4 Übungen zu den Syntaxneuerungen in JDK 11 bis 17

Mit den folgenden Aufgaben sollen die Neuerungen aus JDK 11 bis 17 anhand von Übungen in ihrer Handhabung vertieft werden. In Abschnitt 4.2 finden Sie die passenden Musterlösungen.

4.1 Aufgaben

✍ Aufgabe 1 – Syntaxänderungen bei `switch` ✍

Vereinfachen Sie einige herkömmliche `switch-case`-Konstrukte durch die neue Syntax.

Aufgabe 1a Die nachfolgende Methode soll durch die Angabe mehrerer Werte bei `case` und das Verwenden von Pfeilen kürzer und übersichtlicher werden:

```
private static void switchOld(final int number)
{
    switch (number)
    {
        case 0:
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
            System.out.println("Durchgefallen");
            break;
        case 6:
        case 7:
        case 8:
            System.out.println("Gut");
            break;
        case 9:
        case 10:
            System.out.println("Exzellent");
            break;
        default:
            System.out.println("Ungültig");
            break;
    }
}
```

Aufgabe 1b Verwenden Sie zur Vereinfachung folgender Methode die Möglichkeit, Rückgaben direkt zu spezifizieren:

```
private static int switchBonusOld(final char grade)
{
    int bonus;
    switch (grade)
    {
        case 'A':
            bonus = 2000;
            break;
        case 'B':
            bonus = 1000;
            break;
        case 'C':
            bonus = 500;
            break;
        default:
            bonus = 0;
            break;
    }

    return bonus;
}
```

✍ Aufgabe 2 – Schrittweises Umformen mit `switch` ✍

Der folgende Sourcecode prüft mit einem herkömmlichen `switch-case` auf gerade und ungerade Zahlen. Vereinfachen Sie ihn durch die neue 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:
        case 2:
        case 4:
        case 6:
        case 8:
        case 10:
            result = "even";
            break;

        default:
            result = "only implemented for values from 0 to 10";
    }

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

Aufgabe 2a Nutzen Sie zunächst nur die Angabe mehrerer Werte bei `case` und die Pfeilsyntax, um die Methode kürzer und übersichtlicher zu schreiben.

Aufgabe 2b Verwenden Sie im Anschluss die Möglichkeit, Rückgaben direkt zu spezifizieren, und ändern Sie die Signatur in `String dumbEvenOddChecker(int)`. Verlagern Sie die Ausgabe an die Aufrufstelle.

Aufgabe 2c Wandeln Sie das Ganze so ab, dass bei geraden Zahlen zwischen 2 und 8 (inklusive) noch die Ausgabe »Hurra, es ist gerade und im Bereich 2 - 8« erfolgt.

Tipp Dazu müssen Sie die Spezialform »yield mit Rückgabewert« verwenden.

Aufgabe 2d Während `yield` gerade für die Rückgabe in speziellen Fällen genutzt wurde, rekapitulieren Sie diese neue Syntax für die ursprüngliche Aufgabe.

✍ Aufgabe 3 – Fehlerfixing mit `switch` ✍

Vereinfachen Sie die Methode `detectKeyJava11(char)` mithilfe der neuen Syntax von `switch` und korrigieren Sie gleichzeitig mögliche Flüchtigkeitsfehler. Schreiben Sie dazu eine Methode `detectKeyJava17(char)`.

In einem nächsten Schritt schauen Sie bitte genauer hin und versuchen Sie das Ganze maximal zu vereinfachen.

```
static void detectKeyJava11(final char key)
{
    switch (key)
    {
        case '0':
            System.out.println("You pressed: 0");
            break;
        case '1':
            System.out.println("You pressed: 1");
            break;
        case '2':
            System.out.println("You pressed: 2");
        case '3':
            System.out.println("You pressed: 3");
        case '4':
            System.out.println("You pressed: 4");
            break;
        case '5':
        case '6':
        case '7':
        case '8':
        case '9':
            System.out.println("You pressed: " + key);
            break;
        default:
            System.out.println("Not allowed!");
    }
}
```

Überprüfen Sie Ihre Arbeiten mit den `char`-Werten für `2` und `7`. Erwartet werden folgende Ausgaben:

```
You pressed: 2
You pressed: 7
```

