

# Programmierkurs Java

Dr.-Ing. Dietrich Boles

## Aufgaben zu UE 5 - Anweisungen

(Stand 14.06.2017)

### Aufgabe 1:

Schreiben Sie ein Java-Programm, das die Eingabe eines char-Wertes *zeichen* über die Tastatur erwartet. Das Programm soll anschließend die Eingabe überprüfen und, falls es sich um einen Groß- oder Kleinbuchstaben oder eine Ziffer handelt, ins Morse-Alphabet übersetzen und die entsprechende Morse-Codierung auf den Bildschirm ausgeben. Hier das Morse-Alphabet:

```
A .-      B -...   C -.-.   D -..    E .       F ..-.   G --.
H ....   I ..     J .---  K -.-   L .-..   M --    N -.
O ---    P .--.   Q --.-  R .-.   S ...   T -     U ..-
V ...-   W .--   X -.-.  Y -.-   Z --..  0 ----- 1 .-----
2 ..---- 3 ...-- 4 ..... 5 ..... 6 -..... 7 --.... 8 ----..
9 ----. .
```

Beispiel:

Eingabe: a

Ausgabe: .-

Eingabe: 4

Ausgabe: ....-

### Aufgabe 2:

Sie kennen sicher aus der Schule noch die Römischen Zahlen; bspw. steht die Zeichenkette "XIV" für die Dezimalzahl "14" und die Römische Zahl "MDCXXV" kennzeichnet die Dezimalzahl "1625". Insgesamt werden folgende Buchstaben für die Bildung von Römischen Zahlen verwendet: I, V, X, L, C, D, M.

Schreiben Sie ein Java-Programm, das vom Benutzer einen char-Wert abfragt, diesen daraufhin überprüft, ob es sich um einen gültigen Buchstaben zur Bildung Römischer Zahlen handelt, und eine entsprechende Meldung auf den Bildschirm ausgibt.

### Aufgabe 3:

Schreiben Sie ein Java-Programm, das zunächst die Eingabe eines positiven int-Wertes *max* und eines positiven int-Wertes *div* über die Tastatur erwartet. Anschließend soll das Programm alle Zahlen zwischen 0 und *max*, die sich ohne Rest durch *div* dividieren lassen, auf den Bildschirm ausgeben.

Beispiel:

```
Eingabe: max: 30
          div: 9
Ausgabe: 0
          9
          18
          27
```

### Aufgabe 4:

Ein Musik-Anbieter im Internet habe folgendes Geschäftsmodell: Jeder Nutzer kann sich zunächst 5 Musikstücke kostenlos downloaden. Für die nächsten 30 Stücke muss er jeweils 20 Cent bezahlen und für jeden weiteren Download 15 Cent.

Schreiben Sie ein Java-Programm, das einem Nutzer ermöglicht, die Anzahl an insgesamt gewünschten Downloads anzugeben, das daraufhin die insgesamt entstehenden Gebühren berechnet und diese auf den Bildschirm ausgibt.

### Aufgabe 5:

Schreiben Sie ein Java-Programm, das zunächst die Eingabe eines positiven int-Wertes *anzahl* über die Tastatur erwartet. Anschließend soll das Programm *anzahl*-mal ein Sternchen (\*) auf den Bildschirm ausgeben. Realisieren Sie das Programm viermal, und zwar einmal mit Hilfe einer `for`-Schleife, einmal mit Hilfe einer `while`-Schleife, einmal mit Hilfe einer `do`-Schleife und einmal mit Hilfe einer „Endlos“-Schleife (`while (true)`) und der `break`-Anweisung.

Beispiele:

```
Eingabe: 5
Ausgabe: *****
Eingabe: 9
Ausgabe: *********
```

### Aufgabe 6

Schreiben Sie ein Java-Programm, das zunächst die Eingabe eines char-Wertes *zeichen* über die Tastatur erwartet. Das Programm soll anschließend die Eingabe überprüfen und, falls es sich um einen Buchstaben handelt, den Buchstaben so oft auf den Bildschirm ausgeben, wie es seiner Position im Alphabet entspricht. Realisieren Sie das Programm viermal, und zwar einmal mit Hilfe einer `for`-Schleife,

einmal mit Hilfe einer while-Schleife, einmal mit Hilfe einer do-Schleife und einmal mit Hilfe einer Endlos-Schleife (while (true)) und der break-Anweisung.

Beispiele:

Eingabe: d

Ausgabe: dddd

Eingabe: G

Ausgabe: GGGGGGG

### Aufgabe 7:

Schreiben Sie ein Java-Programm, das nach Eingabe einer ungeraden Natürlichen Zahl „basislaenge“ ein Sterndreieck auf den Bildschirm ausgibt. Die unterste Reihe soll dabei aus „basislaenge“ Sternchen bestehen. Beispielausgabe für „basislaenge == 9“:

```
  *
 ***
*****
*****
*****
```

### Aufgabe 8:

Schreiben Sie ein Java-Programm, das zunächst die Eingabe eines int-Wertes *durchmesser* über die Tastatur erwartet. Überprüfen Sie die Eingabe. Liegt der Eingabewert zwischen 5 und 50 und ist es ein ungerader Wert, dann soll das Programm folgendes bewirken: Es soll eine Raute mit dem entsprechenden Durchmesser - wie in den Beispielen ersichtlich - auf den Bildschirm ausgegeben werden.

Beispiele:

Eingabe: 5

Ausgabe:

```
  .
 / \
 .   .
 \ /
  .
```

Eingabe: 7

Ausgabe:

```
  .
 / \
 /   \
 .     .
 \   /
 \   /
  .
```

## Aufgabe 9:

In dieser Aufgabe geht es darum, Treppen der folgenden Form auf den Bildschirm zu zeichnen:

```
+--+
| |
+-----+
|       |
+-----+
|           |
+-----+
|               |
```

Schreiben Sie ein Java-Programm, das zunächst vom Benutzer einen positiven int-Wert erfragt. Dieser gibt die Höhe der Treppe an (in dem Beispiel 4). Das Programm soll daraufhin eine Treppe in der oben skizzierten Form auf den Bildschirm ausgeben.

## Aufgabe 10:

In dieser Aufgabe geht es darum, Häuser der folgenden Form auf den Bildschirm zu zeichnen:

```
  /\      +
 /  \    /  \
+-----+ +-----+
|       | |       |
|       | |       |
|       | |       |
|       | +-----+
+-----+
```

Schreiben Sie ein Java-Programm, das zunächst vom Benutzer einen positiven int-Wert  $> 4$  erfragt. Dieser gibt die Höhe sowie Breite des Haussockels (quadratisch!) an (in den beiden Beispielen 6 (links) bzw. 5 (rechts)). Das Programm soll daraufhin ein Haus in der oben skizzierten Form auf den Bildschirm ausgeben.

## Aufgabe 11:

**Motivation:** Auf dem Tisch liegen  $n$  äußerlich völlig gleiche Münzen, von denen  $n-1$  gleiches Gewicht haben, während eine, die gefälschte Münze, leichter ist. Dem Computer soll ein Verfahren beigebracht werden, das ihn befähigt, im Dialog mit dem Benutzer, dem eine Balkenwaage mit zwei Waagschalen aber ohne Wägesatz zur Verfügung steht, die gefälschte Münze mit möglichst wenig Fragen zu finden.

