Workshop: Best of Java 17 – 23
Java 21 Exercises

Procedure

This workshop is divided into several lecture parts, which introduce the subject Java 17 to 23 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 21 LTS ((21.0.2 or newer) is installed
2) Current Eclipse 2023-09 with Java 21 Plugin or 2024-03 or IntelliJ IDE 2023.2.2 or newer is installed

Attendees

- Developers with Java experience, as well as
- SW-Architects, who would like to learn/evaluate Java 17 to 23.

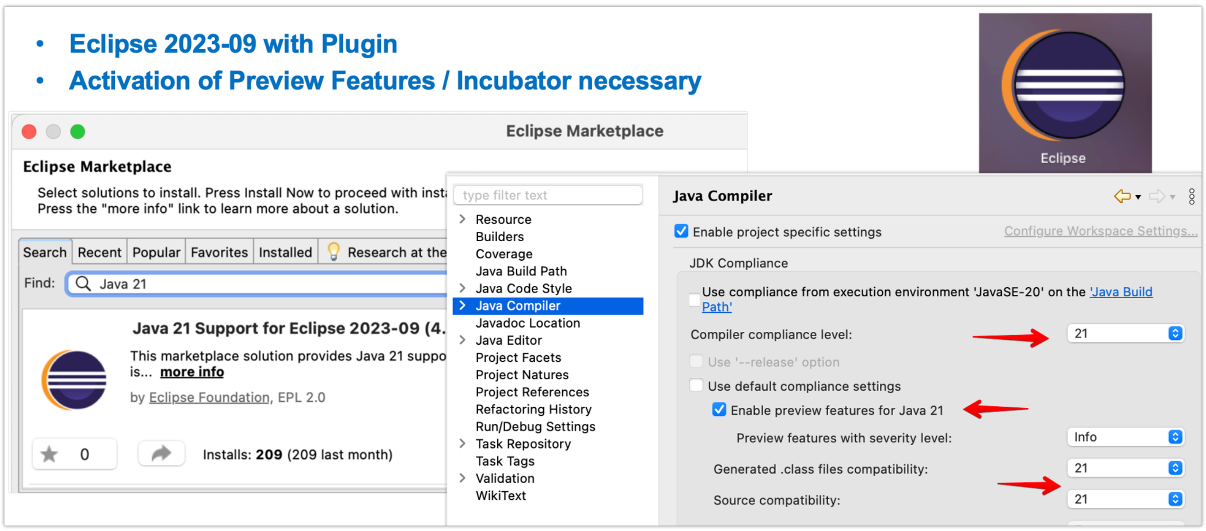
Course management and contact

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

Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with insta
Press the "more info" link to learn more about a solution.

Find: Java 21

Java 21 Support for Eclipse 2023-09 (4.
This marketplace solution provides Java 21 suppo is... more info
by Eclipse Foundation, EPL 2.0
Installs: 209 (209 last month)

Java Compiler

☑ Enable project specific settings

JDK Compliance

☐ Use compliance from execution environment 'JavaSE-20' on the 'Java Build Path'

