technische universität
dortmund

Fakultät für Informatik
Lehrstuhl 14
Lars Hildebrand

Praktikum zu

Einführung in die Informatik für LogWings, WiMas und MedPhys

Wintersemester 2020/21

Übungsblatt 6

Besprechung:
4.–8.1.2021 (KW 1)

Vorbereitende Aufgaben

Aufgabe 6.1: Codeformatierung

Für Programmierer ist die richtige Formatierung von Programmcode sehr wichtig. Lesbarer Programmcode erleichtert die Fehlersuche und macht Code leichter erweiterbar. Dafür gibt es in Java Programmier-Konventionen[1]. Formatieren Sie den folgenden Code, sodass er leichter zu lesen ist.

a)
```
if(a==7){System.out.println("a ist sieben");}
```

Lösung

```
if (a == 7) {
    System.out.println("a ist sieben");
}
```

b)
```
int b=12;for(int i=2;i<12;i++
){b=b*i;}System.out.println(b);
```

Lösung

```
int b = 12;
for (int i = 2; i < 12; i++) {
    b = b * i;
}
System.out.println(b);
```

[1] https://www.oracle.com/technetwork/java/codeconventions-150003.pdf

c) `int a=3;if(a<5){int p=7;while(p>0){System.out.println(p);p--;}}`

Lösung

```
int a = 3;
if (a < 5) {
    int p = 7;
    while (p > 0) {
        System.out.println(p);
        p--;
    }
}
```

d) `for(int i=0;i<23;i++){if(i%2==0){for (int j=1;j<i;j++){System.out.println( i+";"+j);}}else{System.out.println(i);}}`

Lösung

```
for (int i = 0; i < 23; i++) {
    if (i % 2 == 0) {
        for (int j = 1; j < i; j++) {
            System.out.println(i + ";" + j);
        }
    } else {
        System.out.println(i);
    }
}
```

Präsenzaufgaben

Aufgabe 6.2: Umwandlung von Schleifen

In dieser Aufgabe wollen wir Schleifen von einem Typ in einen anderen umwandeln.

a) Wandeln Sie die folgende **for**-Schleife in eine **while**-Schleife um.

```
int a = 0;
for (int b = 0; b < 17; b++) {
    a = a + 2;
}
System.out.println("a: " + a);
```

Lösung

```
int b = 0;
int a = 0;
while (b < 17) {
    a = a + 2;
    b++;
}
System.out.println("a: " + a);
```

b) Wandeln Sie die folgende **do-while**-Schleife in eine **while**-Schleife um.

```
int o = 7;
int p = 256;
do {
    p = p - 20;
    o--;
} while (o > 0);
System.out.println("p :" + p);
```

Lösung

```
int o = 7;
int p = 256;
p = p - 20;
o--;
while (o > 0) {
    p = p - 20;
    o--;
}
System.out.println("p :" + p);
```

c) Wandeln Sie die folgende **while**-Schleife in eine **for**-Schleife um.

```
int b = 0;
int n = 7;
int e = 1;
while ((b < 15) && (n < 9)) {
    e = e + b * n - 2;
    b = b + 2;
}
System.out.println("e: " + e);
```

Lösung

```
int e = 1;
int n = 7;
for (int b = 0; (b < 15) && (n < 9); b = b + 2) {
    e = e + b * n - 2;
}
System.out.println("e: " + e);
```

Aufgabe 6.3: Funktionsköpfe

Nun wollen wir den Umgang mit Funktionen üben. Schreiben Sie Funktionsköpfe für die folgenden Funktionen. Die Funktionsrümpfe oder eine konkrete Implementierung der Funktionen werden dabei nicht benötigt.

a) die Funktion `addFive` soll eine int-Variable `x` mit 5 addieren und die Summe zurückgeben

`public static int addFive(int x)`

b) die Funktion `mult` soll zwei Zahlen miteinander multiplizieren und das Produkt zurückgeben

`public static int mult(int x, int y)`

c) eine Funktion, die eine Potenz berechnet und das Ergebnis zurückgibt

`public static long pow(int n, int m)`

d) eine Funktion, die die Summe dreier Zahlen ausgibt

`public static void printSum(int x, int y, int z)`

e) eine Funktion, die das erste Zeichen eines Wortes zurückgibt

`public static char firstChar(String word)`

f) eine Funktion, die die Gleichheit zweier Zahlen überprüft und das Ergebnis zurückgibt