**Lösungsidee:** Wir nutzen das Halbierungsverfahren. Wenn die Anzahl  $n$  der Münzen auf dem Tisch gerade ist, lassen wir den Nutzer jeweils die Hälfte der Münzen auf jede der beiden Waagschale legen: In der sich hebenden Schale befindet sich die gefälschte Münze. Ist die Münzenzahl ungerade, so reduzieren wir sie um eins und gehen wie eben beschrieben vor. Ergibt sich ein Gleichgewicht, war die beiseitegelegte Münze die gesuchte.

**Aufgabe:** Schreiben Sie ein Java-Programm zur Lösung des Problems. Die Münzen – 1000 Stück - werden dabei durch Zahlen von 1 – 1000 ersetzt. Denken Sie sich vor Programmstart eine Zahl zwischen 1 und 1000 aus (die gefälschte Münze!) und lassen Sie den Computer die Zahl suchen. Der Computer soll dabei gemäß dem Halbierungsverfahren vorgehen. Er kann Ihnen dazu Ja-Nein-Fragen der Art „Ist der Münzhaufen rechts schwerer?“ oder „Sind die beiden Münzhaufen gleichschwer?“ stellen.

Beispielablauf des Programms (Benutzereingaben in <>):

```
Sind die Muenzen 1 bis 500 schwerer als die Muenzen 501 bis 1000(j/n)?<j>
Sind die Muenzen 501 bis 750 schwerer als die Muenzen 751 bis
1000(j/n)?<j>
Sind die Muenzen 751 bis 875 schwerer als die Muenzen 876 bis
1000(j/n)?<n>
Sind die Muenzen 752 bis 813 gleichschwer wie die Muenzen 814 bis
875(j/n)?<n>
Sind die Muenzen 752 bis 813 schwerer als die Muenzen 814 bis 875(j/n)?<n>
Sind die Muenzen 752 bis 782 schwerer als die Muenzen 783 bis 813(j/n)?<j>
Sind die Muenzen 784 bis 798 gleichschwer wie die Muenzen 799 bis
813(j/n)?<j>
Die gesuchte Muenze hat die Nummer 783 !
```

## Aufgabe 12:

**Motivation:** In eine Stadt wird eine ansteckende Krankheit (z.B. Influenza) eingeschleppt. Was passiert? Entsteht eine Epidemie oder erlischt die Infektion? Wie entwickelt sich der Krankenstand; welches ist sein höchster Wert und wann wird dieser erreicht? Wie viele Personen werden irgendwann einmal von der Krankheit betroffen worden sein? Es ist ein mathematisches Modell zu entwerfen und für plausible Daten auf dem Computer zu erproben!

**Modell:** Dazu wird die Bevölkerung in drei Klassen eingeteilt. Es sei

- $x_t$  die Anzahl der zum Zeitpunkt  $t$  noch Gesunden, aber für die Krankheit Anfälligen
- $y_t$  die Anzahl der zum Zeitpunkt  $t$  Erkrankten und damit Ansteckenden sowie
- $z_t$  die Anzahl der zum Zeitpunkt  $t$  aus dem Krankheitsgeschehen Ausgeschiedenen, der Immunen (Genesene, Isolierte oder Tote)

Wir nehmen an, dass die Gesamtbevölkerungszahl  $n$  konstant ist (es gibt also keine Geburten und keine Zu- oder Abwanderungen); zu jedem Zeitpunkt  $t$  gilt damit:

$$x_t + y_t + z_t = n$$

Es finden zweierlei Übergänge statt: Infektionen und Immunisierungen. Erstere machen aus Gesunden Kranke, letztere aus Kranken Immune. Wir betrachten den Prozess zu diskreten Zeitpunkten; Zeiteinheit sei die Dauer, innerhalb der ein infektiös Erkrankter die Krankheit weiterverbreiten kann (bei der Influenza A im Mittel etwa zwei Tage).

Der Infektionsvorgang lässt sich durch folgende Gleichung beschreiben:

$$x_{t+1} - x_t = -a * x_t * y_t \quad (a > 0)$$

d.h. die Abnahme der Zahl der Gesunden in der Periode (t, t+1) ist zur Anzahl der Gesunden und zur Anzahl der Kranken proportional; der Parameter  $a$  heißt *Infektionsrate*.

Den Immunisierungsvorgang beschreiben wir durch die Gleichung

$$z_{t+1} - z_t = b * y_t \quad (0 < b \leq 1)$$

d.h. die Zunahme der Immunen in der Periode (t, t+1) ist zur Anzahl der Kranken proportional.; der Parameter  $b$  heißt *Immunisierungsrate*.

Die drei Gleichungen beschreiben den Prozess vollständig; bei gegebenen Werten für  $a$  und  $b$  und bei gegebenen Anfangswerten  $x_0$ ,  $y_0$  und  $z_0$  lassen sich alle folgenden Werte berechnen.

**Aufgabe:** Schreiben Sie ein Java-Programm, das zunächst die Infektionsrate, die Immunisierungsrate und die Anfangswerte für die Gesunden und die Kranken von der Tastatur einliest. Anschließend soll das Programm 60 Zeiteinheiten simulieren und zu jeder Zeiteinheit die Anzahl der Kranken auf den Bildschirm ausgeben. Außerdem soll sich das Programm den höchsten Krankheitsstand merken und diesen Wert im Anschluss an die 60 Zeiteinheiten ebenfalls auf dem Bildschirm ausgeben.

**Test:** Testen Sie Ihr Programm. Gute Testwerte sind bspw. eine Infektionsrate von 0.0003, eine Immunisierungsrate von 0.06, eine Anfangszahl von 1590 Gesunden und eine Anfangszahl von 10 Kranken.

### Aufgabe 13:

Es soll ein Programm entwickelt werden, das berechnet, ob sich zwei Kreise überlappen, berühren oder nichts von beidem.

**Aufgabe:** Fordern Sie den Benutzer zunächst auf, eine Zahl `anzahl` einzugeben, die die Anzahl an Überprüfungsszenarien angibt. In jedem Überprüfungsszenario wird der Benutzer dann aufgefordert sechs int-Werte einzugeben: für zwei Kreise jeweils die x-Koordinate, die y-Koordinate und den Radius (`x1 y1 r1 x2 y2 r2`). Der Radius soll dabei immer ein positiver Wert sein. Anschließend soll Ihr Programm in jedem Überprüfungsszenario berechnen, ob die beiden Kreise überlappen (Ausgabe: "Circles overlap"), ob sie sich berühren (Ausgabe: "Circles touch") oder keines von beidem (Ausgabe: "Circles are separated").

