Hands On: Best of Java 9 – 13 Workshop Exercises

Michael Inden
CTO & Teamlead SW-Development & Head ASMIQ Academy
ASMIQ AG, Geerenweg 2, 8048 Zürich
E-Mail: michael.inden@asmiq.ch
Course offer: https://asmiq.ch/
Blog: https://jaxenter.de/author/minden

PART 1: Syntax Enhancements in Java 9 – 12

Exercise 1 – Getting to know `var`

Get to know the new reserved word `var`.

Task 1a

Start the JShell or an IDE of your choice. Create a method `funWithVar()`. Define the variables `name` and `age` with the values `Mike` and 47.

Task 1b

Expand your know-how regarding `var` and generics. Use it for the following definition. Initially create a local variable `personsAndAges` and then simplify with `var`:

```
Map.of("Tim", 47, "Tom", 7, "Mike", 47);
```

PART 2: News and API-Changes in Java 9 – 12

Exercise 2 – Collection-Factory-Methods

Define a list, set, and map using the Collection Factory methods `of()` which are newly introduced in JDK 9. The following program fragment with JDK 8 serves as a starting point.

```
private static void collectionsExampleJdk8()
{
        final List<String> names = Arrays.asList("Tim", "Tom", "Mike");
        System.out.println(names);

        final Set<Integer> numbers = new TreeSet<>();
        numbers.add(1);
        numbers.add(3);
        numbers.add(4);
        numbers.add(2);
        System.out.println(numbers);

        final Map<Integer, String> mapping = new HashMap<>();
        mapping.put(5, "five");
        mapping.put(6, "six");
        mapping.put(7, "seven");
        System.out.println(mapping);
}
```

Use a static import as follows:

```
import static java.util.Map.entry;
```

Exercise 3 – Streams Take / Drop While

Extract the head and body information with appropriate predicates and the previously presented methods.

```
final Predicate<String> isBodyStart = // TODO

final Predicate<String> isBodyEnd = // TODO

final List<String> tokens = List.of("<html>",
      "<head>", "<title>This is TTLE</title>", "</head>",
      "<body>",
           "<h1>Hello everyone @ Oracle Code One</h1>",
           "<p>Welcome to this hands-on lab</p>",
      "</body>",
    "</html>");

extractor(tokens, isBodyStart, isBodyEnd).forEach(System.out::println);
```

Tip: Create a help method with the following signature:

```
private static List<String> extractor(final List<String> tokens,
                                      final Predicate<String> isStart,
                                      final Predicate<String> isEnd)
```

Exercise 4 – The Class `Optional`

The following program fragment is given, which executes a multi-level search: first in the cache, then in memory, and finally in the database (just simulated here). This search chain is indicated by three `find()` methods and implemented as shown below.

```
static Optional<String> multiFindCustomerJdk8(final String customerId)
{
        final Optional<String> opt1 = findInCache(customerId);
        if (opt1.isPresent())
        {
                return opt1;
        }
        else
        {
                final Optional<String> opt2 = findInMemory(customerId);
                if (opt2.isPresent())
                {
                        return opt2;
                }
                else
                {
                        return findInDb(customerId);
                }
        }
}
```

```
private static Optional<String> findInMemory(final String customerId)
{
    final Stream<String> customers = Stream.of("Tim", "Tom", "Mike", "Andy");

    return customers.filter(name -> name.contains(customerId))
                    .findFirst();
}

private static Optional<String> findInCache(final String customerId)
{
    return Optional.empty();
}

private static Optional<String> findInDb(final String customerId)
{
    return Optional.empty();
}
```

Simplify the call chain using the new methods from the `Optional<T>` class. See how it all becomes clearer.

Exercise 5 – The Class `LocalDate`

Get to know useful things in the LocalDate class.

Task 5a

Write a program that counts all Sundays in 2017.

Task 5b

Write a program that determines all Fridays 13th in the years from 2013 to 2017:

```
final LocalDate start = LocalDate.of(2013, 1, 1);
final LocalDate end = LocalDate.of(2018, 1, 1);
```

The following values should appear as the result:

```
[2013-09-13, 2013-12-13, 2014-06-13, 2015-02-13, 2015-03-13, 2015-11-13,
 2016-05-13, 2017-01-13, 2017-10-13]
```

Exercise 6: Strings

The processing of strings has been made easier in Java 11 with some useful methods.

Task 6a

Use the following stream as input:

```
Stream.of(2,4,7,3,1,9,5)
```

Implement a method that outputs the numbers one below the other, repeated as often as the digit as follows:

```
22
4444
7777777
333
1
999999999
55555
```

Task 6b

Modify the output so that the numbers are right aligned with a maximum of 10 characters:

```
‘      4444’
‘   7777777’
‘ 999999999’
```

Tip: Use a helper method

```
private static String formatRightAligned(final int num,
                                         final int desiredLength)
{
       // TODO
}
```

Exercise 7: Strings und Files

Until Java 11 it was a bit difficult to write texts directly into a file or to read them from it. Now you can use the methods `writeString()` and `readString()` from the class `Files`. Use them to write the following lines to a file. Read this again and prepare a `List<String>` from it.

```
1: One
2: Two
3: Three
```

PART 3: Multi-Threading with `CompletableFuture`

- No exercises -

PART 4: HTTP/2

Exercise 8 – HTTP/2

The following HTTP communication is given, which accesses the Oracle Web page and formats 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();

    final String content = readContent(connection.getInputStream());
    System.out.println(content);
}

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();
    }
}
```

Task 8a

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
    final BodyHandler<String> asString = // TODO
    final HttpResponse<String> response = // TODO

    printResponseInfo(response);
}
```

```
private static void printResponseInfo(final HttpResponse<String> response)
{
    final int responseCode = response.statusCode();
    final String responseBody = response.body();
    final HttpHeaders headers = response.headers();

    System.out.println("Status:  " + responseCode);
    System.out.println("Body:    " + responseBody);
    System.out.println("Headers: " + headers.map());
}
```

Task 8b
Start the queries asynchronously by calling `sendAsync()` and process the received `CompletableFuture<HttpResponse>`.

PART 5: News in Java 12 / 13

Exercise 9 – Syntax changes for switch

Simplify the following source code with a conventional switch-case with the new syntax of Java 12 / 13.

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

      switch (value)
      {
      case 1, 3, 5, 7, 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);
}
```

Task 9a

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

Task 9b
Now use the possibility to specify returns directly and change the signature in `String dumpEvenOddChecker(int value)`.

Exercise 10 – Text Blocks

Simplify the following source code with a conventional string that spans multiple lines and use the syntax introduced in Java 13.

```
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";
```

Task 11 – Text Blocks with Placeholders

Simplify the following source code with a conventional string that spans multiple lines and use the syntax introduced in Java 13:

```
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"!
```