`public static boolean equals(int x, int y)`

g) eine Funktion, die das Maximum zweier Zahlen zurückgibt

`public static int max(int x, int y)`

h) eine Funktion, die das Maximum zweier Zahlen ausgibt

`public static void printMax(int x, int y)`

i) eine Funktion, die eine Dezimalzahl als Binärzahl ausgibt

`public static void DezToBin(int x)`

Aufgabe 6.4: Funktionen

In dieser Aufgabe wollen wir uns mit der Deklaration und dem Aufruf von Funktionen vertraut machen. Bei der Konstruktion eines Geschwindigkeitsmessers für Automobile soll ein Programm die derzeitige Geschwindigkeit in Kilometern pro Stunde ($\frac{km}{h}$) ausgeben. Die Sensoren geben allerdings die zurückgelegte Strecke nur in Metern und die dabei vergangene Zeit in Sekunden an.

a) Da das Umrechnen von Geschwindigkeiten eine sehr allgemeine Aufgabe ist, die an vielen Stellen nützlich sein kann, wollen wir eine Funktion schreiben, die diese Umrechnung für uns übernimmt. Überlegen Sie sich, wie man eine Geschwindigkeit von $\frac{m}{s}$ in $\frac{km}{h}$ umrechnet.

$$v\left[\frac{m}{s}\right] = v \cdot \frac{\frac{1}{1000}}{\frac{1}{60 \cdot 60}}\left[\frac{km}{h}\right] = v \cdot 3.6\left[\frac{km}{h}\right]$$

b) Legen Sie eine neue Klasse **SpeedSensor** an. Fügen Sie die **main**-Methode ein, in der Sie zwei Variablen **currentMeters** und **currentSeconds** jeweils vom Typ **double** anlegen. Diese sollen die Sensorwerte repräsentieren. Denken Sie sich beliebige Werte dafür aus.

Legen Sie innerhalb der SpeedSensor-Klasse eine statische Funktion mit dem Namen `velocity` an. Diese soll zwei Parameter **meters** und **seconds** vom Typ **double** besitzen. Bestimmen Sie anschließend die *Geschwindigkeit* in $\frac{km}{h}$ und geben Sie diese mit Hilfe des **return**-Statements zurück. Geben Sie in der **main**-Methode mit folgendem Code die derzeitige Geschwindigkeit aus:

```
System.out.println("Current speed is: " +
            velocity(currentMeters, currentSeconds) + " km/h");
```

Lösung

```
1  public class SpeedSensor {
2      public static double velocity(double meters, double seconds) {
3          double kilometers = meters / 1000.0;
4          double hours = seconds / 3600.0;
5          return kilometers / hours;
6      }
7
8      // Alternativ
9      public static double velocity2(double meters, double seconds) {
10         return meters * 3.6 / seconds;
11     }
12
13     public static void main(String[] args) {
14         double currentMeters = 15.0;
15         double currentSeconds = 5.0;
16
17         System.out.println("Current speed is: " +
18             velocity(currentMeters, currentSeconds) + " km/h");
```

```
19      }
20  }
```

c) Jetzt wollen wir dem Fahrer auch noch den Bremsweg anzeigen. Sei v die aktuelle Geschwindigkeit in km/h, dann ist eine Näherung für den Bremsweg s in Metern:

$$s = \frac{1}{2}\left(\frac{v}{10}\right)^2$$

Schreiben Sie eine Funktion **brakingDistance** vom Typ **double**, die die Sensorwerte **meters** und **seconds** als Parameter nimmt und den Bremsweg in Metern zurückgibt. Geben Sie abschließend den berechneten Bremsweg analog wie in der letzten Teilaufgabe in der main-Funktion aus.