#### Beispiel:

| Eingabe:      | Ausgabe:              |
|---------------|-----------------------|
| 3             |                       |
| 0 0 10 0 0 5  | Circles overlap       |
| 0 0 10 0 14 4 | Circles touch         |
| 0 0 10 0 15 4 | Circles are separated |

### Aufgabe 14:

Wandeln Sie, falls möglich, folgende if-Anweisung in eine äquivalente switch-Anweisung um. Falls Sie meinen, das ist nicht möglich, begründen Sie dies.

```

int i = IO.readInt();
if (i > 0 && i < 3) {
    System.out.println("gut");
} else {
    System.out.println("schlecht");
}

```

### Aufgabe 15:

Wandeln Sie, falls möglich, folgende if-Anweisung in eine äquivalente switch-Anweisung um. Falls Sie meinen, das ist nicht möglich, begründen Sie dies.

```

int i = IO.readInt();
if (i < 10) {
    System.out.println("gut");
} else {
    System.out.println("schlecht");
}

```

### Aufgabe 16:

Ersetzen Sie im folgenden Programm die if-Anweisung durch eine switch-Anweisung, ohne die Semantik des Programms zu verändern.

```

int i = IO.readInt();
while (true) {
    if (i == 0) {
        System.out.println("fertig");
        break;
    } else {
        System.out.println(i);
        i--;
    }
}

```

### Aufgabe 17:

Wandeln Sie die folgende if-Anweisung in eine äquivalente switch-Anweisung um:

```

int zahl = IO.readInt();
if (zahl > 3 && zahl < 7) {
    if (zahl > 4 && zahl < 10) {
        System.out.println("ja");
    } else {
        System.out.println("weiss nicht");
    }
} else if (zahl >= -1 && zahl <= 1) {
    System.out.println("nein");
} else {
    System.out.println("vielleicht");
}

```

### Aufgabe 18:

Schreiben Sie ein Java-Programm, das die Anzahl der Überträge beim Addieren zweier Zahlen berechnet.

**Algorithmus:** Zunächst sollen vom Benutzer zwei nicht negative Zahlen mit jeweils weniger als 10 Ziffern eingelesen werden. Sie können dazu die aus der Vorlesung

bekannte Klasse IO verwenden. Anschließend soll die Anzahl an Überträgen beim zifferweisen Addieren der beiden Zahlen von rechts nach links berechnet werden. Zum Schluss soll die Anzahl an Überträgen auf den Bildschirm ausgegeben werden.

### Beispiel:

```
Zahl: 123
Zahl: 594
Anzahl an Uebertraegen: 1
```

### Aufgabe 19:

Schreiben Sie ein Java-Programm, das überprüft, ob eine bestimmte Ziffer in einer bestimmten Zahl vorkommt oder nicht.

Zunächst sollen vom Benutzer ein int-Wert *zahl* und ein char-Wert *ziffer* eingelesen werden. Das Programm soll überprüfen, ob *ziffer* eine Ziffer in *zahl* ist.

### Beispiele:

```
Zahl: 12034
Ziffer: 2
enthalten

Zahl: -2345
Ziffer: 7
Nicht enthalten
```

### Aufgabe 20:

Schauen Sie sich die folgende (unendliche) Zahlenfolge an:

1, 2, 3, 4, 5, 4, 3, 2, 3, 4, 5, 6, 5, 4, 3, 4, 5, 6, 7, 6, 5, ...

Schreiben Sie ein Java-Programm, das die Zahlenfolge auf den Bildschirm ausgibt.

### Aufgabe 21:

Schauen Sie sich die folgende (unendliche) Zahlenfolge an:

1 2 3 4 5 4 3 2 2 3 4 5 6 7 6 5 4 3 3 4 5 6 7 8 9 8 7 6 5 4 4 5 6 7 8 9 10  
11 10 9 8 7 6 5 5 6 7 8 9 10 11 12 13 12 11 10 9 8 7 6...

Schreiben Sie ein Java-Programm, das die Zahlenfolge auf den Bildschirm ausgibt.

### Aufgabe 22:

Gegeben sei folgendes Java-Programm:

```
public static void main(String[] args) {
    int bis = IO.readInt();
    for (int i=0; i<bis; i++) {
        if (i % 2 != 0) {
            if (i < bis/2) {
                continue;
            }
        }
    }
}
```



```

        }
        bis--;
    }
    System.out.println(i);
}
int i = IO.readInt();
System.out.println(i);
}

```

Wandeln Sie das Programm um in ein (syntaktisch korrektes!) äquivalentes Java-Programm, das anstelle der for-Schleife eine while-Schleife verwendet. *Äquivalent* bedeutet: dieselben Eingaben erzeugen dieselben Ausgaben.

### Aufgabe 23:

Gegeben sei folgendes Java-Programm:

```

public static void main(String[] args) {
    int i = IO.readInt();
    if (i == 2)
        System.out.println("zwei");
    if (i == 2 || i == 3)
        System.out.println("zwei oder drei");
    else if (i == 4)
        System.out.println("vier");
    else if (i >= 4 && i <= 10)
        System.out.println("zwischen vier und zehn");
    else
        System.out.println("anders");
}
}

```

Wandeln Sie das Programm um in ein (syntaktisch korrektes!) äquivalentes Java Programm, in dem keine if-Anweisungen und maximal eine switch-Anweisung verwendet werden. *Äquivalent* bedeutet: dieselben Eingaben erzeugen dieselben Ausgaben.

### Aufgabe 24:

Gegeben sei folgendes Java-Programm:

```

int anzahl = IO.readInt();
int i = 0;
while (true) {
    if (i >= anzahl)
        break;
    int j = 0;
    while (true) {
        if (j > i)
            break;
        System.out.print(j);
        j++;
    }
    System.out.println();
    i++;
}

```

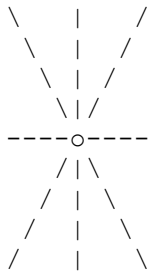
Wandeln Sie das Programm um in ein äquivalentes Java-Programm, in dem keine break-Anweisungen mehr vorkommen. *Äquivalent* bedeutet: dieselben Eingaben erzeugen dieselben Ausgaben.

### Aufgabe 25:

Bald ist Winter, und hoffentlich wird bald der erste Schnee fallen. Sie sollen diesem Tag vorgreifen und schon jetzt Schneeflocken auf den Bildschirm zaubern.

**Aufgabe:** Schreiben Sie ein Java-Programm, das den Benutzer zunächst auffordert, eine Zahl größer gleich 0 einzugeben und das anschließend eine entsprechend große Schneeflocke der folgenden Form auf den Bildschirm ausgibt:

Größe = 4:



### Aufgabe 26:

Bald ist Weihnachten und Sie möchten sich selbst mal eine Freude machen, indem Sie einen ASCII-Weihnachtsbaum auf den Bildschirm Ihres Computers malen.

