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 4

Besprechung:
7.–12.12.2020
(KW 50)

Vorbereitende Aufgaben

Aufgabe 4.1: Variablen – Wiederholung

Zu Beginn wollen wir den Umgang mit Variablen wiederholen.

a) Wie deklarieren Sie eine Variable vom Typ **double** mit dem Namen **velocity**?

double velocity;

b) Wie weisen Sie einer bereits deklarierten Variable mit dem Namen **value** den Wert 25 zu?

value = 25;

c) Wie deklarieren Sie eine Variable vom Typ **int** mit dem Namen **sum** und initialisieren diese mit der Summe zweier bereits deklarierten und initialisierten Variablen **x** und **y**?

int sum = x + y;

Präsenzaufgaben

Aufgabe 4.2: Modulorechnung

In dieser Aufgabe wollen wir ein Verständnis für die Ganzzahldivision, Modulorechnung und für große Zahlen entwickeln. Unser Ziel wird es sein, ein Programm zu schreiben, welches eine Uhrzeit ausgeben kann. Doch zunächst beschäftigen wir uns mit Ganzzahldivision und Modulorechnung.

Quiz

i) Was ergibt 90/60?
   a) 1 ✓ b) 1,5 c) 2

ii) Was ergibt 90%60?
   a) 90 b) 30 ✓ c) 1,5

iii) Welche Werte kann das Ergebnis von %60 annehmen?
   a) $[1, 60]$ b) $[0, \infty)$ c) $[0, 59]$ ✓

Aufgabe 4.3: Zeitumrechnung – Vorbereitung

Nun wenden wir diese Rechenarten an, um eine Uhr anhand eines gegebenen Zeitpunktes zu programmieren. In vielen Rechnersystemen werden Zeitpunkte in vergangenen Sekunden seit dem 1.1.1970 angegeben. Diese Zeitpunkte werden **Zeitstempel** genannt. Anhand solcher Zeitstempel können wir also Uhrzeiten berechnen.

Beispiel: Der Zeitstempel 42 repräsentiert den 1.1.1970 00:00:42 Uhr, der Zeitstempel $86.400 = 24 \cdot 60 \cdot 60$ den 2.1.1970 00:00:00 Uhr und der Zeitstempel 554.992.367 den 3.8.1987 12:32:47 Uhr.

Lösen Sie nun folgenden Aufgaben:

a) Wie viele ganze Minuten und Sekunden sind zum Zeitstempel 255 vergangen?

$255/60 = 4$ **Minuten und** $255\%60 = 15$ **Sekunden**

b) Wie viele ganze Stunden und Minuten sind zum Zeitstempel 9.930 vergangen?

$9930/60/60 = 2$ **Stunden und** $9930/60\%60 = 45$ **Minuten**

c) Wie viele ganze Tage und Stunden sind seit dem Zeitstempel 130.545 vergangen?

$130.545/60/60/24 = 1$ **Tag und** $130.545/60/60\%24 = 12$ **Stunden**

d) Wie viel Sekunden sind seit der letzten vollen Stunde zum Zeitstempel 106.820 vergangen?

$106.820\%(60 \cdot 60) = 2420$ **Sekunden**

e) Wie viele Sekunden seit Tagesbeginn sind zum Zeitstempel 1.030.637.317 vergangen?

$1.030.637.317\%(60 \cdot 60 \cdot 24) = 58.117$

f) Wie sehe die Darstellung des Zeitstempels 221.820 in einer klassischen Digitaluhr (hh:mm:ss) aus?

