Workshop:
Best of Java 11 – 19
Exercises

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 17 is installed, ideally Java 18 or the latest Java 19
2) Current Eclipse 2022-09 is installed (Alternatively: NetBeans or IntelliJ IDEA)

Attendees

- Developers with Java experience, as well as
- SW-Architects, who would like to learn/evaluate Java 9 to 17.

Course management and contact

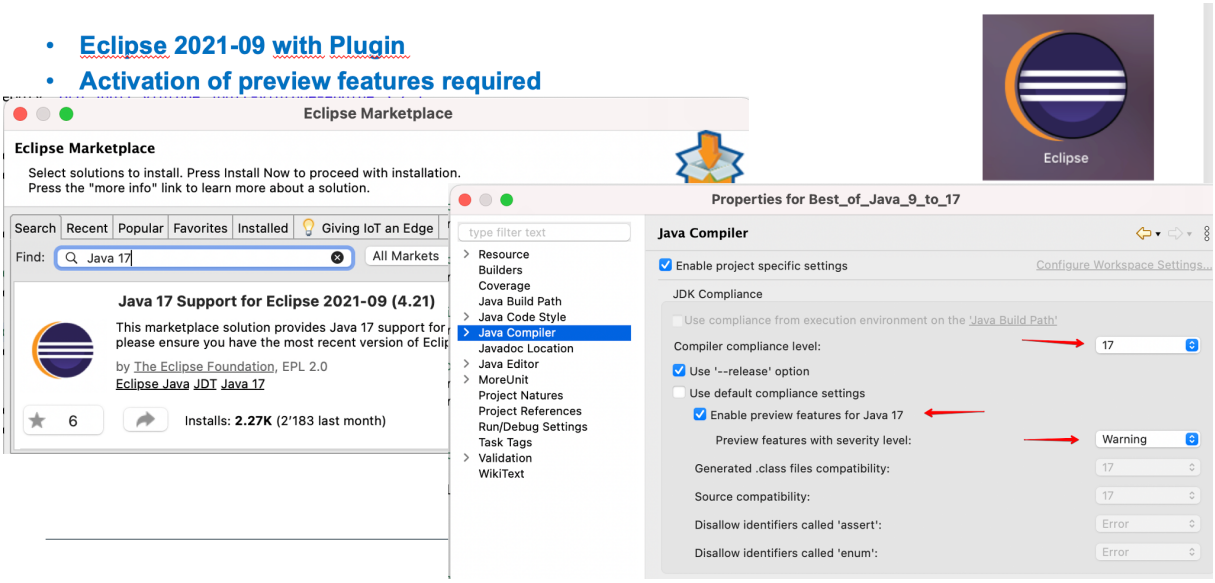
Michael Inden
Head of Development, independent SW Consultant, Author, and Trainer
E-Mail: michael_inden@hotmail.com
Blog: https://jaxenter.de/author/minden