**Aufgabe:** Schreiben Sie ein Java-Program, das den Benutzer zunächst auffordert, eine Zahl größer gleich 2 einzugeben und das anschließend einen entsprechend hohen Weihnachtsbaum der folgenden Form auf den Bildschirm ausgibt:

Höhe = 2:

```
  +-+
  | |
 +-+ +-+
```

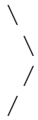
Höhe = 4:

```
      +-+
      | |
    +-+ +-+
    |   |
 +-+   +-+
 |     |
 +-+   +-+
```

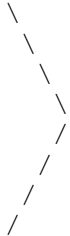
### Aufgabe 27:

Schreiben Sie ein Java-Programm, das den Benutzer zunächst auffordert, eine Zahl größer gleich 2 einzugeben und das, sobald der Benutzer eine entsprechende Zahl eingegeben hat, einen entsprechend großen Pfeil der folgenden Form auf den Bildschirm ausgibt:

Zahl = 2:



Zahl = 4:



### Aufgabe 28:

Bei dieser Aufgabe sollen Sie ein Programm entwickeln, das es einem Menschen erlaubt, gegen den Rechner eine vereinfachte Black-Jack-Variante zu spielen. Der Programmablauf hat dabei folgende Gestalt:

- (1) Das Programm generiert eine Zufallszahl zwischen 16 und 21. Es hält diese Zahl geheim (*Programmzahl*)
- (2) Das Programm generiert eine Zufallszahl zwischen 2 und 11 und teilt sie dem Benutzer mit (*Menschenzahl*).
- (3) Der Benutzer wird aufgefordert, „Weiter“ oder „Stopp“ zu sagen.
- (4) Sagt der Benutzer „Weiter“ werden die Schritte (2) und (3) wiederholt. Die generierten Menschenzahlen werden dabei jeweils addiert.
- (5) Sagt der Benutzer „Stopp“ endet das Programm. Das Programm endet auch bei (2), wenn die Summe der generierten Menschenzahlen größer als 21 ist. In letzterem Fall hat der Mensch unmittelbar verloren. Ansonsten gilt: Ist bei Programmende die Programmzahl größer als die Summe der Menschenzahlen, hat der Rechner gewonnen. Im umgekehrten Fall der Mensch. Bei Gleichstand heißt es Unentschieden. Das Ergebnis wird jeweils verkündet.

**Aufgabe:** Schreiben Sie ein entsprechendes Java-Programm, das sich an dem beispielhaften Programmablauf und den entsprechenden Ausgaben orientiert.

Hinweis für die Berechnung von Zufallszahlen:

```
double r = Math.random(); // liefert Zufallszahl zwischen 0 (einschließlich) und 1 (ausschließlich)
```

Beispiel für einen Programmablauf (in <> stehen Benutzereingaben):

```
Ich habe eine geheime Programmzahl generiert!  
Generierte Menschenzahl: 3  
Summe der Menschenzahlen: 3  
Weiter (w) oder Stopp (s): <w>  
Generierte Menschenzahl: 4  
Summe der Menschenzahlen: 7
```

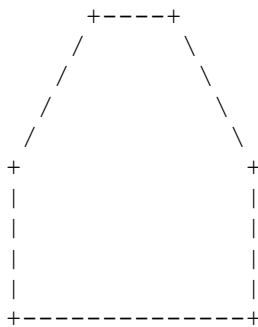
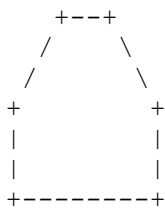
```

Weiter (w) oder Stopp (s): <w>
Generierte Menschenzahl: 5
Summe der Menschenzahlen: 12
Weiter (w) oder Stopp (s): <w>
Generierte Menschenzahl: 5
Summe der Menschenzahlen: 17
Weiter (w) oder Stopp (s): <s>
Programmzahl = 19; Summe der Menschenzahlen = 17
Der Rechner hat gewonnen!

```

### Aufgabe 29:

Schreiben Sie ein Programm, das den Benutzer auffordert, eine Zahl n größer als 0 einzugeben, und das anschließend ein Haus-ähnliches Gebilde der folgenden Form auf den Bildschirm zeichnet:



n (in den Beispielen ist oben n = 2 und unten n = 4) soll dabei die Höhe des Hauses, die Höhe des Daches und die Länge des Dachfirst ausmachen.

### Aufgabe 30:

Schreiben Sie ein Java-Programm, das zunächst die Eingabe eines positiven int-Wertes n über die Tastatur erwartet. Es soll anschließend eine Schale - wie in den Beispielen ersichtlich - auf den Bildschirm ausgegeben werden.

Beispiele:

Eingabe: 3

Ausgabe:



Eingabe: 1

Ausgabe:



### Aufgabe 31:

Implementieren Sie ein Java-Programm, das zunächst vom Benutzer einen int-Wert  $n$  größer als 0 abfragt. Anschließend soll Ihr Programm einen Hammer der folgenden Form auf den Bildschirm ausgeben:

|  |   |
|--|---|
| <b>Beispiel n = 1:</b><br>+--+<br>/... <br>+----+<br> <br> | <b>Beispiel n = 2:</b><br>+-----+<br>/..... <br>/..... <br>+-----+<br> <br> <br> <br> |
|--|---|

|   |  |
|---|--|
| <b>Beispiel n = 3:</b><br>+-----+<br>/..... <br>/..... <br>/..... <br>+-----+<br> <br> <br> <br> <br> | <b>Beispiel n = 4:</b><br>+-----+<br>/..... <br>/..... <br>/..... <br>/..... <br>/..... <br>+-----+<br> <br> <br> <br> <br> <br> <br> <br> |
|---|--|

Dabei gilt:

- Höhe des Kopfes =  $n + 2$
- Länge des Kopfes (oberste Zeile) =  $2n + 2$

- Länge des Kopfes (unterste Zeile) =  $3n + 3$
- Höhe des Stieles =  $2n$
- Positionierung des Stiels = Mitte der untersten Zeile des Kopfes

### Aufgabe 32:

Implementieren Sie ein Java-Programm, das zunächst vom Benutzer einen ungeraden int-Wert  $n$  größer als 0 abfragt. Anschließend soll Ihr Programm eine „Maus“ der folgenden Form auf den Bildschirm ausgeben:

|   |  |
|---|--|
| <p>Beispiel <math>n = 1</math>:</p> <pre> +--+ /... -- +----+</pre> | <p>Beispiel <math>n = 3</math>:</p> <pre> +-----+ /.....  /..... ----- /.....  +-----+</pre> |
|---|--|

|  |  |
|--|--|
| <p>Beispiel <math>n = 5</math>:</p> <pre> +-----+ /.....  /.....  /..... ----- /.....  /.....  +-----+</pre> |  |
|--|--|

Dabei gilt:

- Höhe der Maus =  $n + 2$
- Länge des Mauskörpers (oberste Zeile) =  $2n + 2$
- Länge des Mauskörpers (unterste Zeile) =  $3n + 3$
- Länge des Schwanzes =  $2n$
- Positionierung des Schwanzes = Mitte des Mauskörpers

### Aufgabe 33:

In einem Online-Shop kostet das Buch „Programmieren spielend gelernt mit dem Java-Hamster-Modell“ 26,00 EUR. Beim Kauf von mindestens 6 Exemplaren dieses Buches gibt es ab dem 6. Exemplar 10 % Rabatt. Beim Kauf von mindestens 11

Exemplaren gibt es ab dem 6. Exemplar 10 % Rabatt und ab dem 11. Exemplar 15 % Rabatt (Hinweis: In Deutschland darf man auf Bücher eigentlich gar keine Rabatte geben).

