

Programmierkurs Java

Dr. Dietrich Boles

Aufgaben zu UE15-FunktionenParameter (Stand 20.12.2011)

Aufgabe 1:

Die Ulam-Zahlenfolge beginnt mit einer beliebigen natürlichen Zahl a . Ist a gerade, so erhält man die nächste Zahl b durch Halbierung; ist a ungerade, so erhält man b dadurch, dass man a verdreifacht und 1 hinzuaddiert. Die Folge endet, wenn ein Glied den Wert 1 hat.

Es stellt sich die Frage, ob die Ulam-Folge für alle natürlichen Zahlen als Startwert endlich ist. Man vermutet, dass dies tatsächlich so ist, da bis heute kein Startwert bekannt ist, für den die Folge nicht endlich ist. Ein Beweis ist den Mathematikern aber bisher noch nicht gelungen. Die Ulam-Folge gehört deshalb zu den so genannten "offenen Problemen".

Definieren Sie zunächst eine Funktion *ulamWert*, die als Parameter einen int-Wert n übergeben bekommt und als Ergebnis den nächsten Wert der Ulam-Folge nach n liefert.

Schreiben Sie dann ein Java-Programm, das zunächst den Benutzer auffordert, einen int-Wert *zahl* einzugeben. Anschließend soll die komplette Ulam-Folge zu *zahl* auf den Bildschirm ausgegeben werden. Nutzen Sie dabei die von Ihnen definierte Funktion *ulamWert*.

Beispiel:

Eingabe: 3 Ausgabe: 3 10 5 16 8 4 2 1

Aufgabe 2:

Implementieren Sie folgende Funktionen:

- `int getBetrag(int zahl)` ; liefert den Betrag der übergebenen Zahl; Bsp.: `getBetrag(-3) -> 3`
- `int getAnzahlZiffern(int zahl)` ; liefert die Anzahl an Ziffern der übergebenen Zahl; Bsp.: `getAnzahlZiffern(2542) -> 4` bzw. `getAnzahlZiffern(-342) -> 3`
- `int getZiffernWert(int zahl, int stelle)` ; liefert den Wert der Ziffer der übergebenen Zahl an der Stelle `stelle`; die Stellen werden dabei von rechts nach links angegeben und beginnen bei 0; Sie können davon ausgehen, dass gilt: $0 \leq \text{stelle} < \text{getAnzahlZiffern}(\text{zahl})$; Bsp.: `getZiffernWert(27381, 3) -> 7` bzw. `getZiffernWert(-27381, 0) -> 1`
- `int ersetzeZiffer(int zahl, int stelle, int wert)` ; ersetzt die Ziffer der übergebenen Zahl an der Stelle `stelle` durch den übergebenen `wert` und liefert die neue Zahl; die Stellen werden dabei von rechts nach links angegeben und beginnen bei 0; Sie können davon ausgehen, dass gilt: $0 \leq \text{stelle} < \text{getAnzahlZiffern}(\text{zahl})$ und $0 \leq \text{wert} \leq 9$; Bsp.: `ersetzeZiffer(24135, 3, 7) -> 27135` bzw. `ersetzeZiffer(-12345, 0, 6) -> -12346`

Schreiben Sie ein kleines Programm, mit dem Sie die Funktionen testen können.

Aufgabe 3:

Implementieren Sie in Java folgende Prozeduren/Funktionen! Achten Sie auf Randfälle und nicht korrekte Parameterübergaben! Überprüfen Sie aber zunächst für jede Funktion, ob sie überhaupt mit den (bisher bekannten) Konzepten in Java implementiert werden kann und wenn nicht, begründen Sie, wieso nicht!