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

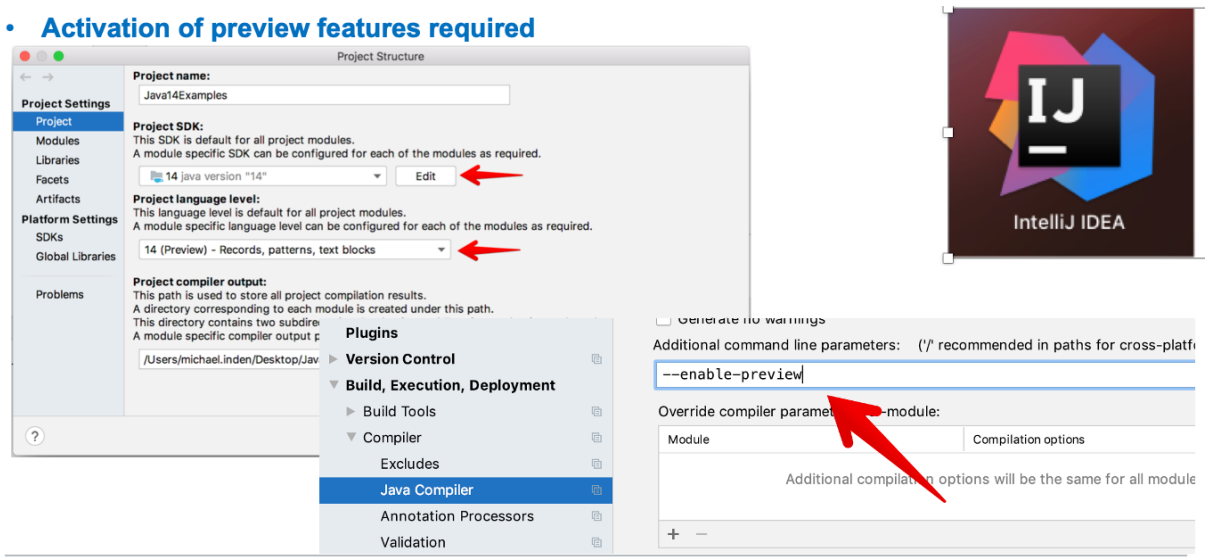
Configuration Eclipse / IntelliJ

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

- Eclipse 2021-09 with Plugin
- Activation of preview features required



- Activation of preview features required



PART 1: Syntax Enhancements / News / API-Changes up to Java 11

Objective: Learn about syntax innovations and various API extensions in Java up to Version 11 using examples.

Exercise 1 – Getting to know `var`

Get to know the new reserved word `var`.

Exercise 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`.

Exercise 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);
```

Exercise 1c

Simplify the following definition with var. What should be considered? Where's the difference?

```
List<String> names = new ArrayList<>();
ArrayList<String> names2 = new ArrayList<>();
```

Exercise 1d

Why do the following lambdas cause problems? How do you solve these?

```
var isEven = n -> n % 2 == 0;
var isEmpty = String::isEmpty;
```

Why then compile the following?

```
Predicate<Long> isEven = n -> n % 2 == 0;
var isOdd = isEven.negate();
```

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.

Use a static import as follows: `import static java.util.Map.entry;`

```
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);
}
```

Exercise 3 – Streams

The Stream API has been enhanced with methods that allow you to read elements only as long as a condition is met or to skip elements while a condition is met. The following two streams are used as starting point:

```
final Stream<String> values1 = Stream.of("a", "b", "c", "", "e", "f");
final Stream<Integer> values2 = Stream.of(1, 2, 3, 11, 22, 33, 7, 10);
```

Exercise 3a

Determine values from the stream `values1` until an empty string is found. Print out the values to the console.

Exercise 3b

In the stream `values2`, skip the values as long as the value is smaller than 10. Print out the values to the console.

Exercise 3c

What is the difference between the two methods `dropWhile()` and `filter()`?

Tip: The expected result is as follows:

```
takeWhile
a
b
c

dropWhile
11
22
33
7
10

with filter
11
22
33
10
```

Exercise 4– 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 @ JAX London</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 5 – The Class `Optional`

Given the following method, which performs a person search and, depending on the result of the match, calls the method `doHappyCase(person)` or otherwise `doErrorCase()`.

```
private static void findJdk8()
{
      final Optional<Person> opt = findPersonByName("Tim");
      if (opt.isPresent())
      {
            doHappyCase(opt.get());
      }
      else
      {
            doErrorCase();
      }

      final Optional<Person> opt2 = findPersonByName("UNKNOWN");
      if (opt2.isPresent())
      {
            doHappyCase(opt2.get());
      }
      else
      {
            doErrorCase();
      }
}
```

```
private static Optional<Person> findPersonByName(final String searchFor)
{
	final Stream<Person> persons = Stream.of(new Person("Mike"),
	                                         new Person("Tim"),
	                                         new Person("Tom"));

	return persons.filter(person -> person.getName().equals(searchFor)).
	               findFirst();

}

private static void doHappyCase(final Person person)
{
	System.out.println("Result: " + person);
}

private static void doErrorCase()
{
	System.out.println("not found");
}
```

Use the new methods from class `Optional<T>` to make the program fragment more elegant within a findJdk9() method that produces the following outputs like findJdk8():

```
Result: Person: Tim
not found$
```

Exercise 6 – 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 7: Strings

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

Exercise 7a

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
```

Exercise 7b
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 7c
Modify the whole thing so that instead of spaces leading zeros are output, as follows:

```
‘0000004444’
‘0007777777’
‘0999999999’
```

Bonus: Expand the whole thing so that any fill characters can be used.