Lösung

```
public static double brakingDistance(double meters, double seconds) {
    double v = velocity(meters, seconds);
    return 0.5 * (v/10) * (v/10);
}
```

Komplettlösung

```
1  public class SpeedSensor {
2      public static double velocity(double meters, double seconds) {
3          double kilometers = meters / 1000.0;
4          double hours = seconds / 3600.0;
5          return kilometers / hours;
6      }
7
8      // Alternativ
9      public static double velocity2(double meters, double seconds) {
10         return meters * 3.6 / seconds;
11     }
12
13
14     // Teilaufgabe c)
15     public static double breakingDistance(double meters, double seconds) {
16         double v = velocity(meters, seconds);
17         return 0.5 * v/10.0 * v/10.0;
18     }
19
20
21     public static void main(String[] args) {
22         double currentMeters = 15.0;
23         double currentSeconds = 5.0;
24
25         System.out.println("Current speed is: " +
26             velocity(currentMeters, currentSeconds) + " km/h");
27     }
28 }
```

Aufgabe 6.5: Aufrufverhalten

In dieser Aufgabe wollen wir uns mit dem Aufruf von statischen Funktionen mit primitiven Datentypen als Parameter vertraut machen.

a) Betrachten Sie folgendes Programm:

```
1  public class Program {
2      public static void add5ToInt(int x) {
3          x = x + 5;
4          System.out.println("x in function add5ToInt: " + x);
5      }
6
7      public static void main(String[] args) {
8          int x = 8;
9
10         System.out.println("x before function call: " + x);
11         add5ToInt(x);
12         System.out.println("x after function call: " + x);
13     }
14 }
```

Quiz

Welchen Wert besitzt *x* bei der ersten Ausgabe (`x before function call`) ?

a) 8 ✓ b) 13

Welchen Wert besitzt *x* bei der zweiten Ausgabe (`x in function add5ToInt call`) ?

a) 8 b) 13 ✓

Welchen Wert besitzt *x* bei der dritten Ausgabe (`x after function call`) ?

a) 8 ✓ b) 13

b) Was fällt Ihnen auf? Was ist der Grund für dieses Programmverhalten?

Der Inhalt der Variable x ändert sich nicht nach dem Aufruf von add5ToInt

Es wird nur ein Call by Value durchgeführt, x ist in der Funktion lokal

Ergänzende Aufgaben

Aufgabe 6.6: Programmstrukturierung

In dieser Aufgabe wollen wir uns mit der Möglichkeit befassen, ein Programm durch Umstrukturierung in Funktionen verständlicher zu machen. Daher wollen wir ein Programm schreiben, das entscheidet, welcher von drei Quadern das größte Volumen besitzt.

a) Markieren Sie zuerst, in welchem Teil des Programmes sich komplexe wiederholende Strukturen befinden.

b) Lagern Sie diese Strukturen in Funktionen aus, indem Sie das Programm umschreiben. Legen Sie dazu eine neue Klasse mit dem Namen **BiggestVolume** an.

```
1  public class BiggestVolume {
2      public static void main(String[] args) {
3          double height1 = 16.5, width1 = 27.5, depth1 = 38.0;
4          double height2 = 20.0, width2 = 20.0, depth2 = 20.0;
5          double height3 = 40.2, width3 = 22.5, depth3 = 18.5;
6
7          if ((height1 * width1 * depth1 >=
8                  height2 * width2 * depth2) &&
9                  (height1 * width1 * depth1 >=
10                 height3 * width3* depth3)) {
11             System.out.println("Volume number 1 is the biggest");
12         }
13         else if ((height2 * width2 * depth2 >=
14                 height1 * width1 * depth1) &&
15                 (height2 * width2 * depth2 >=
16                 height3 * width3 * depth3)) {
17             System.out.println("Volume number 2 is the biggest");
18         }
19         else {
20             System.out.println("Volume number 3 is the biggest");
21         }
22     }
23 }
```

Lösung

Die Volumenberechnung und die Berechnung des Maximums können in Funktionen ausgelagert werden. Dies kann natürlich auf verschiedene andere Arten gemacht werden; dies hier ist nur ein Vorschlag!

```
1  public class BiggestVolumeSolution {
2      public static double volume(double height, double width, double depth) {
3          return height * width * depth;
4      }
5
6      public static int maxOf3(double first, double second, double third) {
7          if (first >= second && first >= third) {
8              return 1;
9          } else if (second >= first && second >= third) {
10             return 2;
11         } else {
12             return 3;
13         }
14     }
15
16     public static void main(String[] args) {
17         double volume1 = volume(16.5, 27.5, 38.0);
18         double volume2 = volume(20.0, 20.0, 20.0);
19         double volume3 = volume(40.2, 22.5, 18.5);
20
21         System.out.println("Volume number " +
```

```
22              maxOf3(volume1, volume2, volume3) + " is the biggest");
23      }
24  }
```