- (1) eine Funktion, die testet, ob eine als Parameter übergebene natürliche Zahl eine Fibonacci-Zahl ist oder nicht (Beispiel: `f(8) == true`),
- (2) eine Prozedur, die die Werte zweier als Parameter übergebener `double`-Variablen vertauscht (Beispiel: `double a=2.0; double b=5.9; f(a, b);` Ergebnis: `a==5.9; b==2.0;`),
- (3) eine Funktion, die einen übergebenen `double`-Wert rundet und als `int`-Wert zurückliefert (Beispiel: `f(-2.6) == 3`)
- (4) eine Funktion, die die nächst kleinere Primzahl einer als Parameter übergebenen natürlichen Zahl (größer als 2) liefert (Beispiel: `f(11) == 7`),
- (5) eine Funktion, die als Parameter einen `int`-Wert übergeben bekommt und die überprüft, ob die Ziffer 7 in dem `int`-Wert vorkommt (Beispiel: `f(-2578) == true`),
- (6) eine Funktion, die als ersten Parameter eine Funktion `g:char->int` und als zweiten Parameter einen `char`-Wert `zeichen` übergeben bekommt und die als Ergebnis den Wert `g(zeichen)` liefert (Beispiel: `public static int pos(char zeichen) {return zeichen - 'a';}` und `f(pos, 'b') == 2`),
- (7) eine Funktion, die als Parameter einen `int`-Wert `n` übergeben bekommt und die als Ergebnis die Summe der Zahlen zwischen 0 und `n` zurückliefert; ist

der Wert des übergebenen Parameters jedoch kleiner als 0, soll die Funktion den Wert false liefern (Beispiel: $f(4) = 10$, $f(-2) == \text{false}$)

- (8) eine Funktion, die als ersten Parameter einen int-Wert n und daraufhin n float-Parameter übergeben bekommt, deren Summe geliefert werden soll (Beispiel: $f(3, 1.1f, 2.2f, 3.3f) == 6.6f$).
- (9) eine Funktion, die als ersten Parameter einen float-Wert x (zwischen 0 und 1000) und als zweiten Parameter einen positiven int-Wert n (zwischen 1 und 5) übergeben bekommt. Die Funktion soll den Wert x auf n Nachkommastellen runden (Beispiel: $f(2.2576F, 3) == 2.258F$)
- (10) eine Funktion, die die Summe zweier Uhrzeiten als Ergebnis liefert; Uhrzeiten werden dabei als float-Werte realisiert, wobei die Vorkommastellen die Stunden und die Nachkommastellen die Minuten darstellen (Beispiel: $f(22.13f, 3.48f) == 2.01f$)
- (11) eine Funktion, die die Summe und die Differenz zweier als Parameter übergebener int-Werte zurückliefert (Beispiel: $f(4, 3) = (7, 1)$)

Schreiben Sie ein Programm, das einem Benutzer eine Auswahl zur Ausführung der implementierbaren Funktionen anbietet, anschließend jeweils passende Werte für die aktuellen Parameter einliest, die ausgewählte Funktion aufruft und ein Ergebnis auf dem Bildschirm ausgibt. Achten Sie darauf, dass die Funktionen nur mit zulässigen Werten aufgerufen werden.

Aufgabe 4:

Schauen Sie sich folgendes Programm an:

```
public class Fragezeichen {
public static void main(String[] args) {
int x = IO.readInt();
int m1 = 2147483647;
int m2 = -2147483648;
int s = 0;
int zahl = 0;
for (int i=0; i<x; i++) {
zahl = IO.readInt("Z: ");
m1 = m(m1, zahl);
m2 = m2 > zahl ? m2 : zahl;
s = s + zahl;
}
if (x > 0) {
IO.println("K: " + m1);
IO.println("G: " + m2);
IO.println("M: " + ((double)s / (double)x));
}
}
public static int m(int zahl1, int zahl2) {
return zahl1 < zahl2 ? zahl1 : zahl2;
}
}
```

Was tut dieses Programm? Wandeln Sie das Programm um in ein "schönes" Programm:

- Nutzen Sie Leerzeichen, Zeilenumbruchzeilen, etc. zur übersichtlicheren Formatierung!
- Wählen Sie aussagekräftige Bezeichner!
- Definieren Sie Funktionen für eine bessere Übersichtlichkeit!
- Integrieren Sie Kommentare, wo notwendig!
- Wählen Sie aussagekräftige Hinweise an den Benutzer!
- Schränken Sie den Gültigkeitsbereich von Variablen so eng wie möglich ein!

