Best of Modern Java 21 –23
My Favorites

Procedure

This workshop is divided into several lecture parts, which introduce the subject Java 9 to 17, and gives an overview of the innovations. Following this, some exercises are to be solved by the participants - ideally in pair / group programming - on the computer.

Requirements

1) Current JDK 23 and JDK 21 LTS are installed
2) Current Eclipse 2024-09 with Java 23 Plugin or IntelliJ IDE2024.2.2 is installed

Attendees

- Developers with Java experience, as well as
- SW-Architects, who would like to learn/evaluate Java 11 to 23.

Course management and contact

Michael Inden
Head of Development, independent SW Consultant, Author, and Trainer
E-Mail: michael_inden@hotmail.com

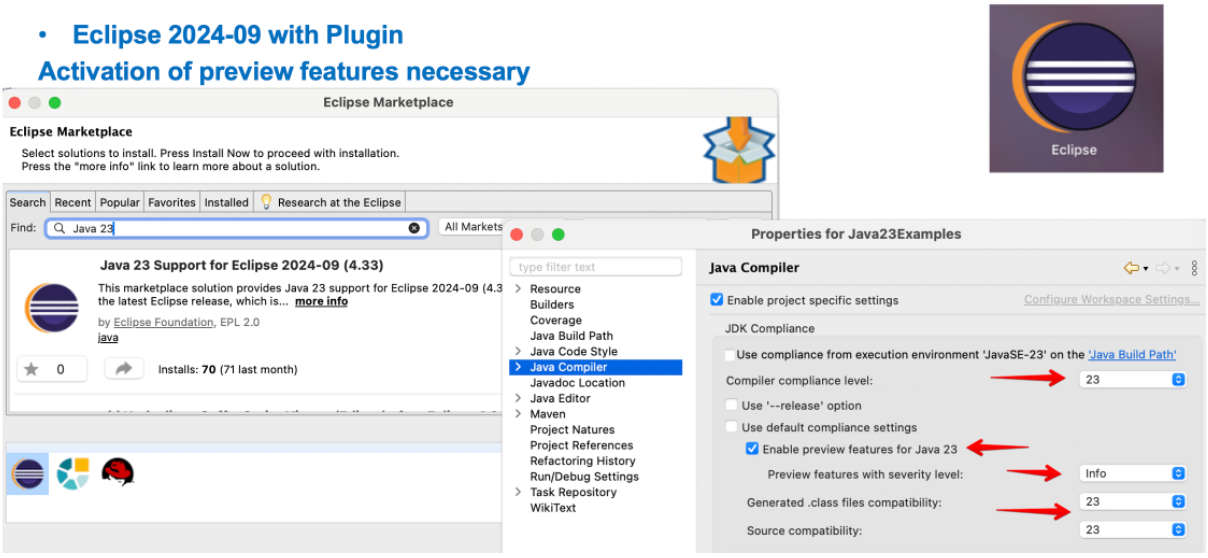
Please do not hesitate to ask: Further courses (Java, Unit Testing, Design Patterns) are available on request as in-house training.

Configuration Eclipse / IntelliJ

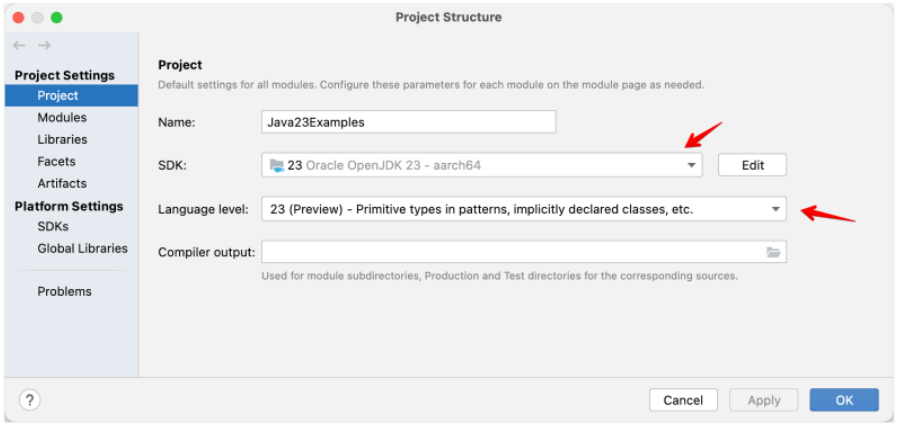
Please note that we have to configure some small things before the exercises if you are not using the latest IDEs.

- **Eclipse 2024-09 with Plugin**

Activation of preview features necessary



- **Activation of preview features / Incubator necessary**



Extensions in Java 22 & 23

Objective: Get to know syntax innovations and API extensions in Java 22 & 23 using examples.

Exercise 1 – Statements before super(…)

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.

```
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 – Statements before super(…)

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?

```
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.

```
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:

```
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 offers not only the well-known `ShutdownOnFailure` strategy, which stops all other calculations when an error occurs but also the `ShutdownOnSuccess` strategy, which is helpful for some applications. This strategy allows multiple calculations to be started and all other subtasks to be stopped after one has returned a result. What can this be useful for? Let's imagine various search queries in which the fastest should win.

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:

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

BONUS: What happens if an exception is thrown in one of the methods?