9 Übungen zu den Neuerungen in den JDKs 10 und 11

Mit den folgenden Aufgaben sollen die Neuerungen aus JDK 10 und 11 anhand von Übungen in ihrer Handhabung vertieft werden.

✍ Aufgabe 1 – Experimentieren mit `var` ✍

Lernen Sie nachfolgend das neue reservierte Wort `var` mit seinen Möglichkeiten und Beschränkungen kennen.

Aufgabe 1a: Starten Sie die JShell oder eine IDE Ihrer Wahl. Erstellen Sie eine Methode `funWithVar()`. Definieren Sie dort die Variablen `name` und `age` mit den Werten Mike bzw. 47. Geben Sie anschließend deren Typ aus.

```
void funWithVar()
{
    // TODO
}
```

Tipp: Für die zweite Variable hilft Folgendes: `((Object)age).getClass()`.

Aufgabe 1b: Erweitern Sie Ihr Know-how bezüglich `var` und Generics. Nutzen Sie es für folgende Definition:

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

Erzeugen Sie initial zunächst eine lokale Variable `personsAndAges` und vereinfachen Sie dann mithilfe von `var`.

Aufgabe 1c: Vereinfachen Sie folgende Definitionen mit `var`. Was ist zu beachten? Worin liegt der Unterschied?

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

Aufgabe 1d: Wieso führen folgende Lambdas zu Problemen? Wie löst man diese?

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

Wieso kompiliert dann aber Folgendes?

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

✍ Aufgabe 2 – Unmodifiable Collections ✍

In Java 9 wurden Unmodifiable Collections und Collection Factory Methods eingeführt. Java 10 bietet das Erzeugen unveränderlicher Kopien und spezielle Kollektoren.

Aufgabe 2a: Welche zwei Varianten bietet Java 10, um eine unveränderliche Kopie folgender Liste zu erstellen?

```
List<String> words = List.of("Hello", "World");
List<String> copy1 = null; /* TODO */
List<String> copy2 = null; /* TODO */
```

Welchen Typ haben die entstehenden Kopien? Geben Sie diesen auf der Konsole aus.

Aufgabe 2b: Worin liegt der Unterschied innerhalb der entstehenden unmodifizierbaren Listen, wenn man eine Modifikation im Array vornimmt?

```
final String[] nameArray = { "Tim", "Tom", "Mike" };

// 3 Varianten von unmodifizierbaren Listen
final List<String> names1 = Arrays.asList(nameArray);
final List<String> names2 = List.of(nameArray);
final List<String> names3 = List.of("Tim", "Tom", "Mike");

// Modifikation im Array
nameArray[2] = "Michael";
```

Was geschieht beim Ändern in der Liste analog zu Folgendem?

```
names1.set(1, "XXX");
```

Aufgabe 2c: Was unterscheidet die drei Varianten von Kopien bzw. Wrapper?

```
List<String> names = List.of("Tim", "Tom", "Mike");

// 3 Varianten, um Kopien von unmodifizierbaren Listen zu erzeugen
List<String> copy1 = new ArrayList<>(names);
List<String> copy2 = List.copyOf(names);
List<String> wrapper = Collections.unmodifiableList(names);
```

Lassen sich Elemente ändern? Lassen sich Elemente löschen? Befüllen Sie nachfolgende Tabelle.

Tabelle 9-1

Variante		Löschen	Ändern
`copy1`			
`copy2`			
`wrapper`			

Tipp Nutzen Sie unter anderem folgende Hilfsmethode:

```
private static void tryModification(final String info, final List<String> list)
{
    try
    {
        list.set(1, "XXX");
        System.out.println(info + ": " + list);
    }
    catch (java.lang.UnsupportedOperationException ex)
    {
        System.out.println(info + ": set() not allowed");
    }
}
```

✍ Aufgabe 3 – Erweiterung in `Optional<T>` ✍

Gegeben sei eine simple, aber unsichere Wertextraktion aus einem `Optional<T>` wie folgt:

```
static <T> T badStyleExtractValue(final Optional<T> opt)
{
    T value = opt.get();
    return value;
}
```

Aufgabe 3a: Schreiben Sie diese mit dem neuen Java-10-API so um, dass der potenziell unsichere Zugriff offensichtlich wird.

Aufgabe 3b: Wie würde man die Methode oder Aufrufer dann umgestalten?