Aufgabe 5:

Schreiben Sie ein Programm, das die Ziffern einer eingegebenen positiven Zahl in aufsteigender Reihenfolge sortiert und diese ausgibt. 0en fallen weg. Sie dürfen dabei keine Arrays oder Rekursion benutzen!

Beispiel:

Eingabe: 1423 Ausgabe: 1234
Eingabe: 1442624 Ausgabe: 1224446
Eingabe: 100132 Ausgabe: 1123

Aufgabe 6:

Schreiben Sie ein Java-Programm, das einen int-Wert zahl mit $0 < \text{zahl} < 10000$ einliest, ihre Quersumme berechnet und die durchgeführte Berechnung sowie den Wert der Quersumme wie nachfolgend beispielhaft dargestellt ausgibt:

Ganze Zahl zwischen 1 und 9999 eingeben: 2546
Die Quersumme ergibt sich zu: $2 + 5 + 4 + 6 = 17$

Schreiben und benutzen Sie dabei eine Funktion `int liefereZiffer(int zahl, int stelle)`, die von einer übergebenen Zahl den Ziffernwert an der Stelle "stelle" (von hinten) liefert; Beispiel: `liefereZiffer(2548, 3)` soll den Wert "5" liefern.

Aufgabe 7:

Zwei verschiedene natürliche Zahlen a und b heißen befreundet, wenn die Summe der (von a verschiedenen) Teiler von a gleich b ist und die Summe der (von b verschiedenen) Teiler von b gleich a ist.

Ein Beispiel für ein solches befreundetes Zahlenpaar ist $(a,b) = (220,284)$, denn a = 220 hat die Teiler 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110 und es gilt

$$1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284 = b.$$

Weiterhin hat $b = 284$ die Teiler 1, 2, 4, 71, 142 und es gilt

$$1 + 2 + 4 + 71 + 142 = 220 = a.$$

Schreiben Sie ein Java-Programm, das in einer Schleife jeweils zwei Zahlen einliest und entscheidet, ob diese miteinander befreundet sind. Das Programm soll abbrechen, falls zwei nicht-befreundete Zahlen eingegeben wurden. Schreiben und verwenden Sie dabei eine `int`-Funktion `teilersumme`, die von der ihr übergebenen Zahl die Teilersumme (ohne die Zahl selbst) zurückliefert.

Der Programmablauf könnte in etwa wie folgt aussehen:

```
Erste Zahl: 220
Zweite Zahl: 284
Die beiden Zahlen sind miteinander befreundet!
Erste Zahl: 10744
Zweite Zahl: 10856
Die beiden Zahlen sind miteinander befreundet!
Erste Zahl: 23
Zweite Zahl: 22
Die beiden Zahlen sind nicht miteinander befreundet!
```

Aufgabe 8:

Eine Natürliche Zahl heißt *potent*, wenn sie sich als Summe von Potenzen (≥ 1) ihrer Ziffern darstellen lässt. Beispielsweise ist 24 wegen $2^3 + 4^2 = 24$ *potent*. Schreiben Sie ein Java-Programm, das alle zweistelligen *potenten* Zahlen ausgibt.

Aufgabe 9:

Keht man bei einer Natürlichen Zahl die Reihenfolge der Ziffern um, so erhält man ihre Spiegelzahl. Eine Zahl heißt *Palindrom*, wenn sie mit ihrer Spiegelzahl übereinstimmt (Bsp.: 15851).

- (a) Schreiben Sie ein Java-Programm, das Zahlen einliest und jeweils überprüft, ob es sich bei der Zahl um ein Palindrom handelt. Definieren Sie dazu eine geeignete Funktion!
- (b) Folgendermaßen lassen sich Palindrome erzeugen: man addiert zu einer gegebenen Natürlichen Zahl ihre Spiegelzahl, zur Summe wieder deren Spiegelzahl usw.; dies wird solange wiederholt, bis ein Palindrom entstanden ist (Beispiel: $178 + 871 + 9401 + 05401 = 15851$). Achtung: Bei bestimmten Zahlen bricht der Algorithmus (wahrscheinlich) niemals ab; Bsp. 196).

