

Ergänzung zum Buch  
"Java ist eine Sprache"  
von Ulrich Grude,  
Vieweg-Verlag 2005

Diese Übungen kann man z. B. in kleinen Gruppen unter Anleitung eines Betreuers oder auch allein (im Selbststudium) bearbeiten.

Hinweise auf Fehler, Verbesserungsvorschläge oder Berichte über Erfahrungen mit diesen Übungen sind jederzeit willkommen, am liebsten per e-mail an die Adresse [grude@tfh-berlin.de](mailto:grude@tfh-berlin.de).

**Inhaltsverzeichnis**

Übung if:.....	3
Lösung if:.....	5
Übung ÜberAus 1:.....	7
Lösung ÜberAus 1:.....	8
Übung Schleifen (ausführen) 1:.....	9
Lösung Schleifen (ausführen) 1:.....	10
Übung Schleifen (programmieren) 2:.....	11
Lösung Schleifen (programmieren) 2:.....	12
Übung Methoden 1:.....	14
Lösung Methoden 1:.....	15
Übung Reihungen 1:.....	18
Lösung Reihungen 1:.....	19
Übung Reihungen 2:.....	21
Lösung Reihungen 2:.....	22
Übung Reihungen 3:.....	25
Lösung Reihungen 3:.....	26
Übung Bojen (ausführliche und vereinfachte) 1:.....	27
Lösung Bojen (ausführliche und vereinfachte) 1:.....	28
Übung Bojen (von Reihungen) 2:.....	29
Lösung Bojen (von Reihungen) 2:.....	30
Übung Klassen 1:.....	32
Lösung Klassen 1:.....	34
Übung Deutsch 1:.....	35
Lösung Deutsch 1:.....	36
Übung Klassen 2:.....	37
Lösung Klassen 2:.....	40
Übung Klassen 3:.....	42
Lösung Klassen 3:.....	43
Übung Strings 1:.....	44
Lösung Strings 1:.....	45
Übung Ausnahmen 1:.....	46
Lösung Ausnahmen 1:.....	48
Übung Ausnahmen 2:.....	49
Lösung Ausnahmen 2:.....	50
Übung Methoden 2:.....	51
Lösung Methoden 2:.....	52
Übung Punkt, Quadrat, Rechteck, Kreis.....	54
Lösung Punkt, Quadrat, Rechteck, Kreis.....	55
Übung Oberklassen/Unterklassen.....	56
Lösung Oberklassen/Unterklassen.....	57
Übung Bitfummeln.....	58
Lösung Bitfummeln.....	59
Übung Einfach boolean.....	60
Lösung Einfach boolean.....	61

## Übung if:

Die verschiedenen Varianten der if-Anweisung werden im Buch im Abschnitt 4.2.2 behandelt.

In der vorliegenden Übung sollen alle Berechnungen mit **Ganzzahlen** (vom Typ `int`) durchgeführt werden (und nicht mit **Bruchzahlen**). Angenommen, Sie haben die folgenden Vereinbarungen schon geschrieben:

```
1 int n1      = EM.liesInt();
2 int n2      = EM.liesInt();
3 int n3      = EM.liesInt();
4 int n4      = EM.liesInt();
5 int n5      = EM.liesInt();
6 int max     = 17;
7 int betrag  = EM.liesInt();
```

Befehlen Sie jetzt mit geeigneten `if`-Anweisungen dem Ausführer, die folgenden Arbeiten zu erledigen:

1. Falls die Variable `n1` einen negativen Wert enthält, soll ihr der Wert 0 zugewiesen werden.
2. Falls die Variable `n2` einen positiven Wert enthält, soll ihr Wert um 10 Prozent (Ganzzahlrechnung!) erhöht werden und dann zusammen mit dem Text "Der Wert von `n2` ist jetzt gleich " zur Standardausgabe ausgegeben werden (mit dem Befehl `println(...)`).
3. Falls die Variable `n3` einen geraden Wert enthält, soll ihr Wert halbiert werden. Sonst soll ihr Wert verdoppelt werden. Um festzustellen, ob `n3` gerade ist oder nicht, sollten Sie die Restoperation (Modulo-Operation) `%` verwenden.
4. Lösen Sie 3. noch einmal, aber diesmal ohne Restoperation `%` (nur mit den Operationen `+`, `-`, `*` und `/`). Was versteht man eigentlich unter dem Rest einer Ganzzahldivision?
5. Die beiden Zahlen `n1` und `n2` sollen (jede auf einer eigenen Zeile) ausgegeben werden, und zwar in aufsteigend sortierter Reihenfolge (d.h. zuerst die kleinere und dann die grössere Zahl).
6. Falls die Variable `n1` den Wert 0 enthält, soll sie unverändert bleiben. Sonst soll ihr Wert in Richtung 0 um 1 verändert werden (z.B. soll der Wert `-5` durch `-4` ersetzt werden oder `+7` durch `+6`).
7. Falls die Variable `n2` den Wert 0 enthält, soll sie unverändert bleiben. Sonst soll ihr Wert um 1 vermindert werden (z.B. soll der Wert `-5` durch `-6` ersetzt werden oder `+7` durch `+6`).
8. Die grösste der drei Zahlen `n1` bis `n3` soll der Variablen `max` zugewiesen werden. Versuchen Sie, eine **einfache** Lösung für diese Aufgabe zu finden, ehe Sie die folgende Aufgabe in Angriff nehmen.
9. Die grösste der fünf Zahlen `n1` bis `n5` soll der Variablen `max` zugewiesen werden.
10. Angenommen, die Variable `betrag` enthält einen Rechnungsbetrag. Wenn dieser Betrag grösser als 100 (aber nicht größer als 500) ist, soll er um 3 Prozent (Rabatt) vermindert werden. Falls der Betrag grösser als 500 (aber nicht grösser als 1000) ist, soll er um 5 Prozent vermindert werden. Falls der Betrag grösser als 1000 ist, soll er um 6 Prozent vermindert werden. Gestalten Sie Ihre Lösung möglichst so, dass der Kollege2 sie leicht um zusätzliche Rabattstufen erweitern kann (z.B. 8 Prozent für Beträge über 10.000,- DM etc.). Dieses Problem kann man mit einer einzigen (relativ komplizierten) `if`-Anweisung oder mit mehreren (relativ einfachen) `if`-Anweisungen lösen. Beide haben Vor- und Nachteile. Welche der beiden Lösungen finden Sie besser? Warum?
11. Führen Sie das Programm `If01` (siehe unten) mit Papier, Bleistift und Radiergummi aus und nehmen Sie dabei an, dass der Benutzer den Wert `-10` eingibt (siehe Zeile 8 des Programms). Erzeugen Sie insbesondere alle Variablen auf ihrem Papier. Was steht nach Ausführung des Programms auf dem Bildschirm?

**Das Programm If01:**

```
1 // Datei If01.java
2 /* -----
3 Verschiedene Varianten der if-Anweisung (if, if-else, if-else-if ...).
4 ----- */
5 class If01 {
6     static public void main(String[] s) {
7         p("A If01: Eine Ganzzahl n? ");
8         int n = EM.liesInt();
9
10        // if-Anweisungen mit und ohne geschweifte Klammern:
11        if (n > 0) pln("B n ist positiv!");
12        if (n < 0) {pln("C n ist negativ!");}
13
14        // if-Anweisung mit mehreren Anweisungen darin:
15        if ((-9 <= n) && (n <= +9)) {
16            n = 2 * n;
17            pln("D n war einstellig und wurde");
18            pln("E verdoppelt zu " + n);
19        }
20
21        // if-else-Anweisung:
22        if (n % 2 == 0) {
23            pln("F n ist eine gerade Zahl!");
24        } else {
25            pln("G n ist eine ungerade Zahl!");
26        }
27
28        // if-else-if-else-Anweisung:
29        if (n % 3 == 0) {
30            pln("H n ist durch 3 teilbar!");
31        } else if (n % 4 == 0) {
32            pln("I n ist durch 4 teilbar!");
33        } else if (n % 5 == 0) {
34            pln("J n ist durch 5 teilbar!");
35        } else {
36            pln("K n ist nicht durch 3, 4 oder 5 teilbar!");
37        }
38
39        // Manchmal geht es besser ohne if-Anweisungen:
40        boolean nIstEinstellig = (-9 <= n) && (n <= +9);
41        boolean nIstZweistellig = (-99 <= n) && (n <= +99) && !nIstEinstellig;
42        pln("L Ist n einstellig? " + nIstEinstellig);
43        pln("M Ist n zweistellig? " + nIstZweistellig);
44
45        pln("N If01: Das war's erstmal!");
46    } // main
47 } // class If01
```

**Lösung if:**

```

1 // ----- Teilaufgabe 1. -----
2 if (n1 < 0) n1 = 0;
3 // ----- Teilaufgabe 2. -----
4 Wenn man Prozentrechnungen mit Ganzzahlen (z.B. int-Werten) durchführt,
5 sollte man die Reihenfolge der Operationen so wählen, dass die Rundungs-
6 fehler möglichst klein bleiben. Um z.B. 110 % von n2 zu berechnen ist
7 n2 * 110 / 100 besser als n2 / 100 * 110 oder n2 + n2/10 etc.
8
9 if (n2 > 0) {
10     n2 = n2 * 110 / 100; // Nicht n2 + n2/10 !!
11     pln("Der Wert von n2 ist jetzt gleich " + n2);
12 }
13 // ----- Teilaufgabe 3. -----
14 if (n3 % 2 == 0) {
15     n3 = n3 / 2; // oder abgekuerzt: n3 /= 2;
16 } else {
17     n3 = n3 * 2; // oder abgekuerzt: n3 *= 2;
18 }
19 // ----- Teilaufgabe 4. -----
20 8 / 3 ist gleich 2. 2 * 3 ist gleich 6. Die Differenz zwischen 6 und
21 der Ausgangszahl 8 ist der Rest (der Division von 8 durch 3).
22 Allgemein: dend % dor ist gleich dend - dend/dor*dor.
23
24 if (n3 - (n3 / 2) * 2 == 0) {
25     n3 = n3 / 2; // oder abgekuerzt: n3 /= 2;
26 } else {
27     n3 = n3 * 2; // oder abgekuerzt: n3 *= 2;
28 }
29 // ----- Teilaufgabe 5. -----
30 if (n1 <= n2) {
31     pln("n1: " + n1);
32     pln("n2: " + n2);
33 } else {
34     pln("n2: " + n2);
35     pln("n1: " + n1);
36 }
37 // ----- Teilaufgabe 6. -----
38 Obwohl der Aufgabentext mit "Wenn n1 gleich 0 ist ..." beginnt, sollte
39 die Lösung nicht mit if (n1 == 0) ... beginnen, denn unter dieser
40 Bedingung soll ja nichts gemacht werden.
41
42 Lösung 1:
43
44 if (n1 < 0) {
45     n1 = n1 + 1;
46 } else if (n1 > 0) {
47     n1 = n1 - 1;
48 }
49
50 Lösung 2 (leichter lesbar?):
51
52 if (n1 < 0) n1 = n1 + 1;
53 if (n1 > 0) n1 = n1 - 1;
54 // ----- Teilaufgabe 7. -----
55 Hier muss nicht zwischen positiven und negativen n1 unterschieden
56 werden:
57
58 if (n1 != 0) n1 = n1 - 1;
59 // ----- Teilaufgabe 8. -----
60 Analogie: Fußballmeisterschaft der Mannschaften n1, n2, n3 nach dem
61 k.o-Verfahren (Ausscheiden nach der ersten Niederlage). Die Variable
62 max ist "das Siegerpodest". Es enthält anfangs eine willkürlich ge-
63 wählte Mannschaft, dann immer den "vorläufigen Sieger" und am Ende
64 den "endgültigen Sieger". max verliert gegen n wenn max < n.
65
66 max = n1;
67 if (max < n2) max = n2;

```

```
68  if (max < n3) max = n3;
69  // ----- Teilaufgabe 9. -----
70  max = n1;
71  if (max < n2) max = n2;
72  if (max < n3) max = n3;
73  if (max < n4) max = n4;
74  if (max < n5) max = n5;
75  // ----- Teilaufgabe 10., Lösung A -----
76  Geschachtelte if-Anweisungen mit einfachen Bedingungen:
77
78  if      (betrag > 1000) {
79      betrag = betrag * 94 / 100; // 6 Prozent Rabatt
80  } else if (betrag > 500) {
81      betrag = betrag * 95 / 100; // 5 Prozent Rabatt
82  } else if (betrag > 100) {
83      betrag = betrag * 97 / 100; // 3 Prozent Rabatt
84  }
85  // ----- Teilaufgabe 10., Lösung B -----
86  Einfache if-Anweisungen mit komplizierteren Bedingungen:
87
88  if ( 100 < betrag && betrag <= 500) betrag = betrag * 97 / 100;
89  if ( 500 < betrag && betrag <= 1000) betrag = betrag * 95 / 100;
90  if (1000 < betrag                ) betrag = betrag * 94 / 100;
91  // ----- Teilaufgabe 11. -----
92  /* Ausgabe des Programms If01 fuer die Eingabe -10:
93  If01: Eine Ganzzahl n? -10
94  n ist negativ!
95  n ist eine gerade Zahl!
96  n ist durch 5 teilbar!
97  Ist n einstellig? false
98  Ist n zweistellig? true
99  If01: Das war's erstmal!
100 ----- */
```

## Übung ÜberAus 1:

Wie man Programme mit Papier und Bleistift ausführen kann, wird im **Kapitel 3 des Buches** erläutert.