✍ Aufgabe 4 – Text Blocks ✍

Vereinfachen Sie den folgenden Sourcecode mit herkömmlichen Stringkonkatenationen, die über mehrere Zeilen gehen, und nutzen Sie dazu die neu eingeführte Syntax der Text Blocks.

```
String multiLineStringOld = "THIS IS\n" +
        "A MULTI\n" +
        "LINE STRING\n" +
        "WITH A BACKSLASH \\\n";
System.out.println(multiLineStringOld);

String multiLineHtmlOld = "<html>\n" +
        "    <body>\n" +
        "        <p>Hello, world</p>\n" +
        "    </body>\n" +
        "</html>";
System.out.println(multiLineHtmlOld);

String jsonTextBlockOld = "{\n" +
                          "    \"version\": \"Java 17\",\n" +
                          "    \"feature\": \"text blocks\",\n" +
                          "    \"attention\": \"very cool!\"\n" +
                          "}\n";
System.out.println(jsonTextBlockOld);
```

Bonus Betreiben Sie etwas ASCII-Art und zeichnen Sie einen Tee- oder Kaffeebecher oder ein Glas Bier mithilfe von Text Blocks.

✍ Aufgabe 5 – Text Blocks mit Platzhaltern ✍

Vereinfachen Sie den folgenden Sourcecode mit einem herkömmlichen String, der über mehrere Zeilen geht, und nutzen Sie die neu eingeführte Syntax inklusive der Möglichkeit, Platzhalter anzugeben und mit Werten zu ersetzen:

```
String multiLineStringWithPlaceholdersOld =
       String.format("HELLO \"%s\"!\n" +
                     "  HAVE %s\n" +
                     "  NICE \"%s\"!",
                     "WORLD", "A", "DAY");
System.out.println(multiLineStringWithPlaceholdersOld);
```

Produzieren Sie folgende Ausgaben mit der neuen Syntax:

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

✍ Aufgabe 6 – Record-Grundlagen ✍

Gegeben seien zwei einfache Klassen, die reine Datencontainer darstellen und somit lediglich ein öffentliches Attribut bereitstellen. Wandeln Sie diese in Records um:

```
public class Square
{
    public final double sideLength;

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

public class Circle
{
    public final double radius;

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

Welche Vorteile ergeben sich – außer der kürzeren Schreibweise – durch den Einsatz von Records statt eigener Klassen?

✍ Aufgabe 7 – Records ✍

Ergänzen Sie im nachfolgend gezeigten Record eine Prüfung der beiden Namensbestandteile sowie des Geburtstags, sodass die Namen jeweils länger als zwei Zeichen sind und das Datum in der Vergangenheit liegt. Erstellen Sie zusätzlich zwei Methoden, die eine JSON- und eine XML-Ausgabe erzeugen – oftmals sollte eine derartige Transformation nicht vom Objekt selbst implementiert werden, hier als Übung aber schon:

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

Wir konstruieren ein Objekt und geben dieses als XML und JSON aus:

```
Person mike = new Person("Michael", "Inden", LocalDate.of(1971, 2, 7));
System.out.println(mike.toXml());
System.out.println(mike.toJSON());
```

Folgendes sollte auf der Konsole erscheinen:

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

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

✍ Aufgabe 8 – `instanceof`-Grundlagen ✍

Gegeben seien folgende Zeilen mit einem `instanceof` sowie einem Cast:

```
Object obj = "BITTE MICHAEL";

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

Vereinfachen Sie das Ganze mit den Neuerungen aus Java 17.

✍ Aufgabe 9 – `instanceof` und OO-Design ✍

Vereinfachen Sie den Sourcecode mithilfe der Syntaxneuerungen bei `instanceof` und danach mithilfe der Besonderheiten bei Records.

Aufgabe 9a Gegeben seien die folgenden beiden Records:

```
record Square(double sideLength) {};
record Circle(double radius) {};
```

Für diese wird deren Fläche nicht besonders elegant (später mehr dazu) berechnet:

```
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");
}
```

Vereinfachen Sie diese Methode durch den Einsatz der neuen Syntax von `instanceof`.

Aufgabe 9b Zwar haben wir durch `instanceof` sicher eine Verbesserung bezüglich Lesbarkeit und Anzahl Zeilen erzielt, jedoch deuten mehrere derartige Prüfungen auf einen Verstoß gegen das Open Closed Principle, eines der SOLID-Prinzipien guten Entwurfs, hin. Was wäre ein objektorientiertes Design? Die Antwort ist in diesem Fall einfach: Oftmals lassen sich `instanceof`-Prüfungen vermeiden, indem ein Basistyp eingeführt wird. Vereinfachen Sie das Ganze durch ein Interface `BaseFigure` und nutzen Sie dieses passend.