Schreiben Sie ein Java-Programm, das nach der Eingabe einer Natürlichen Zahl versucht, daraus gemäß des beschriebenen Algorithmus ein Palindrom zu erzeugen.

Aufgabe 10:

Das Spiel „27“ ist ein Zahlenspiel für 2 Spieler. Bei dem Spiel geht es darum, dass zwei Spieler abwechselnd entweder die Zahl 1 oder die Zahl 2 auswählen, deren Wert von einer Spielzahl abgezogen wird, deren Anfangswert 27 beträgt. Wer als erster die 0 oder eine kleinere Zahl erreicht, hat gewonnen. Es beginnt Spieler A.

Aufgabe: Schreiben Sie ein Java-Programm, bei dem zwei Menschen gegeneinander das Spiel „27“ spielen können. Im Folgenden wird ein möglicher Spielablauf skizziert:

```
Spielzahl = 27
Spieler A, 1 oder 2 wählen: -> 2
Spielzahl = 25
Spieler B, 1 oder 2 wählen: -> 1
Spielzahl = 24
Spieler A, 1 oder 2 wählen:
. . .
Spielzahl = 4
Spieler A, 1 oder 2 wählen: -> 2
Spielzahl = 2
Spieler B, 1 oder 2 wählen: -> 2
Spieler B hat gewonnen!
```

Alternative: Zum Spiel „27“ gibt es eine Gewinnformel sprich Gewinnstrategie. Wenn Spieler B diese kennt und entsprechend spielt, gewinnt er immer. Versuchen Sie diese Gewinnformel herzuleiten. Wenn Sie dies schaffen, können Sie das zu entwickelnde Programm auch derart abändern, dass Spieler A ein Mensch ist und Spieler B der Computer ist, der diese Strategie anwendet und daher immer gewinnt. Ein möglicher Spielablauf sähe dann so aus:

```
Spielzahl = 27
Spieler A, 1 oder 2 wählen: -> 2
Spielzahl = 25
Computer wählt 1
Spielzahl = 24
Spieler A, 1 oder 2 wählen:
. . .
Spielzahl = 3
Spieler A, 1 oder 2 wählen: -> 2
Spielzahl = 1
Computer wählt 1
Spieler B hat gewonnen!
```

Aufgabe 11:

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-Programm, 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:

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

```

      |      |
    +-+    +-+
      |      |
    +-+    +-+

```

Hinweis: Sie dürfen zur Lösung keine globalen Variablen benutzen, sondern müssen stattdessen u. U. Parameter verwenden.

Aufgabe 12:

Sie sind Fußballfan und Sie möchten sich selbst mal eine Freude machen, indem Sie ein ASCII-Fußballstadion auf den Bildschirm Ihres Computers malen.

Aufgabe: Schreiben Sie ein Java-Programm, das den Benutzer zunächst auffordert, eine Zahl größer gleich 1 einzugeben und das anschließend ein Fußballstadion mit entsprechend hohen Tribünen in der folgenden Form auf den Bildschirm ausgibt:

Höhe = 1:

```

+-+ +-+
 | |
+-+

```

Höhe = 3:

```

+-+          +-+
 |           |
+-+          +-+
 |           |
+-+ +-+
 | |
+-+

```

Hinweis: Sie dürfen zur Lösung keine globalen Variablen benutzen, sondern müssen stattdessen u. U. Parameter verwenden.

Aufgabe 13:

Bei dieser Aufgabe geht es darum, zu berechnen, ob sich in einem rechtwinkligen Koordinatensystem zwei Kreise schneiden oder nicht.

Ein Benutzer des Programms soll zunächst insgesamt 6 double-Werte eingeben können:

- x-Koordinate des ersten Kreises
- Y-Koordinate des ersten Kreises
- Radius des ersten Kreises (positiver Wert!)
- x-Koordinate des zweiten Kreises
- y-Koordinate des zweiten Kreises
- Radius des zweiten Kreises (positiver Wert!)