1. Führen Sie das Programm `If01` (siehe vorige Übung) mit Papier, Bleistift und Radiergummi aus und nehmen Sie dabei an, dass der Benutzer den Wert **6** eingibt (siehe Zeile 8 des Programms). Erzeugen Sie insbesondere alle Variablen auf ihrem Papier. Was steht nach Ausführung des Programms auf dem Bildschirm?

2. Wie die vorige Übung, aber mit Eingabe -9.

3. Wie die vorige Übung, aber mit Eingabe 10.

4. Sie (in der Rolle des *Benutzers*) möchten, dass das Programm `If01` unter anderem die Meldung

Ist n einstellig? true

ausgibt (siehe Zeile 40 und 42 des Programms). Welche Zahlen dürfen Sie dann (für den Lesebefehl in Zeile 8) eingeben? Berücksichtigen Sie dabei auch den Befehl in Zeile 16!

5. Beantworten Sie die folgenden Fragen mit je einem kurzen bzw. mittellangen Satz:

5.1. Was ist eine *Variable*?

5.2. Was ist ein *Typ*?

5.3. Was ist ein *Modul*?

6. Geben Sie von jedem der folgenden Befehle an, zu welcher Art von Befehlen er gehört (*Anweisung*, *Ausdruck* oder *Vereinbarung*, siehe dazu den Abschnitt 1.5 im Buch) und übersetzen Sie den Befehl ins Deutsche (oder ins Englische, wenn Sie wollen):

```
1 int mirko = 3;
2 String sascha = " Pickelheringe";
3 mirko + sascha
4 mirko + 14
5 sascha = mirko + sascha;
6 mirko = mirko + sascha.length();
7 if (mirko > 16) mirko = mirko - 1;
```

7. Führen Sie die folgende Befehlsfolge (mit Papier und Bleistift) aus. Welche Werte werden der Variablen `n` "im Laufe der Zeit" nacheinander zugewiesen? Welche Zahl wird zum Schluss als Ergebnis ausgegeben?

```
8 int zaehler = 0;
9 int n = 13;
10 while (n != 1) {
11     if (n % 2 == 0) {
12         n = n / 2;
13     } else {
14         n = 3 * n + 1;
15     }
16     zaehler = zaehler + 1;
17 }
18 pl("Ergebnis: " + zaehler);
```

## Lösung ÜberAus 1:

### 1. Die Ausgabe des Programms If01 für die Eingaben 6:

```
1 If01: Eine Ganzzahl n? 6
2 n ist positiv!
3 n war einstellig und wurde
4 verdoppelt zu 12
5 n ist eine gerade Zahl!
6 n ist durch 3 teilbar!
7 Ist n einstellig? false
8 Ist n zweistellig? true
9 If01: Das war's erstmal!
```

### 2. Die Ausgabe des Programms If01 für die Eingaben -9:

```
10 If01: Eine Ganzzahl n? -9
11 n ist negativ!
12 n war einstellig und wurde
13 verdoppelt zu -18
14 n ist eine gerade Zahl!
15 n ist durch 3 teilbar!
16 Ist n einstellig? false
17 Ist n zweistellig? true
18 If01: Das war's erstmal!
```

### 3. Die Ausgabe des Programms If01 für die Eingaben 10:

```
19 If01: Eine Ganzzahl n? 10
20 n ist positiv!
21 n ist eine gerade Zahl!
22 n ist durch 5 teilbar!
23 Ist n einstellig? false
24 Ist n zweistellig? true
25 If01: Das war's erstmal!
```

4. Genau dann wenn der Benutzer eine Zahl zwischen -4 und 4 (einschliesslich) eingibt, gibt das Programm If01 (unter anderem) die Meldung `Ist n einstellig? true` aus.

### 5. Antworten auf Fragen:

5.1. Eine **Variable** ist ein **Behälter** für **Werte** (ein Wertebehälter).

5.2. Ein **Typ** ist ein **Bauplan** für **Variablen** (für Wertebehälter).

5.3. Ein **Modul** ist ein **Behälter** für Variablen, Unterprogramme, Typen und andere Dinge, der aus einem **sichtbaren** und einem **unsichtbaren** Teil besteht.

### 6. Befehle klassifizieren und übersetzen:

6.1. Vereinbarung: Erzeuge eine Variable namens `mirko` vom Typ `int` mit dem Anfangswert 3.

6.2. Vereinbarung: Erzeuge eine Variable namens `sascha` vom Typ `String` und gib ihr den Anfangswert (eigentlich: Anfangszielwert) `"Pickelheringe"`.

6.3. Ausdruck: Berechne den Wert des Ausdrucks `mirko + sascha`.

6.4. Ausdruck: Berechne den Wert des Ausdrucks `mirko + 14`.

6.5. Anweisung: Berechne den Wert des Ausdrucks `mirko + sascha` und tue ihn in den Wertebehälter `mirko` (oder: weise ihn der Variablen `mirko` zu).

6.5. Anweisung: Berechne den Wert des Ausdrucks `mirko + sascha.length()` und weise ihn der Variablen `sascha` zu.

6.5. Berechne den Wert des Ausdrucks `mirko > 16`. Wenn dieser Wert gleich `true` ist, dann berechne den Wert des Ausdrucks `mirko - 1` und weise ihn der Variablen `mirko` zu.

### 7. Der Variablen `n` werden nacheinander die folgenden Werte zugewiesen:

13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

In der Variablen `zaehler` steht nach Ausführung aller Befehle der Wert 9.



## Übung Schleifen (ausführen) 1:

Einfache Schleifen werden in den Abschnitten 4.2.4 bis 4.2.7 des Buches behandelt, wie man Programme mit Papier und Bleistift ausführt im Kapitel 3.

Führen Sie die folgenden Befehlsfolgen mit Papier, Bleistift und Radiergummi aus und geben Sie genau an, welche Zeichen und Zeilen zur *Standardausgabe* (d.h. zum Bildschirm) ausgegeben werden. Beachten Sie dabei den Unterschied zwischen `p` (der Bildschirmzeiger bleibt unmittelbar hinter den ausgegebenen Zeichen stehen) und `pln` (der Bildschirmzeiger rückt zum Anfang der nächsten Zeile vor). Mit `final int ...` wird eine *unveränderbare Variable* („Konstante“) vom Typ `int` vereinbart. Hat der Ausführer eine Schleife wie `for (int i=1; ...) {...}` fertig ausgeführt, so *zerstört* er die Schleifenvariable `i`.

### 1. Die anna-Schleife:

```
1 int anna = 7;
2 while (anna > 0) {
3     p(anna + " ");
4     anna /= 2;
5 }
6 pln();
```

### 2. Die berta-Schleife:

```
7 int berta = 7;
8 do {
9     berta /= 2;
10    p(" -> " + berta);
11 } while (berta > 0);
12 pln();
```

### 3. Die celia-Schleife:

```
13 for (int celia = -3; celia < 5; celia += 2) {
14     p(2 * celia + 4 + " ");
15 }
16 pln();
```

### 4. Die dora-Schleife:

```
17 for (int dora = 3; 2*dora > -3; dora--) {
18     p(dora + " ");
19 }
20 pln();
```

### 5. Die MAX1-Schleife:

```
21 final int MAX1 = 3;
22 for (int i1 = 1; i1 <= MAX1; i1++) {
23     for (int i2 = 1; i2 <= 2*MAX1; i2++) {
24         p("**");
25     }
26     pln();
27 }
```

### 6. Die MAX2-Schleife:

```
28 final int MAX2 = 5;
29 for (int i1 = 1; i1 <= MAX2; i1++) {
30     for (int i2 = 1; i2 <= i1; i2++) {
31         p("++");
32     }
33     pln();
34 }
```

**Lösung Schleifen (ausführen) 1:****1. Die anna-Schleife**anna: Bildschirm: **2. Die berta-Schleife**berta: Bildschirm: **3. Die celia-Schleife**celia: Bildschirm: **4. Die dora-Schleife**dora: Bildschirm: **5. Die MAX1-Schleife**MAX1: Bildschirm: i1: i2: i2: i2: **6. Die MAX2-Schleife**MAX2: Bildschirm: i1: i2: i2: i2: i2: i2:

## Übung Schleifen (programmieren) 2:

Einfache Schleifen werden in den Abschnitten 4.2.4 bis 4.2.7 des Buches behandelt.

Einige der folgenden Übungsaufgaben *hängen zusammen*, d.h. die Lösung der *einen* kann zu einer Lösung der *anderen* „ausgebaut“ werden. Das ist manchmal leichter, als „alles neu zu programmieren“. Bei Zahlenfolgen (z.B. -5 -2 1 4 7 etc.) ist häufig die *Differenzenfolge* (3, 3, 3, 3 etc.) hilfreich.

1. Programmieren Sie eine Schleife, die die Ganzzahlen von 1 bis 10 (alle auf einer Zeile, durch je ein Blank voneinander getrennt) zum Bildschirm ausgibt, etwa so:

```
1 2 3 4 5 6 7 8 9 10
```

2. Programmieren Sie eine Schleife, die alle durch 3 teilbaren Ganzzahlen zwischen 10 und 40 zur Standardausgabe ausgibt, etwa so:

```
12 15 18 21 24 27 30 33 36 39
```

Ist Ihre Lösung *effizient* oder haben Sie dem Ausführer viele *unnötige Befehle* gegeben?

Programmieren Sie vier Schleifen, die die folgenden Zahlenfolgen zur Standardausgabe ausgeben:

```
3.   -5 -2 1 4 7 10 13 16 19
4.   1 2 4 8 16 32 64 128 256 512 1024 2048 4096 (siehe Anmerkung nach 6.)
5.   3 4 6 10 18 34 66 130 258 514 1026 2050 4098
6.   1 2 4 7 11 16 22 29 37 46 56 67 79 92
```

**Anmerkung zu 4.:** Einen Potenzoperator oder eine Potenzfunktion für Ganzzahlen gibt es in Java *nicht*.

Für die folgenden Aufgaben nehmen Sie bitte an, dass eine Ganzzahlvariable namens `norbert` vereinbart und mit einer Zahl *größer als 0* initialisiert wurde, etwa so:

```
int norbert = EM.liesInt(); // Der Benutzer gibt eine positive Zahl ein!.
```

7. Befehlen Sie dem Ausführer, den *größten* Teiler von `norbert` (der kleiner als `norbert` ist) auszugeben. Falls `norbert` eine Primzahl ist, soll 1 ausgegeben werden.

8. Befehlen Sie dem Ausführer, den *kleinsten* Teiler von `norbert` (der größer als 1 ist) auszugeben. Falls `norbert` eine Primzahl ist, soll `norbert` selbst ausgegeben werden.

9. Befehlen Sie dem Ausführer, die Meldung "`norbert` ist prim" bzw. "`norbert` ist nicht prim" auszugeben, je nachdem, ob die Variable `norbert` eine Primzahl enthält oder nicht.

Für die folgenden Aufgaben nehmen Sie bitte an, dass eine Stringvariable namens `sara` vereinbart wurde, etwa so:

```
String sara = EM.liesString();
```

Die wichtigsten String-Befehle (z.B. `sara.length()`, `sara.charAt(i)` etc.) werden im Abschnitt 10.1 Die Klasse String des Buches erläutert.

10. Befehlen Sie dem Ausführer zu zählen, wie oft das Zeichen 'x' in `sara` vorkommt. Wenn er damit fertig ist, soll er die Anzahl der 'x' ausgeben.

11. Befehlen Sie dem Ausführer die *Dezimalziffern* ('0', '1', ... '9') in `sara` zu zählen und diese Anzahl auszugeben. Zur Erinnerung: `char` ist ein Ganzzahltyp (ähnlich wie `int` und `long` etc.).

12. Befehlen Sie dem Ausführer die *Buchstaben* in `sara` zu zählen und diese Anzahl auszugeben. Als Buchstaben sollen alle Zeichen von 'a' bis 'z' und von 'A' bis 'Z' gelten.