Exercise 7d
Modify the output so that the largest numbers are output last. Find at least two variants:

```
Stream.of(2,4,7,3,1,9,5).sorted().map(mapper1)
Stream.of(2,4,7,3,1,9,5).map(mapper2)
```

Where is the difference?

PART 2: Misc in Java 11

Objective: In this section we want to learn about some practical API extensions: `Arrays` as well as `LocalDate` and `InputStream`.

Exercise 1 – The Class `LocalDate`

Get to know useful things in the LocalDate class.

Exercise 1a

Write a program that counts all Sundays in 2017.

Exercise 1b

List the Sundays, starting with the 5th and ending with the 10th. The result should be as follows:

```
[2017-02-05, 2017-02-12, 2017-02-19, 2017-02-26, 2017-03-05]
```

Exercise 2 – The class `LocalDate`

Learn useful things in the LocalDate class.

Exercise 2a

Write a program that determines all Fridays 13th in the years 2013 to 2017. Use the following lines as starting point:

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

As a result, the following values should appear:

```
[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]
```

Group the occurrences by year. The following issues should appear:

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

Exercise 2b

How many times have there been February 29 between the beginning of 2010 and the end of 2017?

```
final LocalDate start2010 = LocalDate.of(2010, 1, 1);
final LocalDate end2017 = LocalDate.of(2018, 1, 1);
```

Exercise 2c
How often was your birthday a Sunday between the beginning of 2010 and the end of 2017? For the 7th of February the following result should be calculated:

```
My Birthday on Sunday between 2010-2017: [2010-02-07, 2016-02-07]
```

Exercise 3 – The Class `InputStream`

Write a program that creates a copy of a file given by a filename. Simplified Java 8-based source code below:

```
private static void copyFileUsingStream(final File source,
                                        final File dest) throws IOException
{
    try (final InputStream is = new FileInputStream(source);
         final OutputStream os = new FileOutputStream(dest))
    {
        final byte[] buffer = new byte[2048];
        int length;
        while ((length = is.read(buffer)) > 0)
        {
            os.write(buffer, 0, length);
        }
    }
}
```

Exercise 4 – The Class `InputStream`

Simplify the following Java 8-based source code and write methods that do the following:

a) Read in the data from an InputStream completely.
b) Only read the first 6 characters from an InputStream.
c) Transfer all data from an InputStream to an OutputStream.

```
public static void main(final String[] args) throws IOException
{
    final byte[] buffer = { 72, 65, 76, 76, 79 };

    final byte[] resultJdk8 = readAllBytesJdk8(/* TODO */);
    System.out.println(Arrays.toString(resultJdk8));

    transferToJdk8(/* TODO */ System.out);
}

private static byte[] readAllBytesJdk8(final InputStream is) throws IOException
{
    try (final ByteArrayOutputStream os = new ByteArrayOutputStream())
    {
        transferToJdk8(is, os);
        os.flush();

        return os.toByteArray();
    }
}
```

```
private static void transferToJdk8(final InputStream in,
                                   final OutputStream out) throws IOException
{
    final byte[] buffer = new byte[1024];
    int len;
    while ((len = in.read(buffer)) != -1)
    {
        out.write(buffer, 0, len);
    }
}
```

Exercise 5: Enhancement in the class Reader

Transfer the content from a `StringReader` to a file hello.txt. Read this again and give out the content. Complete the following lines:

```
var textFile = new File("hello.txt");
var sr = new StringReader("Hello\nWorld");
try (Writer bfw = null /*TODO*/)
{
      // TODO
}

var sw = new StringWriter();
try (Reader bfr = null /*TODO*/)
{
      // TODO
}
```

Exercise 6: 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
```

Exercise 7 – 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();
    }
}
```