Anschließend soll das Programm berechnen, ob sich die beiden Kreise schneiden oder nicht. Im ersten Fall soll die Ausgabe "Kreise schneiden sich!", im zweiten Fall die Ausgabe "Kreise schneiden sich nicht!" erfolgen.

Beispielablauf:

```
x-Koordinate Kreis 1 eingeben: 1.0
Y-Koordinate Kreis 1 eingeben: 1.0
Radius Kreis 1 eingeben: 1.0
x-Koordinate Kreis 2 eingeben: 2.0
Y-Koordinate Kreis 2 eingeben: 2.0
Radius Kreis 2 eingeben: 1.0
Kreise schneiden sich!
```

Zum Berechnen der Wurzel eines double-Wertes stellt Java übrigens folgende Funktion zur Verfügung: `public static double Math.sqrt(double zahl)`. Die können Sie nutzen.

Aufgabe 14:

Zu einer gegebenen natürlichen Zahl z mit n -Ziffern ist ihre sogenannte „Umlaufzahl“ u folgendermaßen definiert:

- Die erste Ziffer von z ist auch die erste Ziffer von u .
- Die $n+1$ -te Ziffer von u ergibt sich folgendermaßen: Laufe in der Zahl z von der n -ten Ziffer aus gesehen so viele Ziffern nach rechts (gegebenenfalls wieder vorne anfangen!) wie der Wert der n -ten Ziffer angibt. Die erreichte Ziffer in z ist die $n+1$ -te Ziffer von u .
- Sobald eine Ziffer in z zum zweiten Mal erreicht wird, ist der Algorithmus beendet.

Beispiel: ($z = 6354$)

```
6 3 5 4      u: 6
6 3 5 4      u: 65
6 3 5 4      u: 654
6 3 5 4      u: 6544
```

Noch ein Beispiel: ($z = 1213013$)

```
1 2 1 3 0 1 3      u: 1
1 2 1 3 0 1 3      u: 12
1 2 1 3 0 1 3      u: 123
1 2 1 3 0 1 3      u: 1233
1 2 1 3 0 1 3      u: 12331
1 2 1 3 0 1 3      u: 123313
```

Aufgabe: Schreiben Sie zunächst eine int-Funktion *umlaufzahl*, die von einer als Parameter übergebenen Zahl deren Umlaufzahl berechnet. Schreiben Sie dann ein Java-Programm, das eine natürliche Zahl z einliest, mit Hilfe der Funktion *umlaufzahl* die zugehörige Umlaufzahl berechnet und diese ausgibt.

Hinweis: Implementieren Sie weitere Hilfsfunktionen, die bspw. die Länge einer Zahl oder die Ziffer an der n-ten Stelle einer Zahl berechnen (siehe auch Aufgabe 2).

Aufgabe 15:

Sie sollen das bekannte Spiel „Schnick-Schnack-Schnuck“ so implementieren, dass es der Computer gegen einen Menschen spielen kann.

Regeln: Bei Schnick-Schnack-Schnuck werden mehrere Spielrunden absolviert. In jeder Spielrunde wählen die beiden Spieler gleichzeitig eines der folgenden Symbole: Brunnen, Schere, Stein oder Papier. Dabei gilt:

- Brunnen gewinnt gegen Schere
- Brunnen gewinnt gegen Stein
- Brunnen verliert gegen Papier
- Schere verliert gegen Stein
- Schere gewinnt gegen Papier
- Stein verliert gegen Papier

Der Sieger erhält jeweils einen Punkt. Wählen die beiden Spieler dasselbe Symbol, ist das ein Unentschieden und keiner der beiden Spieler erhält einen Punkt.

Ein Spiel endet, wenn ein Spieler insgesamt 10 Punkte erreicht. Dieser Spieler ist Sieger.

Ablauf: In jeder Spielrunde generiert der Computer per Zufall eines der vier Symbole. Anschließend fordert er den menschlichen Spieler zur Eingabe eines Symbols auf. Er ermittelt das Ergebnis und gibt es auf den Bildschirm aus.

Beispiel:

Spielrunde 1:

Symbol eingeben (Brunnen, Schere, Stein, Papier): -> Schere

Computer: Brunnen; Mensch: Schere -> Computer gewinnt

Spielstand: Computer = 1; Mensch = 0

...