Compiler compliance level: 21

☐ Use '--release' option

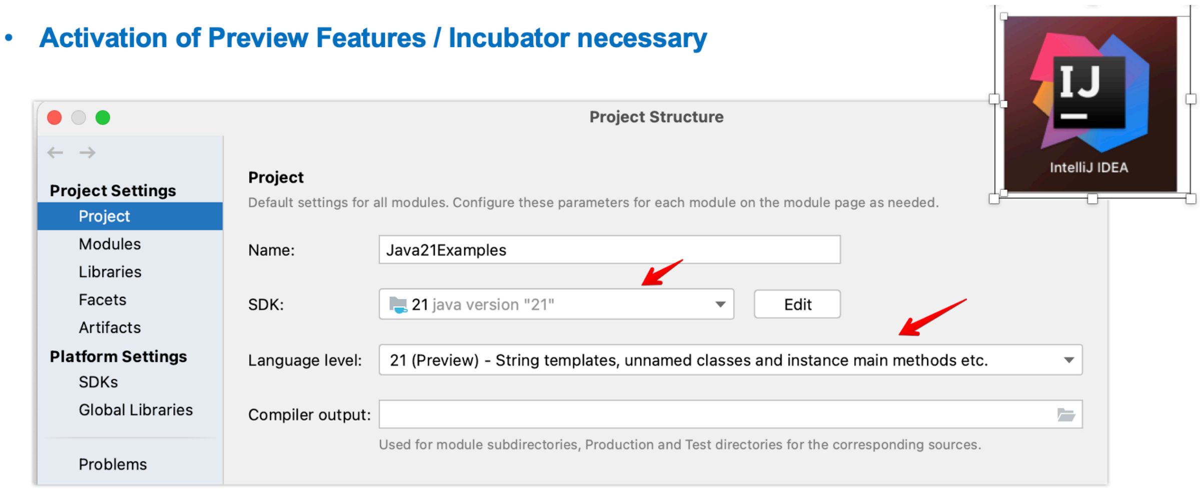
☐ Use default compliance settings

☑ Enable preview features for Java 21

Preview features with severity level: Info

Generated .class files compatibility: 21

Source compatibility: 21



- Activation of Preview Features / Incubator necessary

Project Structure

Project
Default settings for all modules. Configure these parameters for each module on the module page as needed.

Name: Java21Examples

SDK: 21 java version "21"

Language level: 21 (Preview) - String templates, unnamed classes and instance main methods etc.

Compiler output:

Used for module subdirectories, Production and Test directories for the corresponding sources.

PART 1: Syntax Enhancements up to Java 17 LTS

Objective: In this section we will look at syntax extensions up to Java 17 LTS.

Exercise 1 – Syntax changes for switch

Simplify the following source code containing a conventional switch-case with the new syntax.

```
private static void dumbEvenOddChecker(int value)
{
      String result;

      switch (value)
      {
      case 1:
      case 3:
      case 5:
      case 7:
      case 9:
            result = "odd";
            break;

      case 0, 2, 4, 6, 8, 10:
            result = "even";
            break;

      default:
            result = "only implemented for values <= 10";
      }

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

Exercise 1 a

First use the arrow syntax to write the method shorter and clearer.

Exercise 1 b

Now use the option to specify returns directly and change the signature to `String dumbEvenOddChecker(int value)`.

Exercise 1 c

Convert the above code so that it uses the special form "yield with return value".

Exercise 2 – Text Blocks

Simplify the following source code with a conventional string that spans multiple lines and use the syntax introduced in modern Java.

```
String multiLineStringOld = "THIS IS\n" +
            "A MULTI\n" +
            "LINE STRING\n" +
            "WITH A BACKSLASH \\\n";

String multiLineHtmlOld = "<html>\n" +
           "    <body>\n" +
           "        <p>Hello, world</p>\n" +
           "    </body>\n" +
           "</html>";

String java13FeatureObjOld = ""
              + "{\n"
              + "    version: \"Java13\",\n"
              + "    feature: \"text blocks\",\n"
              + "    attention: \"preview!\"\n"
              + "}\n";
```

Exercise 3 – Text Blocks with Placeholders

Simplify the following source code with a conventional string that spans multiple lines and use the syntax introduced in modern Java:

```
String multiLineStringWithPlaceholdersOld =
       String.format("HELLO \"%s\"!\n" +
                     "  HAVE %s\n" +
                     "  NICE \"%s\"!",
                     new Object[]{"WORLD", "A", "DAY"});

System.out.println(multiLineStringWithPlaceholdersOld);
```

Produce the following output with the new syntax:

```
HELLO "WORLD"!
  HAVE A
  NICE "DAY"!
```

Exercise 4 – Record Basics

Two simple classes are given, which represent pure data containers and thus only provide a public attribute. Convert them into records:

```
class Square
{
     public final double sideLength;

     public Square(final double sideLength)
     {
          this.sideLength = sideLength;
     }
}

class Circle
{
     public final double radius;

     public Circle(final double radius)
     {
          this.radius = radius;
     }
}
```

What are the advantages – apart from the shorter spelling – of using records instead of separate classes?

Exercise 5 – Record

Based on the record shown below, create two methods that produce JSON and XML output. Add a validation check so that the last name and first name are at least 3 characters long and the birthday is not in the future.