**Aufgabe:** Schreiben Sie ein Java-Programm, bei dem ein Benutzer nach der Anzahl an zu kaufenden Exemplaren des Buches gefragt, die entsprechende Gesamtkosten berechnet und ausgegeben werden.

**Beispiele** (Benutzereingaben in <>):

```
Anzahl Exemplare: <3>  
Kosten = 78.0
```

```
Anzahl Exemplare: <7>  
Kosten = 176.8
```

```
Anzahl Exemplare: <13>  
Kosten = 313.3
```

### Aufgabe 34:

*Ziel 100* ist als ein Strategiespiel für zwei Spieler. Begonnen wird mit einer zufälligen Zahl kleiner 30. Die Spieler addieren abwechselnd eine selbst gewählte ganze Zahl zwischen 1 und 10 zu dieser Zahl. Gewonnen hat der Spieler, der als erster 100 oder mehr erreicht.

Implementieren Sie das Spiel *Ziel 100* in Java (auf eine imperative Art und Weise), so dass zwei menschliche Spieler gegeneinander antreten können.

Hinweis für die Berechnung von Zufallszahlen:

```
double r = Math.random(); // liefert Zufallszahl zwischen 0 (einschließlich) und 1 (ausschließlich)
```

Beispiel für einen möglichen Spielablauf (Eingaben in Klammern <>):

```
Die Zahl ist 11  
Spieler 1! Zahl eingeben (>= 1, <= 10):<7>  
Die Zahl ist 18  
Spieler 2! Zahl eingeben (>= 1, <= 10):<9>  
Die Zahl ist 27  
Spieler 1! Zahl eingeben (>= 1, <= 10):<10>  
Die Zahl ist 37  
...  
Die Zahl ist 89  
Spieler 2! Zahl eingeben (>= 1, <= 10):<8>  
Die Zahl ist 97  
Spieler 1! Zahl eingeben (>= 1, <= 10):<3>  
Zahl = 100! Spieler 1 hat gewonnen!
```

### Aufgabe 35:

Schreiben Sie ein Java-Programm, das nach Eingabe einer natürlichen Zahl „hoehe“ ein Sterndreieck der entsprechenden Höhe auf den Bildschirm ausgibt. Beispielausgabe für „hoehe == 5“:

```
*
***
*****
*****
*****
```

### Aufgabe 36:

Implementieren Sie auf imperative Art und Weise das folgende kleine Taktikspiel für 2 Spieler.

Anfangs haben beide Spieler je einen Haufen von 100 Kugeln. Gespielt werden 10 Runden. In jeder Runde wählt jeder Spieler (geheim) eine bestimmte Anzahl ( $\geq 0$ ) an Kugeln aus, die von seinem Haufen entfernt werden. Jeweils der Spieler mit der größeren Anzahl an gewählten Kugeln gewinnt die Runde und einen Punkt. Bei Unentschieden bekommt keiner der Spieler einen Punkt. Das Spiel gewinnt der Spieler, der nach 10 Runden die meisten Punkte hat. Bei Gleichstand endet das Spiel unentschieden.

Orientieren Sie sich bei der Implementierung des Spiels an folgendem Beispielablauf (Benutzereingaben in  $\langle \rangle$ ):

```
Runde 1 von 10 Runden
Spieler 1: Wie viele Kugeln von 100? <10>
Spieler 2: Wie viele Kugeln von 100? <6>
Spieler 1 hat die Runde gewonnen!
Spielstand nach Runde 1:
Spieler 1 hat 1 Punkte und 90 Restkugeln
Spieler 2 hat 0 Punkte und 94 Restkugeln
```

```
Runde 2 von 10 Runden
Spieler 1: Wie viele Kugeln von 90? <20>
Spieler 2: Wie viele Kugeln von 94? <20>
Die Runde endet unentschieden!
Spielstand nach Runde 2:
Spieler 1 hat 1 Punkte und 70 Restkugeln
Spieler 2 hat 0 Punkte und 74 Restkugeln
```

```
Runde 3 von 10 Runden
Spieler 1: Wie viele Kugeln von 70? <30>
Spieler 2: Wie viele Kugeln von 74? <2>
Spieler 1 hat die Runde gewonnen!
Spielstand nach Runde 3:
Spieler 1 hat 2 Punkte und 40 Restkugeln
Spieler 2 hat 0 Punkte und 72 Restkugeln
```

...

```
Runde 9 von 10 Runden
Spieler 1: Wie viele Kugeln von 12? <2>
Spieler 2: Wie viele Kugeln von 24? <4>
Spieler 2 hat die Runde gewonnen!
Spielstand nach Runde 9:
Spieler 1 hat 2 Punkte und 10 Restkugeln
Spieler 2 hat 6 Punkte und 20 Restkugeln
```

```
Runde 10 von 10 Runden
Spieler 1: Wie viele Kugeln von 10? <10>
Spieler 2: Wie viele Kugeln von 20? <20>
Spieler 2 hat die Runde gewonnen!
Spielstand nach Runde 10:
```



```
Spieler 1 hat 2 Punkte und 0 Restkugeln  
Spieler 2 hat 7 Punkte und 0 Restkugeln
```

```
Spieler 2 hat gewonnen!
```

### Aufgabe 37:

Implementieren Sie ein Programm, das zählt und anschließend ausgibt, wie viele Eingaben von Kleinbuchstaben der Benutzer innerhalb von N Sekunden (hier N = 5) tätigen kann. Gehen Sie dabei folgendermaßen vor:

Nach Ausgabe von „Start“, misst das Programm die aktuelle Zeit (Startzeit). Anschließend fordert es den Benutzer wiederholt auf, Zeichen einzugeben (`IO.readChar`), Nach der Eingabe überprüft das Programm jeweils, ob ab der Startzeit N oder mehr Sekunden vergangen sind. Ist dies der Fall, gibt das Programm „Ende“ sowie die Anzahl der erreichten Eingaben aus, wobei allerdings nur die Eingabe von Kleinbuchstaben berücksichtigt wird.

Zum Messen der Zeit benutzen Sie bitte folgende Anweisung:

```
long zeit = System.currentTimeMillis();
```

Nach Ausführung dieser Anweisung enthält die Variable `zeit` die Anzahl der Millisekunden, die seit dem 1.1.1970 (00:00 Uhr) und dem Zeitpunkt der Ausführung der Anweisung vergangen sind.

Beispiel für einen Programmablauf (Eingaben in <>):

```
Start  
Eingabe: <a>  
Eingabe: <9>  
Eingabe: <a>  
Eingabe: <a>  
Eingabe: <z>  
Ende  
Sie haben 4 Eingaben erreicht!
```

### Aufgabe 38:

Bei dieser Aufgabe geht es um die Entwicklung eines Programms, das es einem Benutzer erlaubt, sein Zeitabschätzungsgefühl zu trainieren.