Spielrunde 21:

Symbol eingeben (Brunnen, Schere, Stein, Papier): -> Stein

Computer: Brunnen; Mensch: Stein -> Computer gewinnt

Spielstand: Computer = 10; Mensch = 5

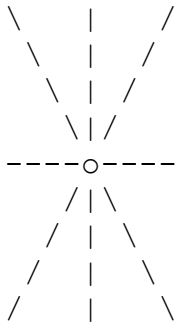
Spielende: Computer hat gewonnen

Aufgabe 16:

Bald ist Winter, und hoffentlich wird bald der erste Schnee fallen. Sie sollen diesem Tag vorgereifen 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 18:

Bei dieser Aufgabe sollen Sie einen Mathetrainer zum Üben des Multiplizierens und Dividierens implementieren.

Der Mathetrainer generiert dabei jeweils zufällig zwei int-Werte zwischen 0 und 9 sowie zufällig entweder den Multiplikations- oder den Divisionsoperator. Er präsentiert die Aufgabe und fordert den Benutzer auf, das Ergebnis einzugeben. Gibt dieser das falsche Ergebnis ein, teilt ihm der Mathetrainer das korrekte Ergebnis mit. Gibt der Benutzer das korrekte Ergebnis ein, wird er gelobt. In beiden Fällen wird anschließend die Gesamtanzahl der bisherigen korrekten Antworten ausgegeben. Das Ganze wird wiederholt, bis der Benutzer 10 korrekte Antworten gegeben hat.

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

```
Start des Mathetrainers
8 / 1 = <8>
Richtig!
Korrekte Antworten: 1
9 / 2 = <4>
Richtig!
Korrekte Antworten: 2
6 * 1 = <4>
Leider falsch! Korrektes Ergebnis ist 6
Korrekte Antworten: 2
0 / 8 = <0>
Richtig!
Korrekte Antworten: 3

...

3 * 2 = <6>
```

Richtig!
Korrekte Antworten: 9
0 * 5 = <0>
Richtig!
Korrekte Antworten: 10
Ende des Mathetrainers

Aufgabe 19:

Beim Spiel *Mäxchen* würfelt ein Spieler zwei Würfel. Würfelt er eine 1 und eine 2, ein so genanntes *Mäxchen*, bekommt er 1000 Punkte. Würfelt er einen Pasch, d.h. zwei gleiche Zahlen, erhält er das Hundertfache der entsprechenden Zahl (also bspw. 400 bei zwei 4en) als Punkte, ansonsten ist die Punktzahl $10 * \text{höhere Augenzahl} + \text{niedrigere Augenzahl}$. Der Wurf 3, 5 hat also bspw. den Wert 53.

Implementieren Sie eine Funktion `maexchen`, die zwei gewürfelte Zahlen als Parameter übergeben bekommt und die erzielten Punkte als Funktionswert liefert.

Schreiben Sie dann ein Programm, in dem die Funktion `maexchen` mit jeweils zwei zufällig erzeugten Zahlen zwischen 1 und 6 so oft aufgerufen wird, bis 100.000 Punkte erreicht sind. Anschließend soll der prozentuale Anteil der Mäxchen-Würfe als `double`-Wert ausgegeben werden.

Aufgabe 20:

Bei dieser Aufgabe geht es darum, ein Programm zu entwickeln, das es erlaubt, die Reaktionszeit der Nutzer zu messen. Der Programmablauf sieht folgendermaßen aus:

- Zunächst wird „Achtung: Start“ ausgegeben.
- Dann werden zwischen 5 und 10 Runden absolviert. Die genaue Zahl der Runden wird per Zufall bestimmt. (`Util.getRandomNumber(<max>)`)
- In jeder Runde wird folgendes gemacht:
 - Es wird zunächst zwischen 2 und 5 Sekunden gewartet. (`Util.wait(<sec>)`). Die genaue Wartezeit wird per Zufall bestimmt.
 - Dann bestimmt das Programm per Zufall einen Kleinbuchstabe (,a' – ,z') und gibt ihn auf dem Bildschirm aus. (Alternative: Zahl)
 - Der Benutzer muss jetzt versuchen, so schnell wie möglich den Buchstaben einzugeben (inkl. <Enter>). Die Zeit zwischen Ein- und Ausgabe wird gemessen (`Util.getMilliseconds()`)
 - Falls der ausgegebene Kleinbuchstabe und das eingegebene Zeichen gleich sind, wird die Zeit zwischen Ausgabe des Buchstabens und Eingabe des Nutzers verarbeitet (die Reaktionszeit). Ansonsten merkt sich das Programm einen Fehler.

- Sind alle Runden absolviert, gibt das Programm „Geschafft: Ende“ aus. Weiterhin werden ausgegeben:
 - Anzahl an Fehlversuchen
 - Mittelwert der Reaktionszeiten
 - Langsamster Versuch
 - Schnellster Versuch

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

```
Achtung: Start!
b
<b>
u
<u>
v
<d>
m
<m>
a
<a>
Geschafft: Ende!
Fehlversuche: 1 von 5
Reaktionszeit-Mittelwert: 2.5748 Sekunden
Langsamster Versuch: 5.813 Sekunden
Schnellster Versuch: 1.218 Sekunden
```

Aufgabe 21:

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

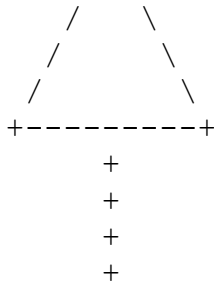
Aufgabe: Schreiben Sie ein Java-Programm, das den Benutzer zunächst auffordert, eine Zahl („Höhe“) größer gleich 1 einzugeben und das anschließend einen entsprechend hohen Weihnachtsbaum der folgenden Form auf den Bildschirm ausgibt:

Höhe = 1:

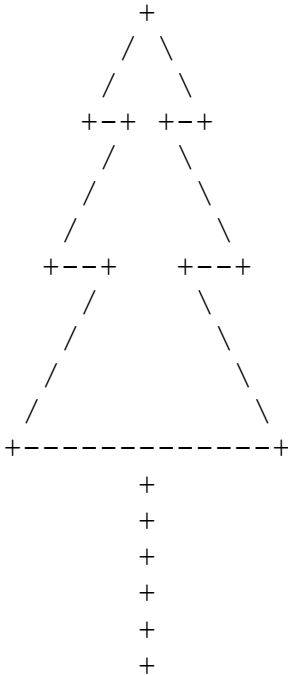
```
  +
 / \
 /   \
+-----+
  +
  +
```

Höhe = 2:

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



Höhe = 3:



Aufgabe 22:

„Glückspiel“ ist ein Spiel für zwei Spieler. Es wird über mehrere Spielrunden gespielt bis ein Spieler 5 Punkte erreicht hat. Dieser Spieler hat dann gewonnen. Eine Spielrunde läuft wie folgt ab:

- Der Computer generiert eine geheime Zufallszahl zwischen 0 und 9.
- Die Spieler geben nacheinander eine Zahl ein. Dabei darf der zweite Spieler nicht dieselbe Zahl eingeben wie der erste Spieler.
- Für jeden Spieler wird die Differenz zwischen seiner Zahl und der generierten Zufallszahl berechnet. Der Spieler mit der betragsmäßig niedrigeren Differenz erhält einen Punkt. Ist die Differenz bei beiden Spielern gleich, erhält keiner der Spieler einen Punkt.

Damit es fair zugeht, wechselt in jeder Spielrunde der Spieler, der als erster seine Zahl eingibt. In der ersten Spielrunde beginnt Spieler 1, in der zweiten Spieler 2, usw.

Implementieren Sie (auf imperative Art und Weise) das Spiel „Glückspiel“ derart, dass es zwei menschliche Spieler gegeneinander spielen können. Behandeln Sie fehlerhafte Benutzereingaben adäquat. Orientieren Sie sich an dem folgenden beispielhaften Programmablauf (Eingaben stehen in <>):

Ich habe eine Zufallszahl zwischen 0 und 9 generiert!
Spieler 1, deine Zahl: <5>
Spieler 2, deine Zahl: <3>
Zufallszahl war die 2
Spieler 2 gewinnt die Runde
Spielstand: Spieler 1 hat 0 Punkte; Spieler 2 hat 1 Punkte.