```
record Person(String firstName, String lastName,
              LocalDate birthday) {}
```

```
<Person>
     <firstName>Michael</firstName>
     <lastName>Inden</lastName>
     <birthday>1971-02-07</birthday>
</Person>
```

```
{
     "firstName" : "Michael",
     "lastName" : "Inden",
     "birthday" : "1971-02-07"
}
```

Exercise 6 – `instanceof` Basics

Given are the following lines with an `instanceof` and a cast. Simplify the whole thing with the new features of modern Java.

```
Object obj ="BITTE ein BIT";

if (obj instanceof String)
{
     final String str = (String)obj;
     if (str.contains("BITTE"))
     {
         System.out.println("It contains the magic word!");
    }
}
```

Exercise 7 – `instanceof` and `record`

Simplify the source code using the syntax innovations in `instanceof` and then using the special features in Records.

```
record Square(double sideLength) {
}

record Circle(double radius) {
}

public double computeAreaOld(final Object figure)
{
     if (figure instanceof Square)
     {
          final Square square = (Square) figure;
          return square.sideLength * square.sideLength;
     }
     else if (figure instanceof Circle)
     {
          final Circle circle = (Circle) figure;
          return circle.radius * circle.radius * Math.PI;
     }
     throw new IllegalArgumentException("figure is not a
recognized figure");
}
```

Although we have certainly achieved an improvement in terms of readability and number of lines by using `instanceof`, several of these checks indicate a violation of the open-closed principle, one of the SOLID principles of good design. What would an object-oriented design be? The answer in this case is simple: Often `instanceof` checks can be avoided by introducing a base type. **Simplify the whole thing with an interface `BaseFigure` and use it appropriately.**

Bonus
Introduce with rectangles another type of figures. However, this should not require any modifications in the `computeArea()` method.

PART 2: API and JVM innovations up to 17 LTS

Objective: In this section, we will cover JVM enhancements and API innovations up to Java 17 LTS.

Exercise 1 – HTTP/2

The following HTTP communication is given, which accesses the Oracle web page and prints it textually.

```
private static void readOraclePageJdk8() throws MalformedURLException,
                                                IOException
{
    final URL oracleUrl = new URL("https://www.oracle.com/index.html");
    final URLConnection connection = oracleUrl.openConnection();

    System.out.println(readContent(connection.getInputStream());
}

public static String readContent(final InputStream is) throws IOException
{
    try (final InputStreamReader isr = new InputStreamReader(is);
         final BufferedReader br = new BufferedReader(isr))
    {
        final StringBuilder content = new StringBuilder();
        String line;
        while ((line = br.readLine()) != null)
        {
            content.append(line + "\n");
        }
        return content.toString();
    }
}
```

Exercise 1a

Convert the source code to use the new HTTP/2 API from JDK 11. Use the classes `HttpRequest` and `HttpResponse` and create a method `printResponseInfo(HttpResponse)`, which reads the body analogous to the method `readContent(InputStream)` above and also provides the HTTP status code.
Start with the following program fragment:

```
private static void readOraclePageJdk11() throws URISyntaxException,
                                                 IOException,
                                                 InterruptedException
{
    final URI uri = new URI("https://www.oracle.com/index.html");
    final HttpClient httpClient = // TODO
    final HttpRequest request = // TODO