13. Befehlen Sie dem Ausführer, die Anzahl der *führenden Nullen* in `sara` zu zählen und diese Anzahl auszugeben.

14. Befehlen Sie dem Ausführer zu zählen, wie oft in `sara` ein Zeichen 'a' unmittelbar vor einem Zeichen 'b' steht. Auch diese Anzahl soll der Ausführer ausgeben.

**Lösung Schleifen (programmieren) 2:**

1. Ausgeben ist die Folge 1 2 3 4 5 6 7 8 9 10

```
1 for (int i = 1; i <= 10; i++) p(i + " ");
```

2. Ausgeben sind alle Zahlen zwischen 10 und 40, die durch 3 teilbar sind, d.h. die Folge

12 15 18 21 24 27 30 33 36 39

Folgt man dem Wortlaut der Aufgabe, erhält man eine komplizierte und ineffiziente Lösung, etwa:

```
2 for (int i=10; i<=40; i++) {
3     if (i%3 == 0) p(i + " ");
4 }
```

Der Rumpf dieser Schleife wird 30 Mal ausgeführt, aber nur bei 10 Ausführungen "passiert etwas Interessantes". Die folgende Lösung ist schwerer zu finden, dann aber leichter zu lesen und leichter auszuführen:

```
5 for (int j = 12; j <= 39; j += 3) p(j + " ");
```

Auch wenn ein Kunde einem ein Problem auf komplizierte Weise erklärt ("alle durch 3 teilbaren Zahlen zwischen 10 und 40"), sollte man nach einer einfachen Lösung dafür suchen.

3. Ausgeben ist die Folge -5 -2 1 4 7 10 13 16 19

```
6 for (int k = -5; k <= 19; k += 3) p(k + " ");
```

4. Ausgeben ist die Folge 1 2 4 8 16 32 64 128 256 512 1024 2048 4096

```
7 for (int i = 1; i <= 4096; i *= 2) p(i + " ");
```

5. Ausgeben ist die Folge 3 4 6 10 18 34 66 130 258 514 1026 2050 4098

Jedes Element dieser Folge ist um 2 größer als das entsprechende Element der vorigen Folge (siehe Teilübung 4.). Man kann die vorige Lösung also abschreiben und muss nur `p(i + " ")` durch `p((i+2) + " ")` ersetzen:

```
8 for (int i = 1; i <= 4096; i *= 2) p((i+2) + " ");
```

6. Ausgeben ist die Folge 1 2 4 7 11 16 22 29 37 46 56 67 79 92

Die Differenzenfolge: 1 2 3 4 5 6 7 8 9 10 11 12 13

ist eine Verlängerung der Folge aus der Teilübung 1. Also können wir die Lösung zu 1. hier wiederverwenden:

```
9 int zahl = 1;
10 for (int i = 1; i <= 14; i++) {
11     p(zahl + " ");
12     zahl += i;
13 }
```

**Alternative Lösung (mit Dank an Herrn Gauss und Frau Mehrnaz Goldeh):**

```
14 for (int n=0; n < 14; n++) {
15     p((n*(n+1)/2 + 1));
16 }
```

7. Der grösste Teiler von norbert: Er kann nicht größer als  $n/2$  sein. Am besten sucht man "von oben nach unten", damit man den größten Teiler zuerst findet:

```
17 int kandidat = norbert / 2;
18 while (norbert % kandidat != 0) kandidat--;
19 pln("Der grösste Teiler von norbert ist " + kandidat);
```

8. Der kleinste Teiler von norbert: Am besten sucht man "von unten nach oben", damit man den kleinsten Teiler zuerst findet

```
20 int kandidat = 2;
21 while (norbert % kandidat != 0) kandidat++;
22 println("Der kleinste Teiler von norbert ist " + kandidat);
```

### 9. Enthält norbert eine Primzahl oder nicht?

```
23 for (int kandidat = 2; kandidat < norbert/2; kandidat++) {
24     if (norbert % kandidat == 0) {
25         meldung = "norbert ist nicht prim!";
26         break;
27     }
28 }
29 println("norbert ist prim!");
```

Schnellere Lösungen für dieses Problem werden auch heute noch im Rahmen von Forschungsprojekten und Doktorarbeiten entwickelt.

### 10. Wie oft kommt 'x' in sara vor? Eine Methode wie `sara.length()` aufzurufen ist relativ teuer. Wiederholte Aufrufe kann man vermeiden, etwa so:

```
30 int anzahl_x = 0;
31 for (int i=sara.length()-1; i >= 0; i--) {
32     if (sara.charAt(i) == 'x') anzahl_x++;
33 }
34 println("Anzahl 'x' in sara: " + anzahl_x);
```

oder so:

```
35 for (int max=sara.length(), i=0; i<max; i++) {
36     if (sara.charAt(i) == 'x') anzahl_x++;
37 }
```

### 11. Wieviele Dezimalziffern kommen in sara vor?

```
38 int anzahlZiffern = 0;
39 for (int i=sara.length()-1; i >= 0; i--) {
40     char c = sara.charAt(i);
41     if ('0' <= c && c <= '9') anzahlZiffern++;
42 }
43 println("Anzahl Ziffern in sara: " + anzahlZiffern);
```

### 12. Wieviele Buchstaben kommen in sara vor?

```
44 int anzahlBuchstaben = 0;
45 for (int i=sara.length()-1; i >= 0; i--) {
46     char c = sara.charAt(i);
47     if (('a' <= c && c <= 'z') ||
48         ('A' <= c && c <= 'Z')) anzahlBuchstaben++;
49 }
50 println("Anzahl Buchstaben in sara: " + anzahlBuchstaben);
```

### 13. Wieviele führende Nullen kommen in sara vor?

```
51 int anzahlFuehrendeNullen = 0;
52 int i = 0;
53 while (sara.charAt(i) == '0') anzahlFuehrendeNullen++;
54 println("Anzahl fuehrender Nullen: " + anzahlFuehrendeNullen);
```

### 14. Wie oft kommt 'a' unmittelbar vor 'b' in sara?

```
55 int anzahlAvorB = 0;
56 int i = 0;
57 int laenge = sara.length();
58 while (i <= laenge - 2) {
59     if (sara.charAt(i) == 'a' && sara.charAt(i+1) == 'b') {
60         anzahlAvorB++;
61         i += 2;
62     } else {
63         i += 1;
64     }
65 }
```

## Übung Methoden 1:

Unterprogramme (oder: Methoden) werden in den Abschnitten 1.4.3 und 2.1 bis 2.2 des Buches kurz eingeführt und im Kapitel 8 gründlicher behandelt.

**Zur Erinnerung:** Methoden mit dem Rückgabotyp `void` werden auch als *Prozeduren* bezeichnet. Alle anderen Methoden ("Methoden mit einem richtigen, von `void` verschiedenen Rückgabotyp") werden auch als *Funktionen* bezeichnet.

1. Ergänzen Sie das folgende "Skelett einer `int`-Funktion" zu einer `int`-Funktion, indem Sie die Auslassung "..." durch geeignete Befehle ersetzen:

```
1 static public int summe(int n1, int n2) {  
2     // Liefert die Summe von n1 und n2 als Ergebnis.  
3     ...  
4 }
```

2. Programmieren Sie eine `static-public`-Prozedur namens `gibAus`, die drei Parameter vom Typ `int` hat. Die Prozedur soll ihre drei Parameter in lesbarer Form (mit kleinen Texten verziert) zur Standardausgabe ausgeben. Die verzierenden Texte können Sie ganz nach Ihrem Geschmack wählen.

3. Programmieren Sie eine `static-public-int`-Funktion namens `hochZwei` mit einem `int`-Parameter namens `grundzahl`. Als Ergebnis soll das Quadrat der `grundzahl` (d.h.  $\text{grundzahl}^2$ ) geliefert werden.

4. Programmieren Sie eine `static-public-int`-Funktion namens `zweiHoch` mit einem `int`-Parameter namens `exponent`. Falls der `exponent` kleiner oder gleich 0 ist, soll als Ergebnis der Wert 1 geliefert werden. Sonst soll die entsprechende Potenz von 2 (d.h.  $2^{\text{exponent}}$ ) als Ergebnis geliefert werden.

5. Programmieren Sie eine `static-public-int`-Funktion namens `hoch` mit zwei `int`-Parametern namens `grundzahl` und `exponent`. Falls der `exponent` kleiner oder gleich 0 ist, soll als Ergebnis die Zahl 1 geliefert werden. Sonst soll die entsprechende Potenz der `grundzahl` (d.h. die Ganzzahl  $\text{grundzahl}^{\text{exponent}}$ ) als Ergebnis geliefert werden.

6. Programmieren Sie eine `static-public-boolean`-Funktion namens `istPrim` mit einem `int`-Parameter namens `n`. Falls `n` eine Primzahl ist, soll die Funktion das Ergebnis `true` liefern und sonst das Ergebnis `false`.

7. Eine *Primzahl-Doublette* besteht aus zwei Primzahlen, deren Differenz gleich 2 ist (z.B. 3 und 5 oder 11 und 13 oder 1019 und 1021 etc.). Programmieren Sie eine `int`-Funktion namens `primDoublette` mit einem `int`-Parameter namens `min`. Diese Funktion soll von der Zahl `min` an aufwärts nach einer Primzahl-Doublette suchen. Falls sie eine findet, soll sie die kleinere Primzahl der Doublette als Ergebnis liefern. Falls es im Bereich zwischen `min` und der größten Zahl des Typs `int` (`Integer.MAX_VALUE`) keine Primzahl-Doublette gibt, soll die Funktion `primDoublette` den Wert 0 als Ergebnis liefern.

8. Programmieren Sie eine parameterlose Prozedur namens `alleRechtwinkligen`, die alle Ganzzahltripel (`a`, `b`, `c`) zur Standardausgabe ausgibt, für die gilt:

8.1. Jede der drei Ganzzahlen `a`, `b` und `c` liegt zwischen 1 und 100 (einschliesslich).

8.2. `a` ist größer oder gleich `b` und `b` ist größer oder gleich `c`.

8.3. Die drei Ganzzahlen repräsentieren die Seiten eines *rechtwinkligen Dreiecks*, d.h. es gilt "der Pythagoras"  $a^2 = b^2 + c^2$ .

## Lösung Methoden 1:

### 1. Eine Funktion namens `summe`:

```
1 static public int summe(int n1, int n2) {  
2     return n1 + n2;  
3 }
```

Diese Teilübung war hoffentlich ganz einfach und diente vor allem zum "Aufwärmen".

### 2. Eine Prozedur namens `gibAus`:

```
4 static public void gibAus(int otto, int emil, int berta) {  
5     pln("Die erste Zahl: " + otto);  
6     pln("Die zweite Zahl: " + emil);  
7     pln("Die dritte Zahl: " + berta);  
8 }
```

Hoffentlich haben Sie eine Lösung mit ein bisschen phantasievolleren Texten gefunden.