Exercise 7a

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());
    }
```

Exercise 7b
Start the queries asynchronously by calling `sendAsync()` and process the received `CompletableFuture<HttpResponse>`.

Aufgabe 8 – REST-CALL

Imagine that you need to calculate the exchange rates for a period of several months. On the net, these can be retrieved via REST at http://data.fixer.io/ (previously freely available* at https://exchangeratesapi.io/). For this you can use the HTTP/2-API profitably, for which we write a method `performGet()`. To define the time period we use the innovation `datesUntil()` to create a `Stream<LocalDate>` with monthly increments. With a `TemporalAdjuster` we jump to the end of the month and then execute the GET call.

For the period January until October 2021 we expect the following issues:

```
2021-01-31 reported
{"success":true,"timestamp":1612137599,"historical":true,"base":"EUR","date":"2021-
01-31","rates":{"CHF":1.080718}}
2021-02-28 reported
{"success":true,"timestamp":1614556799,"historical":true,"base":"EUR","date":"2021-
02-28","rates":{"CHF":1.09705}}
2021-03-31 reported
{"success":true,"timestamp":1617235199,"historical":true,"base":"EUR","date":"2021-
03-31","rates":{"CHF":1.107039}}
2021-04-30 reported
{"success":true,"timestamp":1619827199,"historical":true,"base":"EUR","date":"2021-
04-30","rates":{"CHF":1.09715}}
2021-05-31 reported
{"success":true,"timestamp":1622505599,"historical":true,"base":"EUR","date":"2021-
05-31","rates":{"CHF":1.099181}}
2021-06-30 reported
{"success":true,"timestamp":1625097599,"historical":true,"base":"EUR","date":"2021-
06-30","rates":{"CHF":1.096862}}
2021-07-31 reported
{"success":true,"timestamp":1627775999,"historical":true,"base":"EUR","date":"2021-
07-31","rates":{"CHF":1.074508}}
2021-08-31 reported
{"success":true,"timestamp":1630454399,"historical":true,"base":"EUR","date":"2021-
08-31","rates":{"CHF":1.081158}}
2021-09-30 reported
{"success":true,"timestamp":1633046399,"historical":true,"base":"EUR","date":"2021-
09-30","rates":{"CHF":1.079352}}
```

***meanwhile you need a key, for the workshop you can use the following:**
`access_key=5e9375c8c908bdc0d6e6a356ea14b860`

Exercise 9 – Direct compilation and execution

Write a HelloWorld class in the package `java11.direct.compilation` and save it in a Java file of the same name (or use `cat > HelloWorld.java` *Ctrl-C). Then execute these directly with the command java.

```
public class Exercise1_HelloWorld
{
       public static void main(String... args)
       {
              System.out.println("Hello Execute After Compile");

              var procesInfo = ProcessHandle.current().info();
              System.out.println(procesInfo);
    }

}
```

What is special about that?

Bonus
Create a bash script `exec_hello.sh` to run directly, remember the correct permissions (`chmod u+x`).

Exercise 10 – JAXB

Now experiment with the JAXBExample project and change the implementation of the Book class so that it now uses the `java.time.LocalDate` class instead of `java.util.Date`.

PART 3: Syntax Enhancements in Java 12 to 17

Objective: In this section we will look at syntax extensions in Java 12 to 17.

Exercise 1 – Syntax changes for switch

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

```
private static void dumbEvenOddChecker(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);
}
```

Exercise 1a

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

Exercise 1b

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

Exercise 1c

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 4: API-News in Java 12 to 17

Objective: In this section, we look at extensions and API innovations in Java 12 through 17.

Exercise 1: The class `CompactNumberFormat`

Write a program to output and parse 1,000, 1,000,000 and 1,000,000,000 depending on locale and style. Use the Locale GERMANY for SHORT and ITALY for LONG.

Use the following values for parsing:

```
List.of("13 KILO", "1 Mio.", "1 Mrd.")
List.of("1 mille", "1 milione")
```

Exercise 2: Strings

The processing of strings has been extended by two methods in Java 12. First, get to know `indent()` better.

Exercise 2a

Enter the following input by 7 characters, output this and remove 3 more characters from the indentation.

```
String originalString = "first_line\nsecond_line\nlast_line";
```

Exercise 2b

What happens if you put a left-aligned text with negative values for the indent? What happens if you use an -10 index for subsequent input?