Aufgabe 4 – Erweiterungen in der Klasse `Reader`

Transferieren Sie den Inhalt aus einem `StringReader` in eine Datei `hello.txt`. Lesen Sie diese wieder ein und geben Sie den Inhalt aus. Ergänzen Sie dazu folgende Zeilen:

```
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
}
System.out.println(sw.toString());
```

Aufgabe 5 – Strings

Die Verarbeitung von Strings wurde in Java 11 mit einigen Methoden erleichtert.

Aufgabe 5a: Realisieren Sie eine Ausgabe, die fünf Zahlen untereinander ausgibt, jeweils so oft wiederholt, wie die Ziffer, also verkürzt wie folgt:

```
22
4444
7777777
333
999999999
```

Nutzen Sie folgenden Stream als Eingabe:

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

Aufgabe 5b: Modifizieren Sie die Ausgabe so, dass die Zahlen rechtsbündig mit maximal 10 Zeichen ausgegeben werden:

```
'      4444'
'   7777777'
' 999999999'
```

Implementieren und nutzen Sie dazu folgende Hilfsmethode:

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

Aufgabe 5c: Ändern Sie das Ganze so ab, dass nun statt Leerzeichen führende Nullen ausgegeben werden, etwa wie folgt:

```
'0000004444'
'0007777777'
'0999999999'
```

Aufgabe 5d: Modifizieren Sie die Ausgabe so, dass die größten Zahlen zuletzt ausgegeben werden. Finden Sie mindestens zwei Varianten für folgende Eingaben:

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

✍ Aufgabe 6 – Erweiterungen in `Optional<T>` in Java 11 ✍

Vereinfachen Sie folgendes Konstrukt mit der neuen Methode `isEmpty()` aus der Klasse `Optional<T>`:

```
static <T> Stream<T> asStream(final Optional<T> opt)
{
    if (!opt.isPresent())
    {
        return Stream.empty();
    }

    return Stream.of(opt.get());
}
```

✍ Aufgabe 7 – Erweiterung in `Predicate<T>` ✍

Vereinfachen Sie folgende Prädikate bezüglich der Negation:

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

Predicate<String> isBlank = String::isBlank;
var notIsBlank = isBlank.negate();
```

✍ Aufgabe 8 – Strings und Dateien ✍

Bis Java 11 war es etwas mühsam, Texte direkt in eine Datei zu schreiben bzw. daraus zu lesen. Dazu gibt es nun die Methoden `writeString()` und `readString()` aus der Klasse `Files`. Schreiben Sie mit deren Hilfe folgende Zeile in eine Datei:

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

Lesen Sie diese Zeilen wieder ein und bereiten Sie daraus eine `List<String>` auf.

✍ Aufgabe 9 – HTTP/2-API ✍

Gegeben sei folgende HTTP-Kommunikation, die auf die Webseite von Oracle zugreift und diese textuell aufbereitet. Wandeln Sie den Sourcecode so um, dass das neue HTTP/2-API aus JDK 11 zum Einsatz kommt.

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

    final String contents = readContent(url.openStream());
    System.out.println(contents);
}

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();
    }
}
```

Tipp Nutzen Sie die neuen Klassen `HttpRequest` und `HttpResponse` und erstellen Sie eine Methode `printResponseInfo(HttpResponse)`, die analog zu der obigen Methode `readContent(InputStream)` den Body ausliest und ausgibt. Zusätzlich soll noch der HTTP-Statuscode ausgegeben werden.

```
public static void main(final String[] args) throws Exception
{
    final URI uri = new URI("https://www.oracle.com/index.html");

    final HttpClient httpClient = HttpClient.newHttpClient();
    final HttpRequest request = // ...
    final BodyHandler<String> asString = // ...
    final HttpResponse<String> response = // ...

    printResponseInfo(response);
}

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

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

Bonus: Starten Sie die Abfragen asynchron und verarbeiten Sie das erhaltene `CompletableFuture<HttpResponse>` durch Aufruf von `sendAsync()` in etwa wie folgt:

```
final CompletableFuture<HttpResponse> asyncResponse =
                                      httpClient.sendAsync(request, asString);
asyncResponse.thenAccept(response -> printResponseInfo(response));

waitForCompletion();

if (asyncResponse.isDone())
{
    final HttpResponse<String> response = asyncResponse.get();
    printResponseInfo(response);
}
else
{
    asyncResponse.cancel(true);
    System.err.println("timeout");
}
```

✍ Aufgabe 10 – Direkte Kompilierung und Ausführung ✍

Schreiben Sie eine Klasse `HelloWorld` im Package `direct.compilation` und speichern Sie diese in einer gleichnamigen Java-Datei. Führen Sie diese direkt mit dem Kommando `java` aus.