### 3. Eine Funktion namens `hochZwei`:

```
9 static public int hochZwei(int grundzahl) {  
10     return grundzahl * grundzahl;  
11 }
```

### 4. Eine Funktion namens `zweiHoch`:

```
12 static public int zweiHoch(int exponent) {  
13     if (exponent <= 0) return 1;  
14     int ergebnis = 2;  
15     for (; exponent > 1; exponent--) {  
16         ergebnis *= 2;  
17     }  
18     return ergebnis;  
19 }
```

### 5. Eine Funktion namens `hoch`:

```
20 static public int hoch(int grundzahl, int exponent) {  
21     if (exponent <= 0) return 1;  
22     int ergebnis = grundzahl;  
23     for (; exponent > 1; exponent--) {  
24         ergebnis *= grundzahl;  
25     }  
26     return ergebnis;  
27 }
```

### 6a. Eine Funktion namens `istPrim` (simple Version):

```
28 static public boolean istPrim(int n) {  
29     // Liefert true, falls n prim ist, und sonst false.  
30  
31     // Falls n negativ, gleich 0 oder 1 ist:  
32     if (n <= 1) return false;  
33  
34     // Falls n gleich 2 oder groesser ist:  
35     for (int kandidat=2; kandidat < n/2; kandidat++) {  
36         if (n % kandidat == 0) return false;  
37     }  
38     return true;  
39 } // istPrim
```

### 6b. Eine Funktion namens `istPrim` (effizientere Version):

```
40 static public boolean istPrim(final int n) {  
41     // Liefert true, falls n prim ist, und sonst false.  
42     // Prueft nur Zahlen bis zur Quadratwurzel aus n als Teiler.  
43     // Behandelt alle geraden Zahlen "vorweg" als Sonderfälle.  
44     // Der Parameter n wurde als final ("konstant") vereinbart,
```

```

45 // damit der Ausfuehrer unten in der for-Schleife
46 // Math.sqrt(n) nur einmal berechnen muss.
47
48 if (n <= 1) return false; // Alle zu kleinen Zahlen erledigen.
49 if (n == 2) return true; // Einzige gerade Primzahl.
50 if (n%2 == 0) return false; // Alle anderen geraden Zahlen sind nicht prim!
51
52 for (int kandidat=3; kandidat < Math.sqrt(n); n+=2) {
53     if (n % kandidat == 0) return false;
54 }
55 return true;
56 } // istPrim

```

In Zeile 35 steht `kandidat < n/2`, in Zeile 52 steht statt dessen `kandidat < Math.sqrt(n)`. Wieviel schneller ist die Lösung 6b im Vergleich zu 6a allein dadurch geworden?

n	n/2	Math.sqrt(n)	Beschleunigungsfaktor
10 000	5 000	100	50
1 000 000	500 000	1000	500
100 000 000	50 000 000	10 000	5 000

#### 6c. Eine Funktion namens `istPrim` (noch effizientere Version):

```

57 static public boolean istPrim(final int n) {
58     // Liefert true, falls n prim ist, und sonst false.
59     // Prueft nur Zahlen bis zur Quadratwurzel aus n als Teiler.
60     // Behandelt durch 2 und durch 3 teilbare Zahlen "vorweg" und
61     // ueberspringt dann beim Suchen nach einem Teiler alle durch
62     // 2 und alle durch 3 teilbaren Teiler (das sind 2/3 aller Zahlen).
63
64     // Ein paar Sonderfaelle erledigen:
65     if (n <= 1) return false; // alle zu kleinen Zahlen erledigen
66     if (n == 2) return true;
67     if (n == 3) return true;
68     if (n % 2 == 0) return false; // alle durch 2 teilbaren Zahlen erledigt
69     if (n % 3 == 0) return false; // alle durch 3 teilbaren Zahlen erledigt
70
71     // Und jetzt den allgemeinen Fall bearbeiten:
72     int teiler = 5;
73     while (teiler < Math.sqrt(n)) {
74         if (n % teiler == 0) return false;
75         teiler += 2; // eine gerade Zahl ueberspringen
76         if (n % teiler == 0) return false;
77         teiler += 4; // eine durch 3 teilbare und zwei gerade Zahlen ueberspring.
78     }
79     return true;
80 } // istPrim

```

#### 7. Die Funktion `primDoublette`:

```

81 static public int primDoublette(int minimum) {
82     if (minimum <= 3) return 5; // kleinste Prim-Doublette
83     if (minimum % 2 == 0) minimum++; // minimum ungerade machen
84     for (int n=minimum; n<=Integer.MAX_VALUE-2; n += 2) {
85         if (istPrim(n) && istPrim(n+2)) return n;
86     }
87     return 0;
88 } // primDoublette

```



**8. Eine Prozedur namens alleRechtwinkligen:**

```
89 static public void alleRechtwinkligen() {
90     // Gibt alle Ganzzahltripel (a, b, c) aus fuer die gilt:
91     // 1. a, b und c liegen zwischen 1 und MAX (einschliesslich)
92     // 2. a ist groesser/gleich b und b ist groesser/gleich c
93     // 3.  $a^2$  ist gleich  $b^2 + c^2$  (Pythagoras)
94
95     final int MAX = 100;
96     int nr = 0;
97
98     for (int a=1; a <= MAX; a++) {
99         for (int b=1; b <= a; b++) {
100             for (int c=1; c <= b; c++) {
101                 if (a*a == b*b + c*c) {
102                     nr++;
103                     println("Nr. " +
104                         nr + ": " +
105                         a + ", " +
106                         b + ", " +
107                         c + ".")
108                 }
109             } // if
110         } // for c
111     } // for b
112 } // for a
113 } // alleRechtwinkligen
```

**Übung Reihungen 1:**

**Reihungen** werden im Kapitel 7 des Buches und **Methoden** im Kapitel 8 behandelt.

Ergänzen Sie die folgenden Methoden-*Skelette* zu richtigen *Methoden*.

```
1 // -----
2 static public void printR(int[] ir) {
3     // Gibt die Reihung ir in lesbarer Form zur Standardausgabe aus.
4     ...
5 } // printR
6 // -----
7 static public int min(int[] ir) {
8     // Liefert die kleinste Komponente von ir bzw. Integer.MAX_VALUE
9     // falls ir aus 0 Komponenten besteht.
10    ...
11 } // min
12 // -----
13 static public boolean mindEineGerade(int[] ir) {
14     // Liefert true, wenn ir mindestens eine gerade Zahl enthaelt und
15     // sonst false.
16     ...
17 } // mindEineGerade
18 // -----
19 static public boolean alleGerade(int[] ir) {
20     // Liefert true, wenn alle Komponenten von ir gerade Zahlen sind und
21     // sonst false.
22     ...
23 } // alle Gerade
24 // -----
25 static public boolean kommtVor(int dieserWert, int[] inDieserReihung) {
26     // Liefert true, wenn dieserWert als Komponente in inDieserReihung
27     // vorkommt und sonst false.
28     ...
29 } // kommtVor
30 // -----
31 static public int index(int n, int[] ir) {
32     // Liefert den kleinsten Index i fuer den gilt n == ir[i] bzw. -1,
33     // falls es keinen solchen Index gibt (d.h. falls n in ir nicht vor-
34     // kommt.
35     ...
36 } // index
37 // -----
38 static public void pAlleDurch2(int[] ir) {
39     // Teilt jede Komponente von ir durch 2.
40     ...
41 } // pAlleDurch2
42 // -----
43 static public int[] fAlleDurch2(int[] ir) {
44     // Liefert eine Kopie der Reihung ir, in der alle Komponenten durch 2
45     // geteilt wurden.
46     ...
47 } // fAlleDurch2
48 // -----
49 static public boolean sindGleich(int[] ira, int[] irb) {
50     // Liefert true, wenn ira und irb gleich lang sind und jede Kompo-
51     // nente ira[i] gleich der entsprechenden Komponente irb[i] ist.
52     // Liefert sonst false.
53     ...
54 } // sindGleich
55 // -----
56 static public boolean sindDisjunkt(int[] ira, int[] irb) {
57     // Liefert true, falls jede Zahl in ira verschieden ist von jeder
58     // Zahl in irb. Liefert sonst false.
59     ...
60 } // sindDisjunkt
61 // -----
```

## Lösung Reihungen 1:

```

1 // Datei Ueb_Reihungen1.java
2 /* -----
3 Methoden zum Bearbeiten einstufiger Reihungen.
4 Von einigen Methoden gibt es zwei Varianten, z. N. minA und minN.
5 Die A-Version enthaelt eine alte for-Schleife (seit Java 1.0)
6 Die N-Version enthaelt eine neue for-Schleife (seit Java 5)
7 ----- */
8 class Ueb_Reihungen1 {
9     // -----
10    public static void printRA(int[] ir) {
11        // Gibt die Reihung ir in lesbarer Form zur Standardausgabe aus.
12        // Hinweis: Auf jeden Fall muessen auch leere Reihungen "lesbar"
13        // ausgegeben werden. Ob auch eine Nullreferenz ir "lesbar" aus-
14        // gegeben werden sollte oder eine NullPointerException ausloesen
15        // sollte, kann man diskutieren.
16        pln("-----");
17
18        if (ir == null) {
19            pln("printRA: ir zeigt nicht auf eine Reihung!");
20            return;
21        }
22        pln("Eine Reihung mit int-Komponenten:");
23        if (ir.length == 0) {
24            pln("Die Reihung besteht aus 0 Komponenten!");
25            return;
26        }
27        for (int i=0; i<ir.length; i++) {
28            pln("Index " + i + ", Komponente " + ir[i]);
29        }
30    } // printRA
31    // -----
32    public static void printRN(int[] ir) {
33        // Gibt die Reihung ir in lesbarer Form zur Standardausgabe aus.
34        // Hinweis: Auf jeden Fall muessen auch leere Reihungen "lesbar"
35        // ausgegeben werden. Ob auch eine Nullreferenz ir "lesbar" aus-
36        // gegeben werden sollte oder eine NullPointerException ausloesen
37        // sollte, kann man diskutieren.
38        pln("-----");
39
40        if (ir == null) {
41            pln("printRN: ir zeigt nicht auf eine Reihung!");
42            return;
43        }
44        pln("Eine Reihung mit int-Komponenten:");
45        if (ir.length == 0) {
46            pln("Die Reihung besteht aus 0 Komponenten!");
47            return;
48        }
49        for (int n: ir) {
50            pln("Komponente " + n);
51        }
52    } // printRN
53    // -----
54    public static int minA(int[] ir) {
55        // Liefert die kleinste Komponente von ir,
56        // bzw. Integer.MAX_VALUE falls ir aus 0 Komponenten besteht.
57        int erg = Integer.MAX_VALUE;
58        for (int i=0; i<ir.length; i++) {
59            if (erg > ir[i]) erg = ir[i];
60        }
61        return erg;
62    } // minA
63    // -----
64    public static int minN(int[] ir) {
65        // Liefert die kleinste Komponente von ir,
66        // bzw. Integer.MAX_VALUE falls ir aus 0 Komponenten besteht.
67        int erg = Integer.MAX_VALUE;
68        for (int n : ir) {
69            if (erg > n) erg = n;

```

```
70     }
71     return erg;
72 } // minN
73 // -----
74 public static boolean mindEineGeradeA(int[] ir) {
75     // Liefert true, wenn ir mindestens eine gerade Zahl enthaelt und
76     // sonst false.
77     for (int i=0; i<ir.length; i++) {
78         if (ir[i] % 2 == 0) return true;
79     }
80     return false;
81 } // mindEineGeradeA
82 // -----
83 public static boolean mindEineGeradeN(int[] ir) {
84     // Liefert true, wenn ir mindestens eine gerade Zahl enthaelt und
85     // sonst false.
86     for (int n: ir) {
87         if (n % 2 == 0) return true;
88     }
89     return false;
90 } // mindEineGeradeN
91 // -----
92 public static boolean alleGeradeA(int[] ir) {
93     // Liefert true, wenn alle Komponenten von ir gerade Zahlen sind und
94     // sonst false.
95     for (int i=0; i<ir.length; i++) {
96         if (ir[i] % 2 != 0) return false;
97     }
98     return true;
99 } // alleGeradeA
100 // -----
101 public static boolean alleGeradeN(int[] ir) {
102     // Liefert true, wenn alle Komponenten von ir gerade Zahlen sind und
103     // sonst false.
104     for (int n: ir) {
105         if (n % 2 != 0) return false;
106     }
107     return true;
108 } // alleGeradeN
109 // -----
110 public static boolean kommtVorA(int dieserWert, int[] inDieserReihung) {
111     // Liefert true, wenn dieserWert als Komponente in der Reihung
112     // namens inDieserReihung vorkommt und sonst false.
113     for (int i=0; i<inDieserReihung.length; i++) {
114         if (dieserWert == inDieserReihung[i]) return true;
115     }
116     return false;
117 } // kommtVorA
118 // -----
119 public static boolean kommtVorN(int dieserWert, int[] inDieserReihung) {
120     // Liefert true, wenn dieserWert als Komponente in der Reihung
121     // namens inDieserReihung vorkommt und sonst false.
122     for (int n: inDieserReihung) {
123         if (dieserWert == n) return true;
124     }
125     return false;
126 } // kommtVorN
127 // -----
128 public static int index(int n, int[] ir) {
129     // Liefert den kleinsten Index i fuer den gilt n == ir[i] bzw. -1,
130     // falls es keinen solchen Index gibt (d.h. falls n in ir nicht vor-
131     // kommt.
132     for (int i=0; i<ir.length; i++) {
133         if (n == ir[i]) return i;
134     }
135     return -1;
136 } // index
137 // -----
138 public static void oAlleDurch2(int[] ir) {
139     // Teilt jede Komponente von ir durch 2 (o wie "Originalparameter")
140     for (int i=0; i<ir.length; i++) {
```

```
141         ir[i] = ir[i] / 2;
142     }
143 } // oAlleDurch2
144 // -----
145 public static int[] kAlleDurch2(int[] ir) {
146     // Liefert eine Kopie der Reihung ir, in der alle Komponenten durch 2
147     // geteilt wurden.
148     int[] erg = new int[ir.length];
149     for (int i=0; i<erg.length; i++) {
150         erg[i] = ir[i] / 2;
151     }
152     return erg;
153 } // kAlleDurch2
154 // -----
155 public static boolean sindGleich(int[] ira, int[] irb) {
156     // Liefert true, wenn ira und irb gleich lang sind und jede Kompo-
157     // nente ira[i] gleich der entsprechenden Komponente irb[i] ist.
158     // Liefert sonst false.
159
160     if (ira.length != irb.length) return false;
161     for (int i=0; i<ira.length; i++) {
162         if (ira[i] != irb[i]) return false;
163     }
164     return true;
165 } // sindGleich
166 // -----
167 public static boolean sindDisjunktA(int[] ira, int[] irb) {
168     // Liefert true, falls jede Zahl in ira verschieden ist von jeder
169     // Zahl in irb. Liefert sonst false.
170     for (int ia = 0; ia < ira.length; ia++) {
171         for (int ib=0; ib < irb.length; ib++) {
172             if (ira[ia] == irb[ib]) return false;
173         }
174     }
175     return true;
176 } // sindDisjunktA
177 // -----
178 public static boolean sindDisjunktN(int[] ira, int[] irb) {
179     // Liefert true, falls jede Zahl in ira verschieden ist von jeder
180     // Zahl in irb. Liefert sonst false.
181     for (int na: ira) {
182         for (int nb: irb) {
183             if (na == nb) return false;
184         }
185     }
186     return true;
187 } // sindDisjunktN
188 // -----
189 public static void printRA(String[] sonja) {
190     // Gibt die Reihung sonja in lesbarer Form zur Standardausgabe aus.
191
192     pln("-----");
193     if (sonja == null) {
194         pln("printRA: sonja zeigt nicht auf eine Reihung!");
195         return;
196     }
197     pln("Eine Reihung mit String-Komponenten:");
198     if (sonja.length == 0) {
199         pln("Die Reihung besteht aus 0 Komponenten!");
200         return;
201     }
202     for (int i=0; i<sonja.length; i++) {
203         pln("Index " + i + ", Komponente " + sonja[i]);
204     }
205 } // printRA
206 // -----
207 } // class Ueb_Reihungen1
```