```
String multipleIndendedString =
       "class A {\n    public static void main(String[] args) {" +
       "\n        System.out.println(\"Hello\"";
```

Exercise 3: Strings

The processing of strings has been extended by two methods in Java 12. Learn more about `transform()`. Given the following comma-separated input:

```
var csvText = "HELLO,WORKSHOP,PARTICIPANTS,!,LET'S,HAVE,FUN";
```

Exercise 3a

Convert these completely to lowercase and replace the commas with spaces.

Exercise 3b

Use other transformations and replace HELLO with the Swiss greeting "GRÜEZI", then split the whole thing into individual components, resulting in the following list as result:

```
[GRÜEZI, workshop, participants, !, let's, have, fun]
```

Exercise 4: Teeing-Collector

Use the Teeing Collector to find both minimum and maximum in one pass. Start with the following lines:

```
Stream<String> values = Stream.of("CCC", "BB", "A", "DDDD");
List<Optional<String>> optMinMax = values.collect(teeing(…
```

BONUS: Experiment with comparing on length instead of alphabetical sorting.

Exercise 5: Teeing- Collector

Vary the `BiFunction` to properly influence the results of the Teeing Collector.
Start with the following lines and complete them at the marked places:

```
var names = Stream.of("Michael", "Tim", "Tom", "Mike", "Bernd");

final Predicate<String> startsWithMi = text -> text.startsWith("Mi");
final Predicate<String> isShort = text -> text.length() <= 4;

final BiFunction<List<String>, List<String>, List<List<String>>>
      combineResults = (list1, list2) -> List.of(list1, list2);

final BiFunction<List<String>, List<String>, Set<String>>
      combineResultsUnique = null; // TODO;

final BiFunction<List<String>, List<String>, Set<String>>
      combineResultsIntersection = null; // TODO;

var result = names.collect(teeing(
                           filtering(startsWithMi, toList()),
                           filtering(isShort, toList()),
                           combineResults));
```

The expected results are

- `combineResults: [[Michael, Mike], [Tim, Tom, Mike]]`
- `combineResultsUnique: [Mike, Tom, Michael, Tim]`
- `combineResultsIntersection: [Mike]`

Exercise 6: Teeing-Collector

Use a Teeing Collector to find all cities in Europe by name, as well as the number of cities in Asia. Start with the following lines and convert the City class into a record.

```
Stream<City> exampleCities = Stream.of(
                  new City("Zürich", "Europe"),
                  new City("Bremen", "Europe"),
                  new City("Kiel", "Europe"),
                  new City("San Francisco", "America"),
                  new City("Aachen", "Europe"),
                  new City("Hong Kong", "Asia"),
                  new City("Tokyo", "Asia"));

Predicate<City> isInEurope = city -> city.locatedIn("Europe");
Predicate<City> isInAsia = city -> city.locatedIn("Asia");

var result = exampleCities.collect(teeing(…
```

Given the class City is as follows:

```
static class City
{
      private final String name;
      private final String region;

      public City(final String name, final String region)
      {
            this.name = name;
            this.region = region;
      }

      public String getName()
      {
            return name;
      }

      public String getRegion()
      {
            return region;
      }

      public boolean locatedIn(final String region)
      {
            return this.region.equalsIgnoreCase(region);
      }
}
```

PART 5: JVM-Neuerungen in Java 12 bis 17

Lernziel: In diesem Abschnitt beschäftigen wir uns mit JVM-Erweiterungen in Java 12 bis 17.

Exercise 1 – JMH

Create a JMH project using the Maven command from the slides. Expand this project and create a simple benchmark (for inspiration: open the existing project from the JMH-Benchmarking.zip and copy one of the benchmark classes). Build the project and run the benchmark(s).

Exercise 2 – 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'
```

Exercise 3 – JShell-API

Use the JShell API to perform some dynamic calculations

- `int result = x * y;`
- `var today = LocalDate.now();`
- `var values = List.of(1,  2,  3,  4);`

List all variables and their values. Use the `variables()` method for this. Output the list of values to the console.

Tip: Think of the appropriate imports!