**Aufgabe:** Schreiben Sie ein Java-Programm, das folgendes tut:

- (1) Es wird eine Zufallszahl zwischen 10 und 20 generiert. Sie spiegelt die Anzahl an Sekunden wieder, die der Benutzer abschätzen soll (erwartete Zeit).
- (2) Die Zahl wird ausgegeben.
- (3) Der Benutzer wird zweimal zu einer beliebigen Eingabe aufgefordert. Unmittelbar nach jeder Eingabe wird die Zeit gemessen. Nutzen Sie dazu die Funktion `System.currentTimeMillis()`, die als `long`-Wert die Anzahl an Millisekunden liefert, die zwischen dem 1.1.1970 und dem Zeitpunkt ihres Aufrufs verstrichen ist: `long zeit = System.currentTimeMillis();`
- (4) Berechnen Sie die zwischen den beiden Eingaben verstrichene Zeit.
- (5) Berechnen Sie und geben Sie die erwartete Zeit, die tatsächlich verstrichene Zeit, die erlaubte Abweichung (hier A = 1 Sekunde) und die tatsächliche Abweichung aus.

(6) Ist die tatsächliche Abweichung größer als die erwartete Abweichung, beginnt alles von neuem. Im anderen Fall wird das Programm beendet.

Orientieren Sie sich dabei, was den Programmablauf als auch was die Ein- und Ausgaben angeht, an folgendem Beispiel (Eingaben stehen in <>):

Bitte warten Sie möglichst genau 14 Sekunden zwischen Ihren nächsten beiden Eingaben!

Beliebige Eingabe, um die Zeitnahme zu starten: <>

Beliebige Eingabe, um die Zeitnahme zu stoppen: <>

Erwartete Zeit = 14 Sekunden

Ihre Zeit = 12 Sekunden plus 770 Millisekunden

Erlaubte Abweichung = 1 Sekunden

Ihre Abweichung = 1 Sekunden plus 230 Millisekunden

Das war nicht genau genug: Bitte wiederholen!

Bitte warten Sie möglichst genau 15 Sekunden zwischen Ihren nächsten beiden Eingaben!

Beliebige Eingabe, um die Zeitnahme zu starten: <>

Beliebige Eingabe, um die Zeitnahme zu stoppen: <>

Erwartete Zeit = 15 Sekunden

Ihre Zeit = 15 Sekunden plus 205 Millisekunden

Erlaubte Abweichung = 1 Sekunden

Ihre Abweichung = 0 Sekunden plus 205 Millisekunden

Ziel erreicht: Glückwunsch!

### Aufgabe 39:

Die **Russische Bauernmultiplikation** (auch *Ägyptisches Multiplizieren*, *Abessinische Bauernregel* oder *Verdopplungs-Halbierungs-Methode* genannt) ist ein einfaches Verfahren zur Multiplikation zweier natürlicher Zahlen. Am Beispiel  $25 \times 13 = 325$  wird im Folgenden das Schema der russischen Bauernmultiplikation dargestellt:

| Linke Seite | Rechte Seite | Dies addieren |
|-------------|--------------|---------------|
| 25          | 13           | 13            |
| 12          | 26           | -             |
| 6           | 52           | -             |
| 3           | 104          | 104           |
| 1           | 208          | 208           |
|             |              | Ergebnis: 325 |

Der Algorithmus besteht aus folgenden Schritten (aus Wikipedia):

- Man schreibt die beiden zu multiplizierenden Zahlen nebeneinander.
- Auf der linken Seite werden die Zahlen jeweils halbiert (Reste abgerundet) und die Ergebnisse untereinander geschrieben, bis man zur 1 gelangt.
- Auf der rechten Seite werden die Zahlen verdoppelt und untereinander geschrieben.
- Die rechts stehenden (verdoppelten) Zahlen werden gestrichen, wenn die links stehende Zahl gerade ist.



### Aufgabe 43:

*Mini Dice* ist ein einfaches Glücksspiel mit zwei Würfeln (jeder mit Augenzahlen zwischen 1 und 6). Es wird vor allem in australischen Spielbanken und auf Schiffscasinos gespielt (aus Wikipedia).

Die hier vorgestellte Variante von Mini Dice bietet zwei Wettmöglichkeiten:

- „Single die“: Man setzt eine der Zahlen 1, 2, 3 oder 4. Diese Zahlen repräsentieren die Augenzahlen 1 bis 4. Zeigen beide Würfel die gesetzte Augenzahl, so gewinnt man den dreifachen Einsatz. Zeigt nur ein Würfel die gesetzte Augenzahl, so gewinnt man den einfachen Einsatz;
- „Field“: Man setzt eine der Zahlen 10, 11 oder 12 und wettet darauf, dass die Augensumme der beiden Würfel diesen Wert annimmt. Im Gewinnfall erhält man den doppelten Einsatz.

Hat man kein Glück mit seiner Wette, ist der Einsatz verloren.

**Aufgabe:** Implementieren Sie (auf imperative Art und Weise) ein Consolen-Spiel, bei dem ein einzelner menschlicher Spieler Mini Dice spielen kann. Der Spieler hat anfangs 100 EUR Startkapital. Das Spiel besteht aus einzelnen Spielrunden. In jeder Runde wählt der Spieler eine der erlaubten Zahlen und gibt seinen Einsatz, d.h. einen Wert größer 0 und kleiner gleich seinem aktuellen Kapital an. Anschließend wird gewürfelt und im Fall, dass der Spieler gewinnt, der Gewinn ausgezahlt, oder im Fall, dass der Spieler verliert, der Einsatz eingezogen. Das Spiel ist beendet, wenn der Spieler nach einer Spielrunde kein Geld mehr oder mehr als 5.000 EUR hat.

Orientieren Sie sich bei den Ausgaben an folgendem Beispiel für einen Programmablauf (Benutzereingaben in <>):

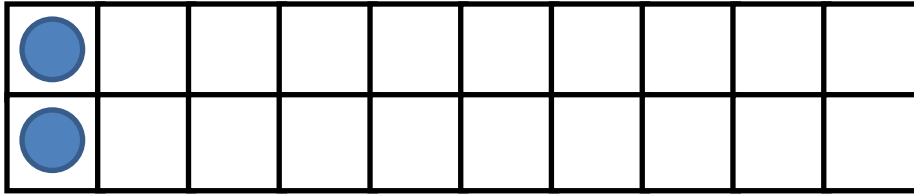
```
Runde 1:  
Kapital: 100 EUR  
Zahl: <2>  
Einsatz: <10>  
Würfelzahlen: 1 und 3  
Verloren
```

```
Runde 2:  
Kapital: 90  
Zahl: <12>  
Einsatz: <15>  
Würfelzahlen: 6 und 6  
Gewonnen (Field)
```

```
Runde 3:  
Kapital: 120 EUR  
...
```

### Aufgabe 44:

Wythoff ist ein Spiel für 2 Spieler. Auf einem Spielbrett, das aus 2 Reihen und N (hier N = 10) Spalten besteht, liegt anfangs auf den beiden linken Feldern jeweils ein Plättchen.