**Übung Reihungen 2:**

**Reihungen** werden im **Kapitel 7** des Buches und **Methoden** im **Kapitel 8** behandelt.

Ergänzen Sie folgenden Methoden-Skelette zu richtigen Methoden. Alle Methoden dienen dazu, zwei-stufige Reihungen zu bearbeiten:

```
1 // -----
2 static public int summe(int[][] irr) {
3     // Liefert die Summe aller int-Komponenten von irr.
4     ...
5 } // summe
6 // -----
7 static public boolean alleGerade(int[][] irr) {
8     // Liefert true, falls alle int-Komponenten von irr gerade sind und
9     // sonst false.
10    ...
11 } // alleGerade
12 // -----
13 static public int anzahlGerade(int[][] irr) {
14     // Liefert die Anzahl der geraden int-Komponenten von irr.
15     ...
16 } // anzahlGerade
17 // -----
18 static public boolean kommtVor(int dieserWert, int[][] inDieserReihung) {
19     // Liefert true, wenn dieserWert in inDieserReihung (als int-Komponente)
20     // vorkommt, und sonst false.
21     ...
22 } // kommtVor
23 // -----
24 static public int anzahlKomponenten(int[][] irr) {
25     // Liefert die Anzahl der int-Komponenten von irr.
26     ...
27 } // anzahlKomponenten
28 // -----
29 static public void printR(String[] sonja) {
30     // Gibt die Reihung sonja in lesbarer Form zur Standardausgabe aus.
31     ...
32 } // printR
33 // -----
34 static public int anzBuchstaben(String[] sonja) {
35     // Liefert die Anzahl der Buchstaben, die in den Komponenten von
36     // sonja vorkommen.
37     ...
38 } // anzBuchstaben
39 // -----
```

**Lösung Reihungen 2:**

```

1 // Datei Ueb_Reihungen02.java
2 /* -----
3 Methoden zum Bearbeiten von zweistufigen Reihungen.
4 Hinweis: Von jeder Methode gibt es zwei Versionen, z.B. summeN und summeA.
5 Die N-Version enthaelt eine neue for-Schleife (seit Java 5 neu).
6 Die A-Version enthaelt eine alte for-Schleife (seit Java 1.0)
7 ----- */
8 class Ueb_Reihungen02 {
9     // -----
10    static public int summeN(int[][] irr) {
11        // Liefert die Summe aller int-Komponenten von irr.
12        int erg = 0;
13        for (int[] ir: irr) {
14            for (int i: ir) {
15                erg += i;
16            }
17        }
18        return erg;
19    } // summeN
20    // -----
21    static public int summeA(int[][] irr) {
22        // Liefert die Summe aller int-Komponenten von irr.
23        int erg = 0;
24        for (int i1=0; i1<irr.length; i1++) {
25            for (int i2=0; i2<irr[i1].length; i2++) {
26                erg += irr[i1][i2];
27            }
28        }
29        return erg;
30    } // summeA
31    // -----
32    static public boolean alleGeradeN(int[][] irr) {
33        // Liefert true, falls alle int-Komponenten von irr gerade sind und
34        // sonst false.
35        for (int[] ir: irr) {
36            for (int i: ir) {
37                if (i % 2 != 0) return false;
38            }
39        }
40        return true;
41    } // alleGeradeN
42    // -----
43    static public boolean alleGeradeA(int[][] irr) {
44        // Liefert true, falls alle int-Komponenten von irr gerade sind und
45        // sonst false.
46        for (int i1=0; i1<irr.length; i1++) {
47            for (int i2=0; i2<irr[i1].length; i2++) {
48                if (irr[i1][i2] % 2 != 0) return false;
49            }
50        }
51        return true;
52    } // alleGeradeA
53    // -----
54    static public int anzahlGeradeN(int[][] irr) {
55        // Liefert die Anzahl der geraden int-Komponenten von irr.
56        int erg = 0;
57        for (int[] ir: irr) {
58            for (int i: ir) {
59                if (i % 2 == 0) erg++;
60            }
61        }
62        return erg;
63    } // anzahlGeradeN
64    // -----
65    static public int anzahlGeradeA(int[][] irr) {
66        // Liefert die Anzahl der geraden int-Komponenten von irr.

```

```
67     int erg = 0;
68     for (int i1=0; i1<irr.length; i1++) {
69         for (int i2=0; i2<irr[i1].length; i2++) {
70             if (irr[i1][i2] % 2 == 0) erg++;
71         }
72     }
73     return erg;
74 } // anzahlGeradeA
75 // -----
76 static public boolean kommtVorN(int dieserWert, int[][] inDieserReihung) {
77     // Liefert true, wenn dieserWert in inDieserReihung (als int-Komponente)
78     // vorkommt, und sonst false.
79     for (int[] ir: inDieserReihung) {
80         for (int i: ir) {
81             if (dieserWert == i) return true;
82         }
83     }
84     return false;
85 } // kommtVorN
86 // -----
87 static public boolean kommtVorA(int dieserWert, int[][] inDieserReihung) {
88     // Liefert true, wenn dieserWert in inDieserReihung (als int-Komponente)
89     // vorkommt, und sonst false.
90     for (int i1=0; i1<inDieserReihung.length; i1++) {
91         if (inDieserReihung[i1] == null) continue;
92         for (int i2=0; i2<inDieserReihung[i1].length; i2++) {
93             if (dieserWert == inDieserReihung[i1][i2]) return true;
94         }
95     }
96     return false;
97 } // kommtVorA
98 // -----
99 static public int anzahlKomponentenN(int[][] irr) {
100     // Liefert die Anzahl der int-Komponenten von irr.
101     int erg = 0;
102     for (int[] ir: irr) {
103         erg += ir.length;
104     }
105     return erg;
106 } // anzahlKomponentenN
107 // -----
108 static public int anzahlKomponentenA(int[][] irr) {
109     // Liefert die Anzahl der int-Komponenten von irr.
110     int erg = 0;
111     for (int i=0; i<irr.length; i++) {
112         erg += irr[i].length;
113     }
114     return erg;
115 } // anzahlKomponentenA
116 // -----
117 static public void printR(String[] sonja) {
118     // Gibt die Reihung sonja in lesbarer Form zur Standardausgabe aus.
119     pln("Eine Reihung mit String-Komponenten:");
120     if (sonja.length == 0) {
121         pln("Die Reihung besteht aus 0 Komponenten!");
122         return;
123     }
124     for (int i=0; i<sonja.length; i++) {
125         pln("Index " + i + ", Komponente " + sonja[i]);
126     }
127 } // printR
128 // -----
```



```
129 static public int anzBuchstabenN(String[] sonja) {
130     // Liefert die Anzahl der Buchstaben, die in den Komponenten von
131     // sonja vorkommen.
132     int erg =0;
133     for (String s: sonja) {
134         for (int j=0, laengeJ=s.length(); j<laengeJ; j++) {
135             char c = s.charAt(j);
136             if ('a' <= c && c <= 'z' || 'A' <= c && c <= 'Z') erg++;
137         }
138     }
139     return erg;
140 } // anzBuchstabenN
141 // -----
142 static public int anzBuchstabenA(String[] sonja) {
143     // Liefert die Anzahl der Buchstaben, die in den Komponenten von
144     // sonja vorkommen.
145     int erg =0;
146     for (int i=0, laengeI=sonja.length; i<laengeI; i++) {
147         for (int j=0, laengeJ=sonja[i].length(); j<laengeJ; j++) {
148             char c = sonja[i].charAt(j);
149             if ('a' <= c && c <= 'z' || 'A' <= c && c <= 'Z') erg++;
150         }
151     }
152     return erg;
153 } // anzBuchstabenA
154 // -----
155 } // class Ueb_Reihungen02
```

**Übung Reihungen 3:**

**Mehrstufige Reihungen** werden im **Abschnitt 7.4** des Buches behandelt.

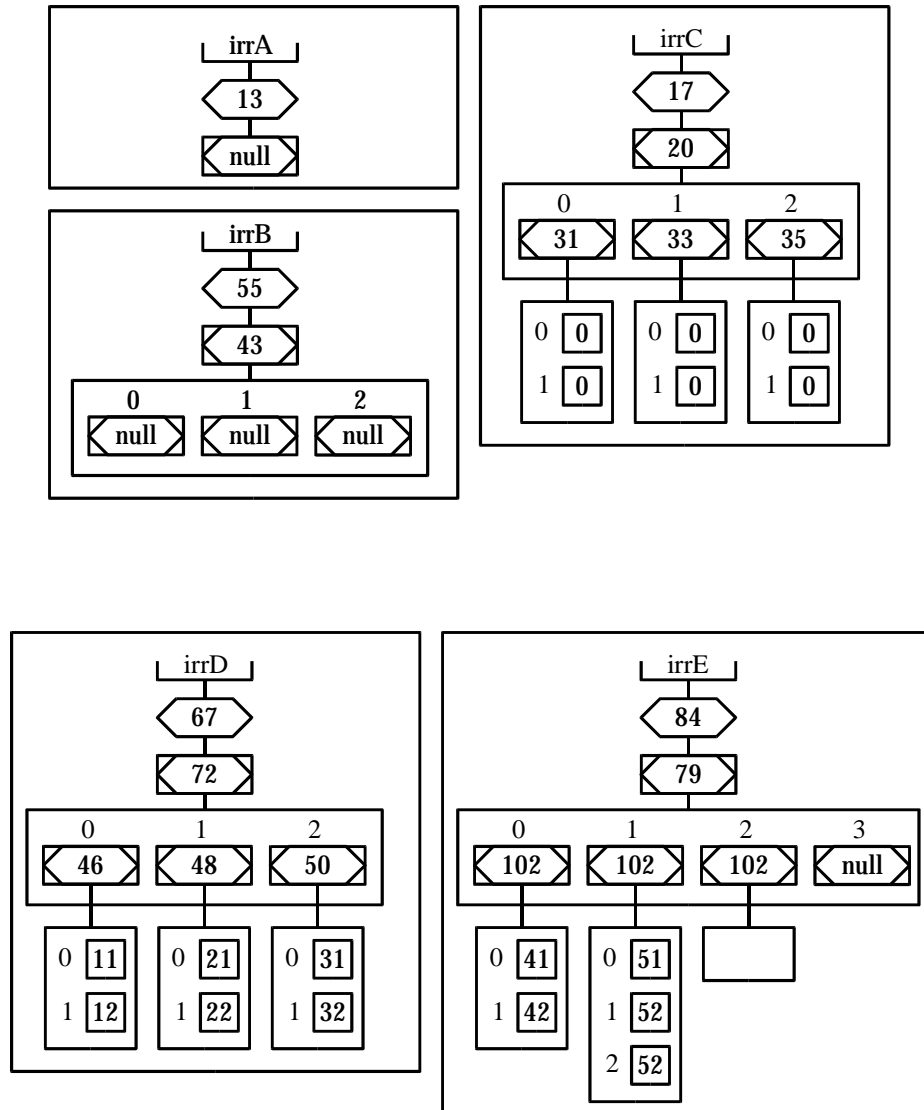
Betrachten Sie die folgenden fünf Vereinbarungen von zweistufigen Reihungsvariablen:

```
1 int[][] irrA = null; // Keine Reihung
2 int[][] irrB = new int[3][]; // 2-stufige Reihung
3 int[][] irrC = new int[3][2]; // 2-stufige Reihung
4 int[][] irrD = { {11, 12,}, // 2-stufige Reihung
5                 {21, 22,},
6                 {31, 32,},
7                 };
8 int[][] irrE = new int[][] { {41, 42,}, // 2-stufige Reihung
9                             {51, 52, 53},
10                            {},
11                            null,
12                            };
```

Stellen Sie die fünf Variablen `irrA` bis `irrE` als *Bojen* dar. Die *vereinfachte Bojendarstellung* (ohne die Referenzen von Komponentenvariablen und ohne das `length`-Attribut) ist hier ausdrücklich *erlaubt*.