Ich habe eine Zufallszahl zwischen 0 und 9 generiert!
Spieler 2, deine Zahl: <2>
Spieler 1, deine Zahl: <7>
Zufallszahl war die 7
Spieler 1 gewinnt die Runde
Spielstand: Spieler 1 hat 1 Punkte; Spieler 2 hat 1 Punkte.

Ich habe eine Zufallszahl zwischen 0 und 9 generiert!
Spieler 1, deine Zahl: <5>
Spieler 2, deine Zahl: <6>
Zufallszahl war die 5
Spieler 1 gewinnt die Runde
Spielstand: Spieler 1 hat 2 Punkte; Spieler 2 hat 1 Punkte.

...
Ich habe eine Zufallszahl zwischen 0 und 9 generiert!
Spieler 1, deine Zahl: <4>
Spieler 2, deine Zahl: <5>
Zufallszahl war die 7
Spieler 2 gewinnt die Runde
Spielstand: Spieler 1 hat 2 Punkte; Spieler 2 hat 5 Punkte.

Spieler 2 hat gewonnen

Aufgabe 23:

Ein Marathonlauf hat eine Länge von 42,195 km. Für einen Marathonläufer, der eine bestimmte Zielzeit anvisiert, ist es wichtig zu wissen, mit welchen Zeiten er die einzelnen Kilometer laufen muss, um mindestens seine Zielzeit zu erreichen. Daran kann er einschätzen, ob er zu schnell oder zu langsam ist. Er will dabei natürlich möglichst gleichmäßig schnell laufen. Helfen Sie ihm beim Berechnen der einzelnen Kilometerzeiten.

Aufgabe: Implementieren Sie ein Programm, das den Benutzer zunächst nach den anvisierten Stunden und anschließend nach den anvisierten Minuten beim anstehenden Marathon fragt. Anschließend soll das Programm berechnen und ausgeben, bei welchen Zeiten (Stunden:Minuten:Sekunden) er die einzelnen Kilometer passieren muss, um bei einem möglichst gleichmäßigen Tempo möglichst exakt aber mindestens seine Zielzeit zu erreichen. Weiterhin soll das Programm die entsprechende Halbmarathon- und Marathonzeit berechnen und ausgeben.

Achtung: Definieren Sie die Länge des Marathonlaufes als eine Konstante und setzen Sie zur Berechnung und Ausgabe der KM-Zeiten eine Schleife ein. Ihr Programm soll auch korrekt arbeiten, wenn die Länge des Marathonlaufes mal geändert werden sollte.

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

```

Anvisierte Stunden: <3>
Anvisierte Minuten: <30>
1 KM = 00:04:58
2 KM = 00:09:57
3 KM = 00:14:55
...
13 KM = 01:04:41
14 KM = 01:09:40
15 KM = 01:14:39
...
24 KM = 01:59:26
25 KM = 02:04:25
26 KM = 02:09:23
27 KM = 02:14:22
...
37 KM = 03:04:08
38 KM = 03:09:07
39 KM = 03:14:05
40 KM = 03:19:04
41 KM = 03:24:03
42 KM = 03:29:01
Halbmarathon = 01:44:59
Marathon = 03:29:59

```

Aufgabe 24:

Schreiben Sie ein Java-Programm, das den Benutzer zunächst zur Eingabe eines positiven `int`-Wertes `n` auffordert und anschließend ein „Auto“ in der im Folgenden skizzierten Form auf den Bildschirm ausgibt. Überlegen Sie anhand der Beispiele selbst, wo und in welcher Form sich der Wert von `n` auf die Größe und Gestalt des Autos auswirkt.

Beispiel (`n = 1`):

```

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

```

Beispiel (`n = 2`):

```

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

```

Beispiel (`n = 3`):

```

      +-----+
     /         \

```