Die beiden Spieler sind abwechselnd an der Reihe. Es besteht Zugzwang. Ein gültiger Spielzug läuft folgendermaßen ab:

- Entweder bewegt der Spieler eines der beiden Plättchen um eine beliebige Anzahl an Feldern nach rechts
- Oder der Spieler verschiebt beide Plättchen um dieselbe Anzahl an Feldern nach rechts.

Verboten ist:

- Plättchen nach links zu ziehen
- Plättchen aus ihrer Reihe zu entfernen
- Plättchen außerhalb des Spielfeldes zu platzieren

Der Spieler, der es schafft, dass nach seinem Spielzug beide Plättchen auf den beiden rechten Feldern des Spielbrettes liegen, gewinnt. Hält sich ein Spieler nicht an die Regeln, verliert er sofort.

Aufgabe: Schreiben Sie ein Konsolen-Programm in Java, das es erlaubt, dass zwei Spieler gegeneinander das Wythoff-Spiel spielen. Orientieren Sie sich bei den Ausgaben an folgendem Beispiel für einen Programmablauf (Benutzereingaben in <>):

```
Spielbrett: 10 Spalten
Plaettchen 1: Spalte 1
Plaettchen 2: Spalte 1
```

```
Spieler 1 ist am Zug!
1 Plaettchen ziehen (j/n)?<j>
Oberes Plättchen ziehen (j/n)?<j>
Wie viele Felder?<3>
Plaettchen 1: Spalte 4
Plaettchen 2: Spalte 1
```

```
Spieler 2 ist am Zug!
1 Plaettchen ziehen (j/n)?<n>
Wie viele Felder?<4>
```

```
Plaettchen 1: Spalte 8
Plaettchen 2: Spalte 5
```

```
Spieler 1 ist am Zug!
1 Plaettchen ziehen (j/n)?<j>
Oberes Plättchen ziehen (j/n)?<n>
Wie viele Felder?<3>
Plaettchen 1: Spalte 8
Plaettchen 2: Spalte 8
```

```
Spieler 2 ist am Zug!
```

```
1 Plaettchen ziehen (j/n)?<n>  
Wie viele Felder?<2>  
Plaettchen 1: Spalte 10  
Plaettchen 2: Spalte 10
```

Glueckwunsch! Spieler 2 hat gewonnen!

Hinweis: Nach den Original-Regeln darf natürlich der erste Spieler nicht unmittelbar beide Plättchen ganz nach rechts bewegen. Um die Implementierung des Spiels aber nicht unnötig zu verkomplizieren, brauchen Sie das nicht zu berücksichtigen.

### **Aufgabe 45:**

Arzneimittel enthalten in der Regel einen kritischen Wirkstoff. Dieser reduziert sich jedoch kontinuierlich. Er darf allerdings eine gewisse Konzentration nicht unterschreiten, sonst verliert das Medikament an Wirkung und darf nicht mehr angewendet werden.

Beispiel: Gehen wir davon aus, dass ein Medikament bei der Herstellung 80mg des Wirkstoffs enthält, aber jeden Monat 0.25% des Wirkstoffs verliert. Minimal muss das Medikament jedoch immer 70mg des Wirkstoffs enthalten. Dann darf das Medikament nach 54 Monaten nicht mehr angewendet werden.

Schreiben Sie ein Programm, das zunächst die Anfangskonzentration (mg), die kritische Minimalkonzentration (mg) und die monatliche prozentuale Abnahme abfragt. Anschließend soll es berechnen, nach wie vielen Monaten das Medikament seine kritische Wirkstoffkonzentration unterschreitet, d.h. nach wie vielen Monaten das Medikament nicht mehr angewendet werden darf. Hinweis: Sie brauchen die Gültigkeit der Benutzereingaben nicht überprüfen.

Beispiel (Benutzereingaben in <>):

**Anfangskonzentration:** <80.0>

**Minimalkonzentration:** <70.0>

**monatliche prozentuale Abnahme:** <0.25>

**Das Medikament darf nach 54 Monaten nicht mehr angewendet werden.**

### **Aufgabe 46:**

*Groker* ist ein Spiel für zwei Spieler. Jeder Spieler hat einen Vorrat von unbegrenzt vielen Chips und einen Gewinnhaufen, der zu Beginn leer ist. Das Spiel wird in mehreren Runden gespielt. In jeder Runde setzen die Spieler jeweils für den Gegenspieler verdeckt eine beliebige Anzahl Chips – ihren Einsatz –, mindestens aber einen Chip. Aufgedeckt wird dann gleichzeitig. Unterscheidet sich die Zahl der gesetzten Chips um höchstens 10, dürfen beide Spieler ihren Einsatz komplett dem eigenen Gewinnhaufen hinzufügen. Ist die Differenz größer, darf lediglich derjenige mit dem kleineren Einsatz die Chips seinem Gewinnhaufen hinzufügen. Derjenige Spieler hat gewonnen, der in seinem Gewinnhaufen als erster mindestens 100 Chips mehr als sein Gegenspieler hat (aus Bundeswettbewerb Informatik).

Implementieren Sie ein Java-Programm, mit dem zwei menschliche Spieler gegeneinander an einer Konsole das Spiel *Groker* spielen können. Das Programm soll Benutzereingaben überprüfen und bei fehlerhaften Eingaben den Benutzer erneut zur Eingabe auffordern. Orientieren Sie sich, auch was die Ausgaben angeht, an folgendem Beispielablauf meiner Musterlösung (Benutzereingaben in Klammern <>):

```
Spieler 1! Bitte Anzahl eingeben (>0): <66>
Spieler 2! Bitte Anzahl eingeben (>0): <61>
Gewinnhaufen Spieler 1: 66
Gewinnhaufen Spieler 2: 61
Spieler 1! Bitte Anzahl eingeben (>0): <55>
Spieler 2! Bitte Anzahl eingeben (>0): <44>
Gewinnhaufen Spieler 1: 66
Gewinnhaufen Spieler 2: 105
Spieler 1! Bitte Anzahl eingeben (>0): <80>
Spieler 2! Bitte Anzahl eingeben (>0): <65>
Gewinnhaufen Spieler 1: 66
Gewinnhaufen Spieler 2: 170
Spielende! Spieler 2 hat gewonnen!
```

### Aufgabe 47:

In dieser Aufgabe geht es um das ASCII-Zeichen von Häkchen:



Schreiben Sie ein Java-Programm, das den Benutzer zunächst nach einem positiven int-Wert `groesse` fragt. Kontrollieren Sie die Eingabe. Das Programm soll anschließend in der entsprechenden Größe ein Häkchen der folgenden Form auf den Bildschirm ausgeben. Der Einsatz von Arrays ist bei dieser Aufgabe nicht erlaubt.

Beispiel: Größe = 1:

```
  /
```

Beispiel: Größe = 2:

```
  / \
 /   \
```

Beispiel: Größe = 3:

```
  /  \
 /    \
 /      \
```

## Aufgabe 48:

Schreiben Sie ein Programm, bei dem der Nutzer wiederholt aufgefordert wird, jeweils ein einzelnes char-Zeichen einzugeben, bis er irgendwann ein \$-Zeichen eingibt. Damit endet das Programm. Das Programm soll zählen, wie oft kleine (a, e, i, o, u) und wie oft große (A, E, I, O, U) Vokale eingegeben wurden und die jeweilige Anzahl am Ende ausgeben.