**Lösung Reihungen 3:**

Die fünf zweistufigen Reihungsvariablen `irrA` bis `irrE` als (vereinfachte) Bojen dargestellt:



## Übung Bojen (ausführliche und vereinfachte) 1:

**Bojen** werden in den Abschnitten 5.7, 7.1 und 10.2 des Buches behandelt (die Begriffe ausführliche Bojendarstellung und vereinfachte Bojendarstellung werden im Abschnitt 7.1, S. 156-158 erklärt).

Nehmen Sie an, dass eine Klasse namens `Otto` wie folgt vereinbart wurde:

```
1 class Otto {
2     int    zahl;                // Ein primitives Objekt-Attribut
3     String text;                // Ein Referenz-Objekt-Attribut
4
5     Otto() {                    // Ein Standard-Konstruktor
6         zahl = -1;              // Nur zahl wird initialisiert,
7     } // Konstruktor Otto      // text behaelt den Wert null
8
9     Otto(int zahl, String text) { // Noch ein Konstruktor
10        this.zahl = zahl;
11        this.text = text;
12    } // Konstruktor Otto
13    ...
14 } // class Otto
15
```

Nehmen Sie weiter an, dass irgendwo drei `Otto`-Objekte vereinbart wurden, etwa wie folgt:

```
47    ...
48    Otto ob1 = new Otto(17, "Hallo ");
49    Otto ob2 = new Otto(25, "Sonja!" );
50    Otto ob3 = new Otto();
51    ...
52
```

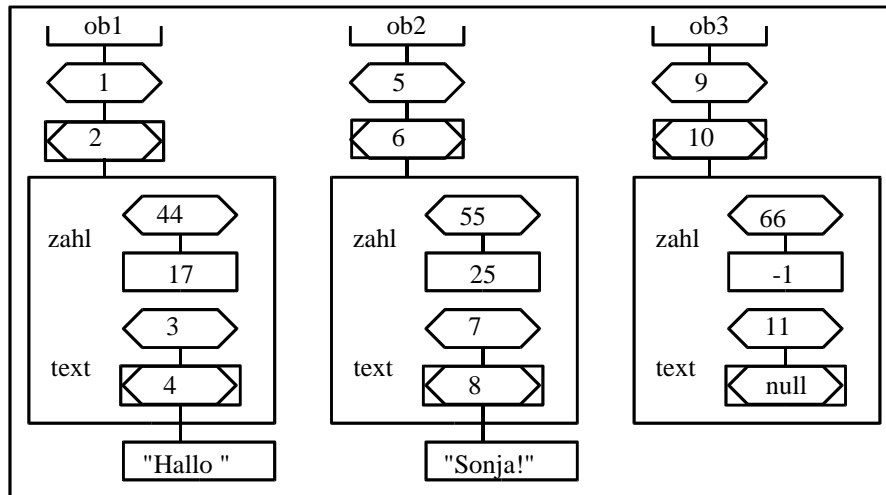
1. Stellen Sie die Variablen `ob1` bis `ob3` als Bojen dar, und zwar in *ausführlicher Darstellung* (d.h. zeichnen Sie von jedem Attribut die Referenz und den Wert).
2. Stellen Sie die Variablen `ob1` bis `ob3` als Bojen dar, diesmal in *vereinfachter Darstellung* (d.h. zeichnen Sie von jedem Attribut nur den Wert, aber nicht die Referenz).
3. Wie sehen die Variablen aus, nachdem die folgenden drei Zuweisungen ausgeführt wurden?

```
51    ...
52    ob1.zahl = ob2.zahl;
53    ob1.text = ob2.text;
54    ob3.text = ob1.text;
55    ...
```

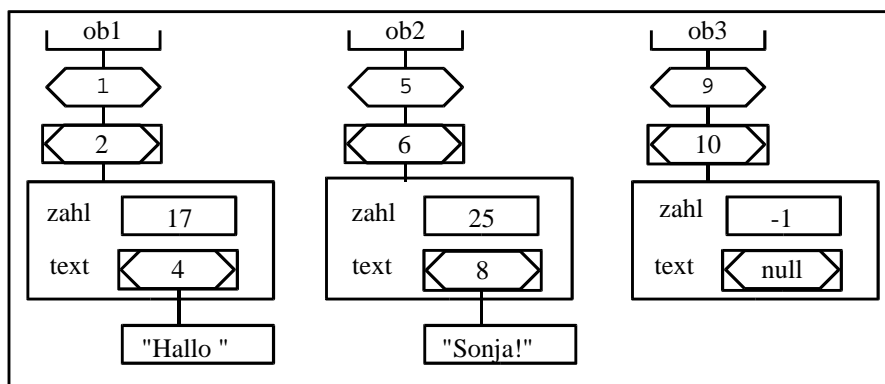
Zeichnen Sie die Variablen `ob1` bis `ob3` erneut in vereinfachter Bojendarstellung.

**Lösung Bojen (ausführliche und vereinfachte) 1:**

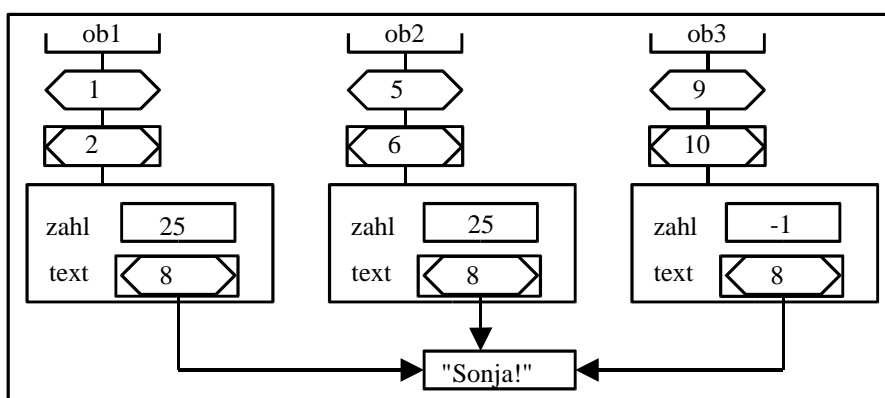
1. Die Variablen ob1 bis ob3 in ausführlicher Bojendarstellung:



2. Die Variablen ob1 bis ob3 in vereinfachter Bojendarstellung (ohne Referenzen von Attributen):



3. Die Variablen ob1 bis ob3 nach Ausführung der Zuweisungen (wieder in vereinfachter Bojendarstellung):



## Übung Bojen (von Reihungen) 2:

In den folgenden Programmfragmenten wird jeweils eine Reihung vereinbart und dann bearbeitet. Wie sieht diese Reihung *vor* ihrer Bearbeitung aus und wie sieht sie *nach* ihrer Bearbeitung aus? Stellen Sie jede Reihung zweimal als *Boje* dar (einmal *vor* ihrer Bearbeitung und einmal *nach* ihrer Bearbeitung).

```
1 // 1. Die Reihung ali:
2 int[] ali = new int[] {25, 50, 34, 87};
3 for (int i=ali.length-1; i>=0; i--) {
4     ali[i]++;
5 }
6
7 // 2. Die Reihung bernd:
8 int[] bernd = new int[] {12, 19, 23, 10, 15};
9 for (int i=1; i<bernd.length-1; i++) {
10     bernd[i] = (bernd[i-1] + bernd[i] + bernd[i+1]) / 3;
11 }
12
13 // 3. Die Reihung christian:
14 int[] christian = new int[] {6, 7, 8, 9};
15 for (int i=0; i<christian.length; i++) {
16     if (christian[i] % 2 == 0) {
17         christian[i] /= 2;
18     } else if (christian[i] % 3 == 0) {
19         christian[i] /= 3;
20     } else {
21         christian[i] = 0;
22     }
23 }
24
25 // 4. Die Reihung dirk:
26 StringBuilder[] dirk = {
27     new StringBuilder("Auf-"),
28     new StringBuilder("der-"),
29     new StringBuilder("Mauer.")
30 };
31 for (int i=1; i<dirk.length; i++) {
32     dirk[i].insert(0, dirk[i-1]);
33 }
34
35 // 5. Die Reihung ertan:
36 StringBuilder[] ertan = new StringBuilder[3];
37 ertan[0] = new StringBuilder("eins");
38 ertan[1] = new StringBuilder("zwei");
39 ertan[2] = new StringBuilder("drei");
40 ertan[1].append("mal");
41 ertan[0].setCharAt(0, 'E');
42 ertan[2] = new StringBuilder("vier");
43
44 // 6. Die Reihung frank:
45 int[] frank = new int[] {22, 78, 78, 35, 22};
46 for (int i1=0; i1<frank.length-1; i1++) {
47     for (int i2=i1+1; i2<frank.length; i2++) {
48         if (frank[i1] == frank[i2]) {
49             ++frank[i1];
50             --frank[i2];
51         }
52     }
53 }
```

## Lösung Bojen (von Reihungen) 2:

### 0. Ausgabe des Programms Ueb\_Reihungen03:

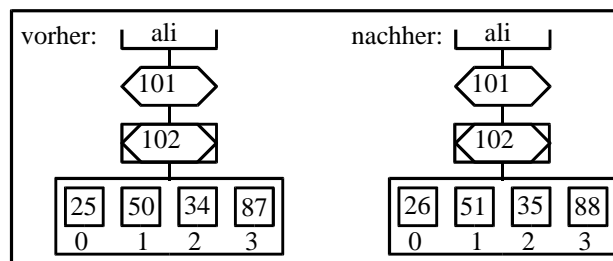
(diese Programmausgabe dient nur zum Überprüfen der folgenden Bojendarstellungen)

```

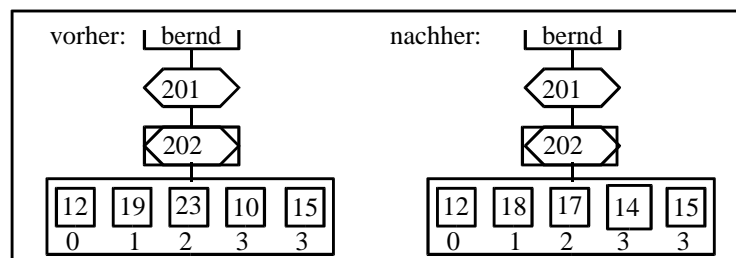
54 Ueb_Reihungen03: Jetzt geht es los!
55 Die Reihung ali      (vorher) : 25 | 50 | 34 | 87 |
56 Die Reihung ali      (nachher): 26 | 51 | 35 | 88 |
57 Die Reihung bernd    (vorher) : 12 | 19 | 23 | 10 | 15 |
58 Die Reihung bernd    (nachher): 12 | 18 | 17 | 14 | 15 |
59 Die Reihung christian (vorher) : 6 | 7 | 8 | 9 |
60 Die Reihung christian (nachher): 3 | 0 | 4 | 3 |
61 Die Reihung dirk     (vorher) : Auf- | der- | Mauer. |
62 Die Reihung dirk     (nachher): Auf- | Auf-der- | Auf-der-Mauer. |
63 Die Reihung ertan     (vorher) : null | null | null |
64 Die Reihung ertan     (mittendrin): eins | zwei | drei |
65 Die Reihung ertan     (nachher): Eins | zweimal | vier |
66 Die Reihung frank     (vorher) : 22 | 78 | 78 | 35 | 22 |
67 Die Reihung frank     (nachher): 23 | 79 | 77 | 35 | 21 |
68 Ueb_Reihungen03: Das war's erstmal!