    printResponseInfo(response);
}
```

Exercise 1b

Start the queries asynchronously by calling `sendAsync()` and process the received `CompletableFuture<HttpResponse>`.

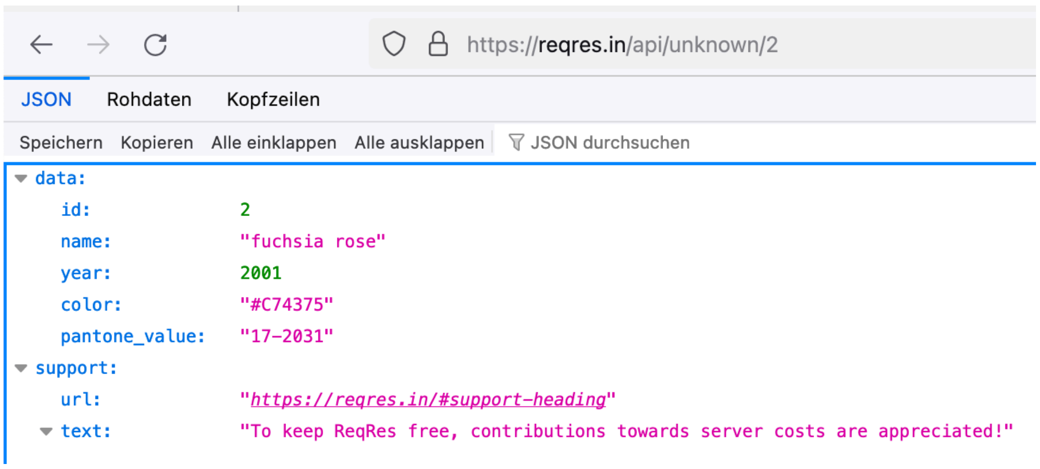
Exercise 2 – Direct Compilation

Exercise 2a

Write a HelloWorld class to output a greeting on the console; save it in a Java file of the same name. Run it directly with the command `java`.

Exercise 2b

Write a class `PerformGetWithHttpClient` to perform a REST call such as GET `https://reqres.in/api/unknown/2`. Save the program in a Java file with the same name. Run this directly with the command `java`. The data returned should be like that returned by the call from the browser.



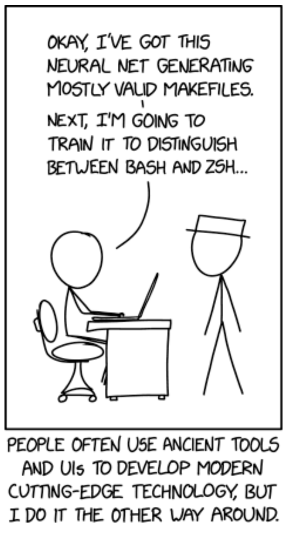
https://reqres.in/api/unknown/2

JSON Rohdaten Kopfzeilen

Speichern Kopieren Alle einklappen Alle ausklappen JSON durchsuchen

```
▼ data:
    id:              2
    name:            "fuchsia rose"
    year:            2001
    color:           "#C74375"
    pantone_value:   "17-2031"
▼ support:
    url:             "https://reqres.in/#support-heading"
  ▼ text:            "To keep ReqRes free, contributions towards server costs are appreciated!"
```

Alternatively, you can also use the page `xkcd`, for example the graphic with this address: https://imgs.xkcd.com/comics/modern_tools.png


OKAY, I'VE GOT THIS NEURAL NET GENERATING MOSTLY VALID MAKEFILES.
NEXT, I'M GOING TO TRAIN IT TO DISTINGUISH BETWEEN BASH AND ZSH...
PEOPLE OFTEN USE ANCIENT TOOLS AND UIs TO DEVELOP MODERN CUTTING-EDGE TECHNOLOGY, BUT I DO IT THE OTHER WAY AROUND.

Write a class that downloads this graphic and saves it as a PNG file.

Bonus

Create a bash script `exec_rest.sh` for direct execution, remember the correct rights (`chmod u+x`). For Windows, use the workaround via `.bat` file.

Exercise 3 – JPackage

Have a look at the PackackingDemo project and modify it to use another dependency, for example on Apache Commons.

```
jpackage --input target/ --name JPackageDemoApp
         --main-jar PackagingDemoExamples-1.0.0-SNAPSHOT.jar
         --main-class de.java17.ApplicationExample
         --type <dmg / msi / …>
```

Supplement as needed:

```
--java-options '--enable-preview'
```

PART 3a: Enhancements in Java 18 to 21 LTS

Objective: In this section, we look at enhancements in Java 18 to 21 LTS.

Aufgabe 1 – Convert to Record Patterns

Given is a definition of a journey through the following records:

```
record Person(String firstname, String lastname, LocalDate birthday) {
}

record TravelInfo(LocalDate start, Duration maxTravellingTime) {
}

record City(Integer zipCode, String name) {
}

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

In addition, various consistency checks and validations are performed that access nested components. For this purpose, one sometimes sees implementations - especially in legacy code - that contain deeply nested `if`s and various `null` checks.

```
static boolean checkFirstNameTravelTimeAndDestZipCode(final Object obj) {
    if (obj instanceof Journey journey) {
        if (journey.person() != null) {
            var person = journey.person();

            if (journey.travelInfo() != null) {
                var travelInfo = journey.travelInfo();

                if (journey.to() != null) {
                    var to = journey.to();
```

The task now is to implement the whole thing in a more understandable and compact way using record patterns.