Im Folgenden wird ein möglicher Programmablauf skizziert (Eingaben in <>):

```
Zeichen ($=Ende): <a>
Zeichen ($=Ende): <G>
Zeichen ($=Ende): <U>
Zeichen ($=Ende): <o>
Zeichen ($=Ende): <F>
Zeichen ($=Ende): <&>
Zeichen ($=Ende): <>
Zeichen ($=Ende): <i>
Zeichen ($=Ende): <$>
Anzahl kleine Vokale = 3
Anzahl grosse Vokale = 1
```

## Aufgabe 49:

Wandeln Sie das folgende Programm um in ein äquivalentes Programm, das anstelle der for-Schleife eine while-Schleife und anstelle der switch-Anweisung eine if-Anweisung enthält. Dabei gilt: Die Programme sind äquivalent, wenn sie bei denselben Eingaben dieselben Ausgaben erzeugen.

```
String ausgabe = "";
boolean tag = false;
for (char zeichen = IO.readChar("Zeichen eingeben: ");
    zeichen != '#';
    zeichen = IO.readChar("Zeichen eingeben: "))
    {
    switch (zeichen) {
    case '<':
        tag = true;
        break;
    case '>':
        tag = false;
        break;
    default:
        if (!tag) {
            ausgabe += zeichen;
        }
    }
}
IO.println(ausgabe);
```

## Aufgabe 50:

Um eine zweistellige Zahl  $z$  mit 11 zu multiplizieren, kann man nach folgendem Algorithmus vorgehen: Sei  $a$  die erste Ziffer von  $z$  und  $b$  die zweite Ziffer von  $z$  und  $s$  die Summe von  $a$  und  $b$ . Ist  $s$  kleiner als 10, dann ist das Produkt von  $z$  mit 11 die Zahl mit der Ziffernfolge  $asb$ . Ist  $s$  größer gleich 10 mit der Ziffernfolge  $cd$ , dann ist das Produkt die Zahl mit der Ziffernfolge  $(a+1)db$ .



Beispiel 1:  $72 * 11$

$z = 72, a = 7, b = 2$

$s = 9$

Produkt = 792

Beispiel 2:  $85 * 11$

$z = 85, a = 8, b = 5$

$s = 13, c = 1, d = 3$

Produkt = 935

Aufgabe: Schreiben Sie ein Programm, das zunächst den Benutzer auffordert eine zwei-stellige Zahl einzugeben und dabei fehlerhafte Eingaben zurückweist. Anschließend soll das Programm nach obigem Algorithmus das Produkt der eingegebenen Zahl mit 11 berechnen und auf die Konsole ausgeben. Orientieren Sie sich dabei was Eingaben und Ausgaben angeht an folgendem beispielhaften Ablauf des Programms (Benutzereingaben in <>):

```
2-stellige Zahl: <123>
Fehler! 2-stellige Zahl: <85>
85 * 11 = 935
```

Randbedingung: Der einzige Datentyp, der zur Lösung dieser Aufgabe benutzt werden darf, ist `int`.

### Aufgabe 51:

Wandeln Sie das Programm um in ein äquivalentes Programm, das anstelle der `while`-Schleife eine `for`-Schleife und anstelle der `switch`-Anweisung eine `if`-Anweisung enthält. Dabei gilt: Die Programme sind äquivalent, wenn sie bei derselben Eingabe dieselben Ausgaben erzeugen. Hinweis: Es gibt mehrere korrekte Lösungen!

```
public class Umwandlung {

    public static void main(String[] args) {
        int zahl = 0;
        int ende = IO.readInt("Endezahl: ");
        int inkr = 1;
        while (zahl < ende) {
            switch (zahl % 2) {
                case 0:
                    IO.println(zahl);
            default:
                IO.println(zahl * zahl);
            }
        }
    }
}
```

```

        zahl = zahl + inkr;
        inkr++;
    }
}

```

### Aufgabe 52:

In dieser Aufgabe geht es um das ASCII-Zeichnen von „Klammernpaarhaufen“. Schreiben Sie ein Java-Programm, das den Benutzer zunächst nach einem positiven int-Wert `zeilen` fragt. Kontrollieren Sie die Eingabe. Das Programm soll anschließend in der folgenden Form einen Haufen von Klammernpaaren auf den Bildschirm ausgeben. Der Einsatz von Arrays ist bei dieser Aufgabe nicht erlaubt.

Beispiel: Zeilen = 1:

()

Beispiel: Zeilen = 2:

()()

()

Beispiel: Zeilen = 3:

()()()

()()

()

Beispiel: Zeilen = 6:

()()()()()()

()()()()()

()()()()

()()()

()()

()

### Aufgabe 53:

*Spiel 21* ist ein Spiel für  $N$  Spieler ( $N \geq 2$ ). Die Spieler spielen reihum. Es besteht Zugzwang. Es beginnt Spieler 1. Der Spieler, der an der Reihe ist, erhöht in seinem Spielzug eine aktuelle Zahl (Anfangswert ist 0) jeweils um 1, 2 oder 3. Der Spieler,

der als erstes 21 oder mehr erreicht, verliert das Spiel. Die anderen Spieler haben alle gewonnen.

Implementieren Sie ein imperatives Java-Programm, mit dem N menschliche Spieler gegeneinander an einer Konsole das Spiel *Spiel 21* spielen können. Das Programm soll Benutzereingaben überprüfen und bei fehlerhaften Eingaben den Benutzer erneut zur Eingabe auffordern. Orientieren Sie sich, auch was die Ausgaben angeht, an folgendem Beispielablauf meiner Musterlösung (Benutzereingaben in Klammern <>):

```
Anzahl Spieler (> 1): <3>
Spieler 1! Zahl eingeben (1, 2, 3): <2>
Aktuelle Zahl: 2
Spieler 2! Zahl eingeben (1, 2, 3): <4>
Fehlerhafte Eingabe! Spieler 2! Zahl eingeben (1, 2, 3): <1>
Aktuelle Zahl: 3
Spieler 3! Zahl eingeben (1, 2, 3): <3>
Aktuelle Zahl: 6
Spieler 1! Zahl eingeben (1, 2, 3): <3>
Aktuelle Zahl: 9
Spieler 2! Zahl eingeben (1, 2, 3): <2>
Aktuelle Zahl: 11
Spieler 3! Zahl eingeben (1, 2, 3): <3>
Aktuelle Zahl: 14
Spieler 1! Zahl eingeben (1, 2, 3): <2>
Aktuelle Zahl: 16
Spieler 2! Zahl eingeben (1, 2, 3): <3>
Aktuelle Zahl: 19
Spieler 3! Zahl eingeben (1, 2, 3): <1>
Aktuelle Zahl: 20
Spieler 1! Zahl eingeben (1, 2, 3): <1>
Aktuelle Zahl: 21
Ende! Spieler 1 hat verloren!
```