```

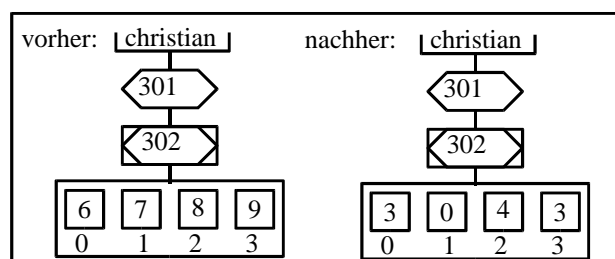
### 1. Die Reihung ali in Bojendarstellung:



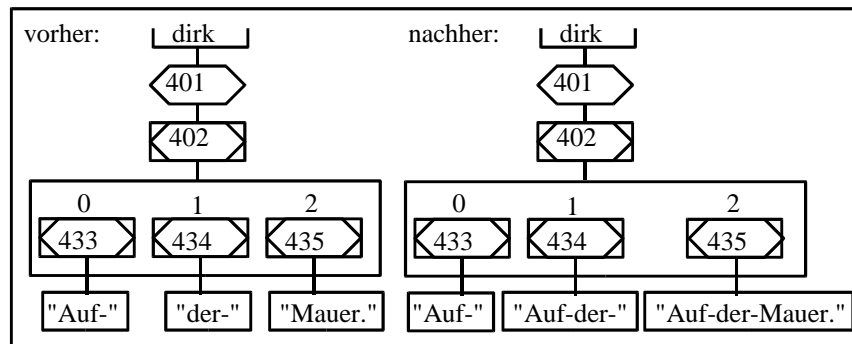
### 2. Die Reihung bernd in Bojendarstellung:



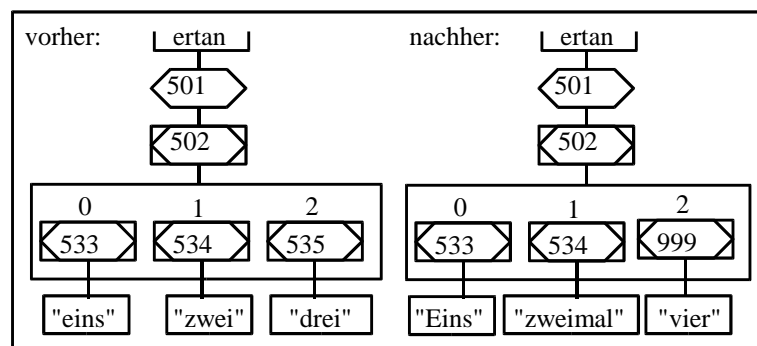
### 3. Die Reihung christian in Bojendarstellung:



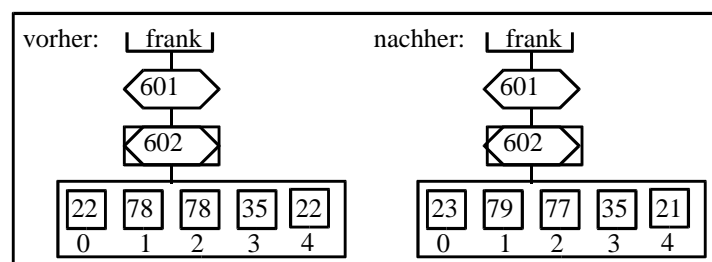
#### 4. Die Reihung dirk in Bojendarstellung:



#### 5. Die Reihung ertan in Bojendarstellung:



#### 6. Die Reihung frank in Bojendarstellung





## Übung Klassen 1:

**Klassen** und **klassische Fachbegriffe** werden im Kapitel 9 des Buches behandelt.

Betrachten Sie die folgende Klasse und beantworten sie dann die nachfolgenden Fragen:

```

1 // Datei Punkt2.java
2 // -----
3 class Punkt2 {
4     static private int    anzahlPunkte;
5     static private float  sp_x, sp_y;    // Schwerpunkt aller Punkte
6     // -----
7     static private void  rein(float x, float y) {
8         sp_x = (sp_x * anzahlPunkte + x) / (anzahlPunkte + 1);
9         sp_y = (sp_y * anzahlPunkte + y) / (anzahlPunkte + 1);
10        anzahlPunkte++;
11    } // rein
12    // -----
13    static private void  raus(float x, float y) {
14        if (anzahlPunkte == 1) {
15            sp_x = 0;
16            sp_y = 0;
17        } else {
18            sp_x = (sp_x * anzahlPunkte - x) / (anzahlPunkte - 1);
19            sp_y = (sp_y * anzahlPunkte - y) / (anzahlPunkte - 1);
20        }
21        anzahlPunkte--;
22    } // raus
23    // -----
24    public static void  verschiebe(Punkt2 p, float dx, float dy) {
25        pln("Verschiebe Punkt   von   x: " + p.x + ",   y: " + p.y);
26        pln("                      um   dx: " + dx + ", dy: " + dy);
27        raus(p.x, p.y);
28        p.x += dx;
29        p.y += dy;
30        rein(p.x, p.y);
31        pln("                      nach  x: " + p.x + ",   y: " + p.y);
32        pln("Neuer Schwerpunkt bei   x: " + sp_x + ",   y: " + sp_y);
33        pln();
34    } // verschiebe
35    // -----
36    private float x, y;
37    // -----
38    Punkt2(float neues_x, float neues_y) {
39        rein(neues_x, neues_y);
40        x = neues_x;
41        y = neues_y;
42        pln("Neuen Punkt erzeugt mit x: " + x + ",   y: " + y);
43        pln("Neue Anzahl aller Punkte: " + anzahlPunkte);
44        pln("Neuer Schwerpunkt bei   x: " + sp_x + ",   y: " + sp_y);
45        pln();
46    } // Konstruktor Punkt2
47    // -----
48    Punkt2() {
49        this(0.0f, 0.0f); // Aufruf eines anderen Konstruktors dieser Klasse
50    } // Konstruktor Punkt2
51    // -----
52    public void  verschiebe(float dx, float dy) {
53        pln("Verschiebe Punkt   von   x: " + x + ",   y: " + y);
54        pln("                      um   dx: " + dx + ", dy: " + dy);
55        raus(x, y);
56        x += dx;
57        y += dy;
58        rein(x, y);
59        pln("                      nach  x: " + x + ",   y: " + y);

```

```

60     pln("Neuer Schwerpunkt bei   x: " + sp_x + ",   y: " + sp_y);
61     pln();
62 } // verschiebe
63 // -----
64 // Mehrere Methoden mit kurzen Namen:
65 static void pln(Object ob) {System.out.println(ob);}
66 static void pln()          {System.out.println(); }
67 // -----
68 } // class Punkt2

```

### Fragen zur Klasse Punkt2:

Geben Sie als Antwort auf die Fragen 1 bis 7 jeweils die *Anzahl* und *die Namen* der betreffenden Elemente an (möglichst übersichtlich auf einem extra Blatt):

**Frage 1:** Alle Elemente der Klasse Punkt2 (Konstruktoren zählen *nicht* zu den Elementen!)

**Frage 2:** Klassenelemente

**Frage 3:** Objektelemente

**Frage 4:** Klassenattribute

**Frage 5:** Klassenmethoden

**Frage 6:** Objektattribute

**Frage 7:** Objektmethoden

**Frage 8:** Wieviele *Konstruktoren* hat die Klasse Punkt2? Wodurch unterscheiden sich diese Konstruktoren voneinander?

Die Fragen 9 bis 16 beziehen sich auf das folgende Programm:

```

1 // -----
2 public class Punkt2Tst {
3     public static void main(String[] sonja) {
4         Punkt2 p1 = new Punkt2(+1.0f, +4.0f);
5         Punkt2 p2 = new Punkt2(+2.0f, +3.0f);
6         Punkt2 p3 = new Punkt2(+3.0f, +2.0f);
7         Punkt2 p4 = new Punkt2(+4.0f, +1.0f);
8         Punkt2.verschiebe(p1, +3.0f, -3.0f);
9         Punkt2.verschiebe(p4, -3.0f, +3.0f);
10    } // main
11 } // class Punkt2Tst
12 // -----

```

Angenommen, der Ausführer hat das Programm Punkt2Tst bis Zeile 5 (einschliesslich) ausgeführt.

**Frage 9:** Wieviele *Module* existieren in diesem Moment und wie heissen diese Module?

**Frage 10:** Welchen *Wert* hat die Variable Punkt2.anzahlPunkte in diesem Moment?

**Frage 11:** Wieviele *Variablen* des Typs float existieren in diesem Moment? Wie heissen diese Variablen? (Für eine Variable namens Y in einem Modul namens X geben Sie als Namen bitte X.Y an).

Angenommen, der Ausführer hat das Programm Punkt2Tst bis Zeile 9 (einschliesslich) ausgeführt.

**Frage 12:** Wieviele *Module* existieren in diesem Moment und wie heissen diese Module?

**Frage 13:** Welchen *Wert* hat die Variable Punkt2.anzahlPunkte in diesem Moment?

**Frage 14:** Wieviele *Variablen* des Typs float existieren in diesem Moment? Wie heissen diese Variablen? (Für eine Variable namens Y in einem Modul namens X geben Sie als Namen bitte X.Y an).

**Frage 15:** Wieviele *Methoden* namens verschiebe gibt es in diesem Moment und wie heissen diese Methoden mit vollem Namen (gemeint sind hier Namen der Form X.verschiebe)?

**Frage 16:** Was gibt das Programm Punkt2Tst zum Bildschirm aus? Führen Sie das Programm mit Papier und Bleistift aus und ermitteln Sie wenigstens die ersten Zeilen der Ausgabe.

**Lösung Klassen 1:**

1. Alle Elemente der Klasse Punkt2: (11 Elemente): anzahlPunkte, sp\_x, sp\_y, rein, raus, verschiebe, x, y, verschiebe, pln, pln.
2. Klassenelemente (8 Stk.): anzahlPunkte, sp\_x, sp\_y, rein, raus, verschiebe, pln, pln.
3. Objektelemente (3 Stk): x, y, verschiebe
4. Klassenattribute (3 Stk): anzahlPunkte, sp\_x, sp\_y
5. Klassenmethoden (5 Stk): rein, raus, verschiebe, pln, pln
6. Objektattribute (2 Stk): x, y
7. Objektmethoden (1 Stk): verschiebe
8. Wieviele **Konstruktoren** hat die Klasse Punkt2? Wodurch unterscheiden sich diese Konstruktoren voneinander? Zwei. Der erste Konstruktor hat **0 Parameter**, der zweite hat **2 Parameter**.
9. Wieviele Module existieren in diesem Moment und wie heissen diese Module? **Vier** Module. Sie heissen Punkt2Tst, Punkt2, p1 und p2.
10. Welchen Wert hat die Variable Punkt2.anzahlPunkte in diesem Moment? Den **Wert 2**.
11. Wieviele Variablen des Typs float existieren in diesem Moment? Wie heissen diese Variablen? Es exist. **6** float-Variablen namens Punkt2.sp\_x, Punkt2.sp\_y, p1.x, p1.y, p2.x, p2.y
12. Wieviele Module existieren in diesem Moment und wie heissen diese Module? **Sechs** Module. Sie heissen Punkt2Tst, Punkt2, p1, p2, p3 und p4.
13. Welchen Wert hat die Variable Punkt2.anzahlPunkte in diesem Moment? Den **Wert 4**.
14. Wieviele Variablen des Typs float existieren in diesem Moment? Wie heissen diese Variablen? Es exist. **10** float-Variablen namens Punkt2.sp\_x, Punkt2.sp\_y, p1.x, p1.y, ..., p4.x, p4.y
15. Wieviele Methoden namens verschiebe gibt es in diesem Moment und wie heissen diese Methoden mit vollem Namen? Es existieren **5** verschiebe-Methoden. Sie heissen (mit vollen Namen) Punkt2.verschiebe, p1.verschiebe, p2.verschiebe, p3.verschiebe und p4.verschiebe.
16. Was gibt das Programm Punkt2Tst zum Bildschirm aus?

```

13 Neuen Punkt erzeugt mit x: 1.0, y: 4.0
14 Neue Anzahl aller Punkte: 1
15 Neuer Schwerpunkt bei x: 1.0, y: 4.0
16
17 Neuen Punkt erzeugt mit x: 2.0, y: 3.0
18 Neue Anzahl aller Punkte: 2
19 Neuer Schwerpunkt bei x: 1.5, y: 3.5
20
21 Neuen Punkt erzeugt mit x: 3.0, y: 2.0
22 Neue Anzahl aller Punkte: 3
23 Neuer Schwerpunkt bei x: 2.0, y: 3.0
24
25 Neuen Punkt erzeugt mit x: 4.0, y: 1.0
26 Neue Anzahl aller Punkte: 4
27 Neuer Schwerpunkt bei x: 2.5, y: 2.5
28
29 Verschiebe Punkt von x: 1.0, y: 4.0
30 um dx: 3.0, dy: -3.0
31 nach x: 4.0, y: 1.0
32 Neuer Schwerpunkt bei x: 3.25, y: 1.75
33
34 Verschiebe Punkt von x: 4.0, y: 1.0
35 um dx: -3.0, dy: 3.0
36 nach x: 1.0, y: 4.0
37 Neuer Schwerpunkt bei x: 2.5, y: 2.5

