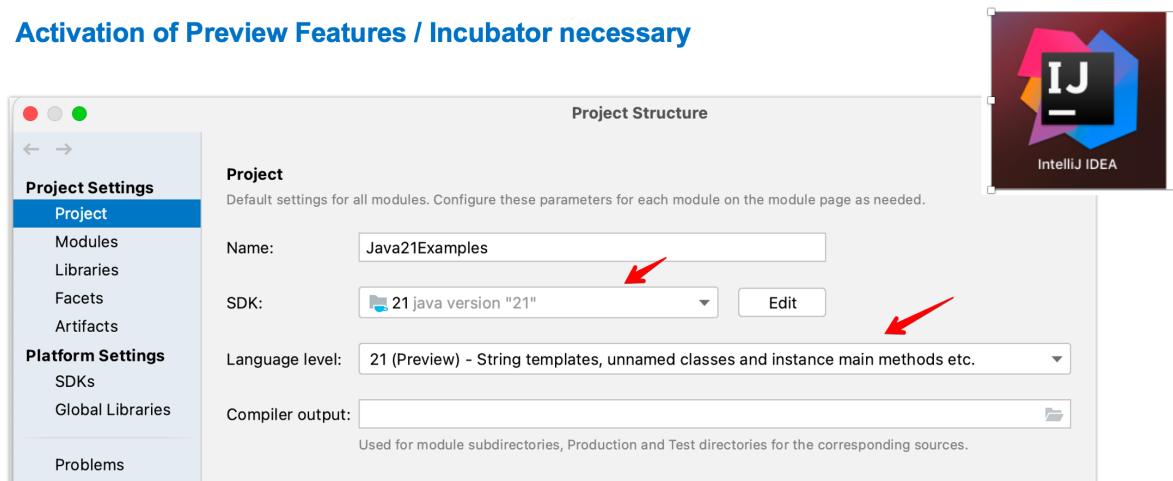
Configuration Eclipse / IntelliJ for Java 25 LTS

Please note that we have to configure some small things before the exercises if you are not using the latest IDEs.

- **Eclipse 2023-09 with Plugin**
- **Activation of Preview Features / Incubator necessary**

- **Activation of Preview Features / Incubator necessary**

Extensions in Java 25 LTS

Objective: Get to know syntax innovations and API extensions in Java 25 LTS using examples.

Exercise 1 – Flexible Constructor Bodies

Discover the elegance of the new syntax for executing actions before `super()` is called. You should perform a validity check of parameters before constructing the base class.

```java
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;
}
```

Exercise 2 – Flexible Constructor Bodies

Use the ability to execute actions before `super()` is called. Here, the base class accepts a different type than the subclass. This is where the trick with the helper method comes into play. In addition to a check and conversion, further complex actions are carried out there. The task now is to convert the whole thing into a more readable form using the new syntax – what are the other advantages of this variant?

```java
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);
}
```

Exercise 3 – Predefined Gatherers

Get to know the new interface `Gatherer` as a basis for extensions of intermediate operations and their possibilities.

Complete the following program snippet to calculate the product of all numbers in the stream. Use one of the new predefined gatherers from the `Gatherers` class.

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

In addition, the following input data is to be divided into groups of 3 elements:

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

Exercise 4 – Structured Concurrency

Structured Concurrency not only offers the joiner `awaitAllSuccessfulOrThrow()`, which stops all other calculations when an error occurs, but also the strategy `anySuccessfulResultOrThrow()`, which is useful for some use cases. This allows multiple calculations to be started and, once one has delivered a result, all other subtasks to be stopped. Why might this be useful? Let's imagine various search queries where the fastest one wins.

As an example, we should model the process of establishing a connection to the mobile network in the variants 5G, 4G, 3G, and WiFi. Bring the following program to life:

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

        NetworkConnection result = null; // TODO

        System.out.println("Wifi " + result1.state() +
                        "/5G " + result2.state() +
                        "/4G " + result3.state() +
                        "/3G " + result4.state());

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