$13 : 37 : 00$

$221.820/60/60\%24 = 13$**;** $221.820/60\%60 = 37$**;** $221.820\%60 = 0$

Aufgabe 4.4: Zeitumrechnung – Implementierung

Wir werden nun die in Aufgabe 4.3 gewonnenen Erkenntnisse nutzen, um ein Programm zur Umrechnung der Uhrzeit zu schreiben.

a) Legen Sie eine neue Klassendatei mit dem Namen **Clock** an. Passen Sie die Datei so an, dass sie diesem Grundgerüst entspricht:

```
1 public class Clock {
2
3     public static void main(String[] args) {
4         /* Anfang des Programmcodes */
5
6         /* Ende des Programmcodes */
7     }
8 }
```

Das Programm soll nun innerhalb des durch Kommentare markierten Bereiches implementiert werden.

b) Um eine Uhrzeit zu repräsentieren, benötigen wir drei Werte: Stunden, Minuten und Sekunden. Verwenden Sie hierfür drei Variablen mit den Namen **seconds**, **minutes** und **hours**. Zusätzlich möchten wir den Zeitstempel, also die Anzahl der verstrichenen Sekunden, in einer Variable mit dem Namen **time** speichern. Deklarieren Sie deshalb insgesamt vier **long** Variablen, in denen diese Werte gespeichert werden sollen.

Lösung

```
1  public class Clock {
2
3      public static void main(String[] args) {
4          long time;
5          long seconds;
6          long minutes;
7          long hours;
8      }
9  }
```

Wenn Sie die Variable **time** als **int** deklarieren würden, welches Problem kann in der Zukunft auftreten?

Ein Zeitstempel ist eine sehr große Zahl.

So groß sogar, dass sie kaum noch in einen 32-Bit Integer passt.

Am 19. Januar 2038 wird der 32-Bit-Wert überlaufen.

c) Initialisieren Sie hinter den Deklarationen den Wert des Zeitstempels mit einem beliebigen Wert. Weisen Sie anschließend den Variablen für Stunde, Minute und Sekunde **in Abhängigkeit zum Zeitstempel** den passenden Wert durch eine Berechnung zu.

Lösung

```
seconds = time % 60;
minutes = (time / 60) % 60;
hours = (time / 3600) % 24;
```

d) Abschließend wollen wir die berechneten Werte nutzen, um die Zeit auszugeben. Rufen Sie die bekannte Systemfunktion **System.out.println()** auf und geben Sie einen informativen Text aus, der sowohl den gewählten Zeitstempel als auch die Uhrzeit ausgibt. Testen Sie auch das Programm mit einem Klick auf den **execute**-Button, wie es auf dem zweitem Übungsblatt beschrieben wurde.

Lösung

```
System.out.println("Time at " + time + ": " + hours + ":" + minutes
                                                + ":" + seconds);
```

e) Was für Probleme gäbe es, würden wir **seconds**, **minutes** und **hours** als **int** deklarieren, **time** jedoch als **long** belassen?

Compilerfehler, denn wir verringern dann durch die Zuweisung den Wertebereich.

Wir müssten dann explizit mit (int) casten

Komplettlösung für das Programm

```
1  public class Clock {
2
3      public static void main(String[] args) {
4          long time = 45123451;
5          long seconds;
6          long minutes;
7          long hours;
8          seconds = time % 60;
9          minutes = (time / 60) % 60;
10         hours = (time / 3600) % 24;
11         System.out.println("Time at " + time + ": " + hours + ":"
12           + minutes + ":" + seconds);
13     }
14 }
```

Aufgabe 4.5: if-Anweisungen – Bedingungen

In dieser Aufgabe wollen wir uns mit der Auswertung boolescher Ausdrücke beschäftigen. Vervollständigen Sie die folgende Tabelle, indem Sie die folgenden Ausdrücke auswerten:

Ausdruck	boolescher Wert
`5 > 6`	false
`true && false`	**false**
`true || false`	**true**
`(5 < 8) && (3 > 1)`	**true**
`17 < 11 || (1 > 2 && true)`	**false**
`true || false && false`	**true**
`true && false || false`	**false**
`5 * 8 > 22 && 8 < 3 || 3 < 2`	**false**

Aufgabe 4.6: if-Anweisungen – Bedingungen selber schreiben

Überlegen Sie, welche Vorbedingungen erfüllt sein müssen, damit folgende Berechnungen ohne Laufzeitfehler oder Semantikfehler ausgeführt werden können.

a) Die Division zweier Zahlen a/b

`if(b != 0)` (Überprüfung auf Gültigkeit)

`if(b == 0)` (Überprüfung auf Abbruch)

b) Die Wurzel aus der Zahl x ziehen

`if(x >= 0)` (Überprüfung auf Gültigkeit)

