

Übungen zu Einführung in die Informatik: Programmierung und Software-Entwicklung:

Lösungsvorschlag

Aufgabe 4-1

Gültigkeitsbereiche und Speicher

Präsenz

Gegeben sei folgender Java-Codeblock:

```

1 {
2   int m = 13, n = 2, z = 0;
3   m = m + n;
4   {
5     double x;
6     x = m / n;
7     int i = (int) x;
8     z = i - n;
9   }
10  n++;
11  int h = n + 1;
12 }
```

- a) Bestimmen Sie den Gültigkeitsbereich von `m`, `n`, `z`, `x`, `i` und `h`.

Lösung:

```

{
  int m = 13, n = 2, z = 0;
  m = m + n;
  {
    double x;
    x = m / n;
    int i = (int) x;
    z = i - n;
  }
  n++;
  int h = n + 1;
}
```

$\left. \begin{array}{l} \text{double } x; \\ x = m / n; \\ \text{int } i = (\text{int}) x; \\ z = i - n; \end{array} \right\} \text{ Gültigkeitsbereich von } x \text{ und } i$

$\left. \begin{array}{l} \text{int } m = 13, n = 2, z = 0; \\ m = m + n; \\ \text{int } h = n + 1; \end{array} \right\} \text{ Gültigkeitsbereich von } m, n, z \text{ und } h$

- b) Zeigen Sie die Veränderung des Speichers während der Ausführung des gesamten Codeblocks.

Lösung:

| | | | | | | | | | | | |
|---|----|---|----|---|----|---|-----|---|-----|---|----|
| | | | | i | 7 | i | 7 | | | | |
| | | | | x | | x | 7.0 | x | 7.0 | h | 4 |
| z | 0 | z | 0 | z | 0 | z | 0 | z | 5 | z | 5 |
| n | 2 | n | 2 | n | 2 | n | 2 | n | 3 | n | 3 |
| m | 13 | m | 15 | m | 15 | m | 15 | m | 15 | m | 15 |

Gegeben sei folgender Java-Codeblock:

```

1 {
2   double a = 2.0, b = 3.4;
3   {
4     a = a + b;
5     int c = (int) a;
6     a = c;
7   }
8   a--;
9   double d = a + b;
10  boolean test = a != d;
11 }

```

- a) Bestimmen Sie den Gültigkeitsbereich von a, b, c, d und test.

Lösung:

```

{
  double a = 2.0, b = 3.4;
  {
    a = a + b;
    int c = (int) a;
    a = c;
  }
  a--;
  double d = a + b;
  boolean test = a != d;
}

```

} Gültigkeitsbereich von c

} Gültigkeitsbereich von a, b, d und test

- b) Zeigen Sie die Veränderung des Speichers während der Ausführung des gesamten Codeblocks.

Lösung:

| | | | | | | | |
|---|---|---|---|---|---|------|--|
| | | | | | | test | <div style="border: 1px solid black; padding: 2px;">true</div> |
| | | c | <div style="border: 1px solid black; padding: 2px;">5</div> | c | <div style="border: 1px solid black; padding: 2px;">5</div> | d | <div style="border: 1px solid black; padding: 2px;">7.4</div> |
| b | <div style="border: 1px solid black; padding: 2px;">3.4</div> | b | <div style="border: 1px solid black; padding: 2px;">3.4</div> | b | <div style="border: 1px solid black; padding: 2px;">3.4</div> | b | <div style="border: 1px solid black; padding: 2px;">3.4</div> |
| a | <div style="border: 1px solid black; padding: 2px;">2.0</div> | a | <div style="border: 1px solid black; padding: 2px;">5.4</div> | a | <div style="border: 1px solid black; padding: 2px;">5.0</div> | a | <div style="border: 1px solid black; padding: 2px;">4.0</div> |

Aufgabe 4-3

Einfache Anweisungen in Java

Präsenz

Ein KFZ-Besitzer möchte ein Java-Programm erstellen, das ihm vor jeder Fahrt den entstehenden Benzin- und Ölverbrauch berechnet. Der Benzinverbrauch des Fahrzeugs beträgt 6.7 Liter pro 100 km, der Ölverbrauch beträgt 0.6 Liter pro 1000 km. Schreiben Sie in einer Klasse **Verbrauch** ein Java-Programm, das für eine gegebene Fahrtstrecke den entstehenden Benzin- und Ölverbrauch berechnet und ausgibt. Testen Sie Ihr Programm mit den Fahrtstrecken 0.1 km, 3 km und 100.13 km. Bestimmen Sie zunächst, welche lokalen Variablen Sie benötigen (mit Typ)!