Bonus: Simplify the specification with `var`.

Exercise 2 – Use Record Patterns for recursive calls

Given are definitions of some graphical figures by the following records:

```java
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 {
}
```

In addition, the following method is defined, which multiplies the x- and y-coordinates of a point. This has already been implemented for `Point`. The `switch` should be extended to include `case`s for `Line` and `Triangle` in which we should be using recursive calls. For the calculation, the respective sub-components in the form of points should be added by calling the already existing method `process()`.

```java
static int process(Figure figure) {
    return switch (figure) {
        case Point(int x, int y) -> x * y;
        // TODO
        default -> throw new IllegalStateException("Unexpected value: " +
                                                    figure);
    };
}
```

Exercise 3 – Convert to Virtual Threads

Given an execution of various tasks using a classic `ExecutorService` and a given pool size of 50:

```java
try (var executor = Executors.newFixedThreadPool(100)) {
    IntStream.range(0, 10_000).forEach(i -> {
        executor.submit(() -> {
            Thread.sleep(Duration.ofSeconds(5));

            System.out.println("Task " + i + " finished!");
            return i;
        });
    });
}
```

Convert the whole thing to use virtual threads. Use a suitable method in `Thread`.

Exercise 4 – Experiment with Sequenced Collections

Given the following class with some TODO comments, the first prime numbers are to be prepared as a list. In addition, elements are to be inserted at the front and at the back and a reverse sequence is to be generated.

```
private static void primeNumbers()
{
    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
}
```

Bonus

Experiment with the interface `SequencedSet`<E> and use the appropriate methods to create a sorted set consisting of the letters A, B and C:

```
private static void createABCSet()
{
    Set<String> numbers = new LinkedHashSet<>();

    // TODO

    // TODO print first and last element
    // TODO print reverser order
}
```

PART 3b: Enhancements in Java 18 to 21 LTS Preview + Incubator

Objective: In this section, we look at enhancements in Java 18 to 21 LTS, which are not yet final.

Exercise 5 – Convert with Structured Concurrency

Given an execution of various tasks using a classic `ExecutorService` and a “combining” of the calculation results to a console output:

```
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());
    }
}
```

Use Structured Concurrency to replace the `ExecutorService` and use the `ShutdownOnFailure` strategy to clarify processing in case of failure. Analyze the processing in case of failure.

Let's run the method with both value assignments:

```
jshell> import java.util.concurrent.*

jshell> executeTasks(false)
 result: 1 / 2 / 3

jshell> executeTasks(true)
| Ausnahme java.util.concurrent.ExecutionException:
java.lang.IllegalStateException: FORCED BUG
   at FutureTask.report (FutureTask.java:122)
   at FutureTask.get (FutureTask.java:191)
   at executeTasks (#19:16)
   at (#21:1)
|  Verursacht von: java.lang.IllegalStateException: FORCED BUG
|        at lambda$executeTasks$1 (#19:8)
```

Exercise 6 – Experiment with Template Processor

Write your own template processor which encloses the values with [[and]] or alternatively with a ' in front and behind:

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

=>

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

a) Use `fragments()` and `values()` and a loop.
b) Simplify the whole thing with `interpolate()` and the Stream API.
c) Use a parameterizable lambda to make the start and end sequence freely configurable.
d) Create a template processor that works similar to the f-strings in Python (`f"Calculation: {x} + {y} = {x + y}"`) without directly referencing STR.

Exercise 7 – Experiment with Unnamed Patterns and Variables

Simplify the following method by using Unnamed Patterns and Variables to increase readability and understandability – take advantage of the fact that the IDEs show unused variables:

```java
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;
}
```