`if(x < 0)` (Überprüfung auf Abbruch)

c) Das Berechnen des Flächeninhalts eines Rechtecks mit den Seitenlängen a und b

`if( a > 0 && b > 0)` (Überprüfung auf Gültigkeit)

`if(a <= 0 || b <= 0)` (Überprüfung auf Abbruch)

Aufgabe 4.7: if- und switch-Anweisungen

Wir wollen ein Programm schreiben, das Zahlen in Worten ausschreibt. Dabei betrachten wir nur natürliche Zahlen (inklusive 0). Das Programm wollen wir nun schrittweise mithilfe von if- und switch-Anweisungen entwickeln.

a) Erstellen Sie eine neue Klasse **IntToText** und legen Sie in der main-Methode eine **int**-Variable mit dem Namen **number** an. Fragen Sie in einer **if**-Anweisung ab, ob **number** den Wert 0 hat und geben Sie in diesem Fall „zero“ aus. Initialisieren Sie **number** mit verschiedenen Werten und testen Sie, ob genau dann, wenn **number** den Wert 0 hat, der Text ausgegeben wird.

Lösung

```
1  class IntToText {
2      public static void main(String[] args) {
3          int number = 7;
4
5          if(number == 0) {
6              System.out.println("zero");
7          }
8      }
9  }
```

b) Erweitern Sie Ihr Programm um eine if-Anweisung, die „one“ ausgibt, wenn **number** den Wert 1 hat. Muss diese if-Anweisung vor oder hinter der if-Anweisung aus Aufgabenteil a) stehen?

egal, denn die Bedingungen schließen sich aus

Lösung

```
if(number == 0) {
    System.out.println("zero");
} else if(number == 1) {
    System.out.println("one");
}
```

c) Erweitern Sie ihr Programm um die passenden Ausgaben bei den Werten 2–4.

Lösung

```
if(number == 0) {
    System.out.println("zero");
} else if(number == 1) {
    System.out.println("one");
} else if(number == 2) {
    System.out.println("two");
} else if(number == 3) {
    System.out.println("three");
} else if(number == 4) {
    System.out.println("four");
}
```

d) Des Weiteren interessiert uns, ob der Wert zwischen 5 und 7 oder größer als 7 ist. Geben Sie im Falle, dass der Wert **number** 5, 6 oder 7 ist, den Text „between five and seven“ aus. Ist der Wert größer als 7, soll „greater than seven“ ausgegeben werden.

Lösung

```
if(number == 0) {
    System.out.println("zero");
} else if(number == 1) {
    /* ... */
} else if(number == 4) {
    System.out.println("four");
} else if(number >= 5 && number <= 7) {
    System.out.println("between five and seven");
} else {
    System.out.println("greater than seven");
}
```

e) Wandeln Sie diese Ansammlung von **if**-Anweisungen in eine **switch**-Anweisung um.

Lösung

```
1  class IntToTextSwitch {
2      public static void main(String[] args) {
3          int number = 7;
4
5          switch(number) {
6          case 0:
7              System.out.println("zero"); break;
8          case 1:
9              System.out.println("one"); break;
10         case 2:
11             System.out.println("two"); break;
12         case 3:
13             System.out.println("three"); break;
14         case 4:
15             System.out.println("four"); break;
```

```
16          case 5: case 6: case 7:
17              System.out.println("between five and seven"); break;
18          default:
19              System.out.println("greater than seven"); break;
20          }
21      }
22  }
```

Ergänzende Aufgaben

Aufgabe 4.8: Zahlen-Ausschreibung

Betrachten Sie Ihren Code aus Aufgabe 4.7. Welche Änderungen sind nötig, um die Zahlen bis 99 richtig auszuschreiben? Eine konkrete Implementierung ist dabei nicht notwendig.

Die Zahlen zwischen 0 und 20 müssen einzeln behandelt werden

Die Zahlen zwischen 20 und 99 bestehen immer aus dem selben Muster

Zehnerstelle + Einerstelle (z.B. twenty+four, twenty+five, etc.)