Mögliche Lösung: Wir benötigen folgende lokalen Variablen:

- eine Variable `benzinverbrauchPro100km` vom Typ `double`
- eine Variable `oelverbrauchPro1000km` vom Typ `double`

- eine Variable `fahrtstrecke` vom Typ `double`
- eine Variable `heutigerBenzinverbrauch` vom Typ `double`, in der wir mit Hilfe der Variablen `benzinverbrauchPro100km` und `fahrtstrecke` den Benzinverbrauch für diese Fahrtstrecke berechnen
- eine Variable `heutigerOelverbrauch` vom Typ `double`, in der wir mit Hilfe der Variablen `oelverbrauchPro1000km` und `fahrtstrecke` den Benzinverbrauch für diese Fahrtstrecke berechnen

```

1 public class Verbrauch {
2     public static void main(String[] args) {
3         double benzinverbrauchPro100km = 6.7;
4         double oelverbrauchPro1000km = 0.6;
5
6         // Berechnung des heutigen Verbrauchs
7         double fahrtstrecke = 100.13;
8         double heutigerBenzinverbrauch =
9             fahrtstrecke * benzinverbrauchPro100km / 100;
10        double heutigerOelverbrauch =
11            fahrtstrecke * oelverbrauchPro1000km / 1000;
12
13        System.out.println("Der heutige Benzinverbrauch ist "
14            + heutigerBenzinverbrauch);
15        System.out.println("Der heutige Oelverbrauch ist "
16            + heutigerOelverbrauch);
17    }
18 }

```

Aufgabe 4-4

Einfache Anweisungen in Java

Hausaufgabe

Ein Pizza-Service berechnet den Preis einer Pizza nach folgendem Schema: Der Grundpreis einer Pizza ist 5.50 Euro und jeder Belag, den ein Kunde auswählt, kostet weitere 0.75 Euro. Schreiben Sie in einer Klasse `PizzaService` ein Java-Programm, das für eine gegebene Anzahl an Belagsorten den Gesamtpreis der Pizza berechnet und ausgibt. Testen Sie Ihr Programm mit 1, 5 und 23 Belagsorten. Bestimmen Sie zunächst, welche lokalen Variablen Sie benötigen (mit Typ)!

Mögliche Lösung: Wir benötigen folgende lokalen Variablen:

- eine Variable `grundpreis` vom Typ `double`
- eine Variable `belagpreis` vom Typ `double`
- eine Variable `anzahlBelagsorten` vom Typ `int`
- eine Variable `gesamtpreis` vom Typ `double`, in der wir mit Hilfe der obigen Variablen `grundpreis`, `belagpreis` und `anzahlBelagsorten` den Gesamtpreis der entsprechenden Pizza berechnen

```

1 public class PizzaService {
2     public static void main(String[] args) {
3         double grundpreis = 5.5;
4         double belagpreis = 0.75;
5
6         // Berechnung des Gesamtpreises
7         int anzahlBelagsorten = 23;
8         double gesamtpreis = grundpreis + anzahlBelagsorten * belagpreis;
9     }

```

```
10     System.out.println("Die Pizza kostet " + gesamtpreis + " Euro.");
11 }
12 }
```

Aufgabe 4-5

Bedingungen in Java

Präsenz

Schreiben Sie in einer Klasse `IntSortierung` ein Javaprogramm, in dem zunächst drei Variablen `x`, `y` und `z` vom Typ `int` deklariert werden. Das Javaprogramm soll die Werte in den Variablen aufsteigend sortieren und auf dem Bildschirm ausgeben. Testen Sie Ihr Programm für die Werte `x = 5`, `y = 3` und `z = 1`.

Lösungsidee:

Wir definieren drei Variablen `x`, `y` und `z` und initialisieren sie hier beispielsweise mit `x = 5`, `y = 3` und `z = 1`.

- Zunächst vergleichen wir `x` und `y`. Falls `x > y` gilt, vertauschen wir `x` und `y` (sonst nicht). (Hier: Da `5 > 3` gilt, vertauschen wir die Werte und erhalten `x = 3` und `y = 5`.)
- Anschließend vergleichen wir `y` und `z`. Falls `y > z` gilt, vertauschen wir `y` und `z` (sonst nicht). (Hier: Da `5 > 1` gilt, vertauschen wir die Werte und erhalten `y = 1` und `z = 5`.)
- Zuletzt müssen wir nochmal `x` und `y` vergleichen. Falls `x > y` gilt, vertauschen wir nochmals `x` und `y` (sonst nicht). (Hier: Da `3 > 1` gilt, vertauschen wir die Werte und erhalten `x = 1` und `y = 3`.)

Diesen Algorithmus setzt man folgendermaßen in Java um:

```
1 public class IntSortierung {
2     public static void main(String[] args) {
3         int x = 5;
4         int y = 3;
5         int z = 1;
6
7         if (x > y) {
8             int tmp = x;
9             x = y;
10            y = tmp;
11        }
12        if (y > z) {
13            int tmp = y;
14            y = z;
15            z = tmp;
16        }
17        if (x > y) {
18            int tmp = x;
19            x = y;
20            y = tmp;
21        }
22
23        System.out.println("Die richtige Reihenfolge ist " +
24            x + "," + y + "," + z);
25    }
26 }
```

In den USA wird für das Jahr 2016 auf Basis folgender Tabelle die Einkommensteuer berechnet; in der Tabelle sind die Bruttogehälter (d.h. vor Steuerabzug) angegeben.

| Steuersatz | Ledige | Verheiratete |
|------------|-----------------------|-----------------------|
| 10% | \$0 - \$9 275 | \$0 - \$18 550 |
| 15% | \$9 276 - \$37 650 | \$18 551 - \$75 300 |
| 25% | \$37 651 - \$91 150 | \$75 301 - \$151 900 |
| 28% | \$91 151 - \$190 150 | \$151 901 - \$231 450 |
| 33% | \$190 151 - \$413 350 | \$231 451 - \$413 350 |
| 35% | \$413 351 - \$415 050 | \$413 351 - \$466 950 |
| 39.6% | ab \$415 051 | ab 466 951 |

Schreiben Sie in einer Klasse **SteuernUSA** ein Javaprogramm, das das Netto-Gehalt (d.h. nach Steuerabzug) in den USA berechnet und ausgibt. Bestimmen Sie zunächst, welche lokalen Variablen Sie benötigen (mit Typ)! Ob eine Person verheiratet ist oder nicht, kann mit einer Variable vom Typ **boolean** ausgedrückt werden. Testen Sie Ihr Programm für eine verheiratete Person mit einem Brutto-Gehalt von \$123 456.78.

Mögliche Lösung: Wir benötigen vier lokale Variablen:

- eine Variable **gehaltBrutto** vom Typ **double**
- eine Variable **verheiratet** vom Typ **boolean**
- eine Variable **steuern** vom Typ **double**, in der wir mit Hilfe der Variablen **gehaltBrutto** und **verheiratet** die zu zahlenden Steuern berechnen
- eine Variable **gehaltNetto** vom Typ **double**, in der wir mit Hilfe der Variablen **steuern** und **gehaltBrutto** das Netto-Gehalt nach Abzug der Steuern berechnen

```

1 public class SteuernUSA {
2     public static void main(String[] args) {
3         double gehaltBrutto = 123456.78;
4         boolean verheiratet = true;
5
6         double steuern;
7         if (verheiratet) {
8             if (gehaltBrutto <= 18550) {
9                 steuern = 0.1 * gehaltBrutto;
10            }
11            else if (gehaltBrutto <= 75300) {
12                steuern = 0.15 * gehaltBrutto;
13            }
14            else if (gehaltBrutto <= 151900) {
15                steuern = 0.25 * gehaltBrutto;
16            }
17            else if (gehaltBrutto <= 231450) {
18                steuern = 0.28 * gehaltBrutto;
19            }
20            else if (gehaltBrutto <= 413350) {
21                steuern = 0.33 * gehaltBrutto;
22            }
23            else if (gehaltBrutto <= 466950) {
24                steuern = 0.35 * gehaltBrutto;
25            }
26            else {
27                steuern = 0.396 * gehaltBrutto;
28            }

```

```

29     }
30     else {
31         if (gehaltBrutto <= 9275) {
32             steuern = 0.1 * gehaltBrutto;
33         }
34         else if (gehaltBrutto <= 37650) {
35             steuern = 0.15 * gehaltBrutto;
36         }
37         else if (gehaltBrutto <= 91150) {
38             steuern = 0.25 * gehaltBrutto;
39         }
40         else if (gehaltBrutto <= 190150) {
41             steuern = 0.28 * gehaltBrutto;
42         }
43         else if (gehaltBrutto <= 413350) {
44             steuern = 0.33 * gehaltBrutto;
45         }
46         else if (gehaltBrutto <= 415050){
47             steuern = 0.35 * gehaltBrutto;
48         }
49         else {
50             steuern = 0.369 * gehaltBrutto;
51         }
52     }
53
54     double gehaltNetto = gehaltBrutto - steuern;
55
56     System.out.println("Ihr Jahresgehalt nach Steuern ist "
57         + gehaltNetto);
58 }
59 }

```

Besprechung der Präsenzaufgaben in den Übungen ab 09.11.2018. Abgabe der Hausaufgaben bis Mittwoch, 21.11.2018, 14:00 Uhr über UniworX (siehe Folien der ersten Zentralübung).

- Erstellen Sie zu jeder Aufgabe eine Klasse, die den Namen trägt, der in der Aufgabe gefordert ist.
- Der Java-Code in ihrer Abgabe muss als separate *.java*-Datei abgegeben werden. Wir benötigen **nicht** Ihre *.class*-Dateien. Öffnen Sie dazu im Explorer den Ordner, den Sie als Eclipse-Workspace ausgewählt haben und navigieren Sie in den entsprechenden Unterordner anhand ihrer Projektstruktur.
- Kopieren Sie alle Dateien Ihrer Abgabe in einen eigenen Ordner (für jedes Übungsblatt ein eigener Ordner) und geben Sie den gesamten Ordner als ZIP-Archiv ab. In dem Ordner dürfen nur Dateien mit der Endung *.java*, *.pdf*, *.jpg* oder *.txt* enthalten sein. **Word-Dokumente werden nicht korrigiert!**
- Unter Windows kann ein ZIP-Archiv wie folgt erstellt werden: rechter Mausklick auf den Ordner, Auswahl von *Senden an* -> *ZIP-komprimierter Ordner*. Unter Mac OS hängen: rechter Mausklick auf den Ordner, Auswahl von *Komprimieren/Compress*.