```

## Übung Deutsch 1:

Kleinere Befehle werden im **Abschnitt 1.5** des Buches ins Deutsche übersetzt. Hier wird zum ersten Mal eine ganze Klasse übersetzt. Betrachten Sie dazu die folgende Vereinbarung einer Klasse:

```

1 class Karoline {
2     private static int    anna  = 17;
3     public static float   berta = 12.34;
4     private              int    carl  = 25;
5     private              float   dirk;
6     public static void    erika() {
7         anna = (anna * 2) / 3;
8         pln("Hallo! " + anna + berta);
9     }
10    public              int    fritz(int n) {
11        carl = carl * 2 / 3;
12        return carl;
13    }
14 } // class Karoline

```

Diese Vereinbarung ist ein **Befehl** (des Programmierers an den Ausführer), den man etwa so ins Deutsche übersetzen kann:

Erzeuge eine Klasse namens `Karoline`, die die folgenden Vereinbarungen enthält:

1. Vereinbarungen von Klassenelementen:

1.1. Erzeuge eine `private` Variable namens `anna` vom Typ `int` mit dem Anfangswert 17.

1.2. Erzeuge eine öffentliche Variable namens `berta` vom Typ `float` mit dem Anfangswert 12.34.

1.3. Erzeuge eine Methode namens `erika`, die keine Parameter hat und kein Ergebnis liefert und aus den folgenden Befehlen besteht:

1.3.1. Berechne den Wert des Ausdrucks  $(anna * 2) / 3$  und tue diesen Wert in die Variable `anna`.

1.3.2. Berechne den Wert des Ausdrucks `"Hallo! " + anna + berta`, nimm diesen Wert als Parameter und führe damit die Methode `println` aus dem Objekt `out` aus der Klasse `System` aus.

2. Vereinbarungen von Objektelementen:

2.1. Erzeuge eine `private` Variable namens `carl` vom Typ `int` mit dem Anfangswert 25.

2.2. Erzeuge eine `private` Variable namens `dirk` vom Typ `float` (mit dem Standardanfangswert 0.0).

2.3. Erzeuge eine öffentliche Methode namens `fritz`, die einen `int`-Parameter namens `n` hat, ein Ergebnis vom Typ `int` liefert und aus den folgenden Befehlen besteht:

2.3.1. Berechne den Wert des Ausdrucks  $n * 2 / 3$  und tue ihn in die Variable `carl`.

2.3.2. Berechne den Wert des Ausdrucks `carl` und liefere ihn als Ergebnis der Methode `fritz`.

Offensichtlich ist die deutsche Version dieses Befehls deutlich länger als die Java-Version (das war auch ein Grund dafür, Java zu erfinden statt Programme auf Deutsch zu schreiben).

Übersetzen Sie entsprechend die folgende Klassenvereinbarung ins Deutsche:

```

15 class Yehyahan {
16     private          long    wisam;
17     public           double   erdogan = 99.9;
18     private static String   selemon = "Hallo! ";
19     public           void     ertan(String s) {
20         wisam = erdogan / 3.0 + 17;
21         pln(selemon + s);
22     }
23     public           void     selcuk() {
24         pln("Ihr Name?");
25         String name = EM.liesString();
26         ertan(name);
27     }
28     public           long     raed() {
29         return wisam;
30     }
31 } // class Yehyahan

```

**Lösung Deutsch 1:**

Übersetzung der Vereinbarung der Klasse `Yehyahan` ins Deutsche:

Erzeuge ein Klasse namens `Yehyahan`, die folgende Vereinbarungen enthält:

1. Vereinbarungen von *Klassenelementen*:

1.1. Erzeuge eine `private String`-Variab. namens `selemon` mit dem Anfangswert `"Hallo! "`.

2. Vereinbarungen von *Objektelementen*:

2.1. Erzeuge eine `long`-Variable namens `wisam`.

2.2. Erzeuge eine `double`-Variable `erdogan` mit dem Anfangswert `99.9`.

2.3. Erzeuge eine öffentliche Methode namens `ertan`, die einen `String`-Parameter namens `s` hat, kein Ergebnis liefert und aus den folgenden Befehlen besteht:

2.3.1. Berechne den Wert des Ausdrucks `erdogan / 3.0 + 17` und weise ihn der Variablen `wisam` zu.

2.3.2. Berechne den Wert des Ausdrucks `selemon + s`, nimm ihn als Parameter und führe damit die Methode `println` aus dem Objekt `out` aus der Klasse `System` aus.

2.4. Erzeuge eine öffentliche Methode namens `selcuk`, die keine Parameter hat und kein Ergebnis liefert und aus den folgenden Befehlen besteht:

2.4.1. Berechne den Wert des Ausdrucks `"Ihr Name?"`, nimm ihn als Parameter und führe damit die Methode `println` aus.

2.4.2. Berechne den Wert des Ausdrucks `EM.liesString()` und erzeuge eine `String`-Variable namens `name` mit diesem Wert als Anfangswert.

2.4.3. Berechne den Wert des Ausdrucks `name`, nimm ihn als Parameter und führe damit die Methode `ertan` (die sich im aktuellen Objekt `this` befindet) aus.

2.5. Erzeuge eine öffentliche Methode namens `raed`, die keine Parameter hat, ein Ergebnis vom Typ `long` liefert und aus dem folgenden Befehl besteht:

2.5.1. Berechne den Wert des Ausdrucks `wisam` und liefere ihn als Ergebnis der Methode `raed`.

## Übung Klassen 2:

1. Führen Sie die folgende Befehlsfolge mit Papier und Bleistift (notfalls mit Papier und einem Kuli :-)) aus. Geben Sie als Lösung dieser Aufgabe an, welche Werte die Komponenten der Reihung `otto` nach Ausführung aller Befehle haben:

```
1 int[] otto = new int[] {10, 17, 24, 31, 14, 27};
2 for (int i=1; i < otto.length-2; i++) {
3     if (otto[i] % 3 != 0) {
4         otto[i]++;
5         i--;
6     }
7 }
```

2. Betrachten Sie die folgende Klasse `EM00`:

```
8 // Datei EM00.java
9 //-----
10 // Ein Modul. der Methoden zum Einlesen von der Standardeingabe zur
11 // Verfuegung stellt. Eingelesen werden koennen Werte der folgenden Typen:
12 // String, int, float und boolean.
13 //-----
14 import java.io.InputStreamReader;
15 import java.io.BufferedReader;
16 import java.io.IOException;
17
18 public class EM00 {
19     // -----
20     // Stellt den Kollegen Unterprogramme zur Verfuegung, mit denen man
21     // Strings, Ganzzahlen, Bruchzahlen und Wahrheitswerte von der
22     // Standardeingabe einlesen kann.
23     // -----
24     static public String liesString() throws IOException {
25         return Bernd.readLine();
26     }
27     // -----
28     static public int liesInt() throws IOException {
29         return Integer.parseInt(Bernd.readLine());
30     }
31     // -----
32     static public float liesFloat() throws IOException {
33         return Float.parseFloat(Bernd.readLine());
34     }
35     // -----
36     static public boolean liesBoolean() throws IOException {
37         return Boolean.valueOf(Bernd.readLine()).booleanValue();
38     }
39     // -----
40     static InputStreamReader Inge = new InputStreamReader(System.in);
41     static BufferedReader Bernd = new BufferedReader(Inge);
42     // -----
43 } // end class EM00
```

Geben Sie als Antwort auf die Fragen 1 bis 4. jeweils die *Anzahl* und die *Namen* der betreffenden Elemente an:

1. Klassenattribute
2. Klassenmethoden
3. Objektattribute
4. Objektmethoden

5. Schreiben Sie zusätzliche Methoden namens `liesShort`, `liesLong` und `liesDouble`, die man zur Klasse `EM00` hinzufügen könnte. Mit diesen zusätzlichen Methoden soll man einen Wert vom Typ `short` (bzw. `long` bzw. `double`) von der Standardeingabe einlesen können. Orientieren Sie sich

beim Schreiben der zusätzlichen Methoden an der Methode `liesInt` (und nicht an `liesString` oder `liesBoolean`).

6. Betrachten Sie die Klasse `IntRech01` (siehe unten). Programmieren Sie ein Klassen namens `IntRech02` als Erweiterung (d.h. als Unterklasse) der Klasse `IntRech01`. Die neue Klasse `IntRech02` soll zusätzliche Methoden namens `mul` und `div` enthalten, mit denen man `int`-Werte *sicher* multiplizieren bzw. dividieren kann.

7. Schreiben Sie ein Programm namens `BigInteger03`, welches eine `int`-Zahl `n` von der Standardeingabe einliest, die Fakultät von `n` (das Produkt aller Ganzzahlen von 1 bis `n`, einschliesslich) als Objekt des Typs `BigInteger` berechnet und zur Standardausgabe ausgibt. Orientieren Sie sich beim Schreiben dieses Programms am Programm `BigInteger01` (siehe unten und bei den Beispielprogrammen). Zur Vereinfachung dieser Aufgabe sei festgelegt: Falls die eingelesene `int`-Zahl `n` kleiner oder gleich 0 ist, soll als Ergebnis die Zahl 1 ausgegeben werden.

**Erweiterung der vorigen Übung:**

8. Lesen Sie (in Ihrem Programm `BigInteger03`) wiederholt `int`-Werte und geben Sie deren Fakultät aus, bis der Benutzer eine 0 eingibt.

Falls der Benutzer eine *negative* Ganzzahl eingibt, sollte eine kleine Fehlermeldung ausgegeben und das Programm fortgesetzt werden.

Geben Sie nicht nur die Fakultät selber aus, sondern zusätzlich auch die *Anzahl ihrer Dezimalziffern* (dazu gibt es in der Klasse `BigInteger` hilfreiche Methoden).

Geben Sie zusätzlich auch noch aus, wieviele *Binärziffern* man braucht, um die gerade ausgegebene Fakultät darzustellen. Siehe dazu die Methode `bitLength` in der Klasse `BigInteger`.

Das Programm `IntRech01`:

```

44 public class IntRech01 {
45     // -----
46     static public int add(int n1, int n2) throws ArithmeticException {
47         long erg = (long) n1 + (long) n2; // Hier wird sicher addiert!
48         pruefeObInt(erg); // Falls noetig wird eine Ausnahme ausgeloeset!
49         return (int) erg;
50     } // add
51     // -----
52     static public int sub(int n1, int n2) throws ArithmeticException {
53         long erg = (long) n1 - (long) n2; // Hier wird sicher subtrahiert!
54         pruefeObInt(erg); // Falls noetig wird eine Ausnahme ausgeloeset!
55         return (int) erg;
56     } // sub
57     // -----
58     static protected void pruefeObInt(long g) throws ArithmeticException {
59         // Loest eine Ausnahme ArithmeticException aus, falls man g nicht
60         // in einen int-Wert umwandeln kann.
61         if (g < Integer.MIN_VALUE || Integer.MAX_VALUE < g) {
62             throw new ArithmeticException(g + " ist kein int-Wert!");
63         }
64     } // pruefeObInt
65     // -----
66     static public void main(String[] sonja) {
67         // Kleiner Test der Methoden add und sub:
68         int int01 = 1000 * 1000 * 1000; // 1 Milliarden
69         int int02 = 2000 * 1000 * 1000; // 2 Milliarden
70
71         pln("A int02 - int01: " + sub(int02, int01));
72         pln("B int02 + 12345: " + add(int02, 12345));
73         pln("C int02 + int01: " + add(int02, int01));
74     } // main
75     // -----
76 } // class IntRech01

```

## Das Programm BigInteger01:

```
1 import java.math.BigInteger;
2
3 public class BigInteger01 {
4     // -----
5     static public void main(String[] emil) throws
6         java.io.IOException,
7         java.lang.ArithmeticException,
8         java.lang.NumberFormatException
9     {
10         pln("A BigInteger01: Jetzt geht es los!");
11
12         while (true) {
13             pln("B Zum Beenden bitte 0 eingeben!");
14             p ("C BigInteger BI1? ");
15             BigInteger bi1 = EM.liesBigInteger();
16             if (bi1.compareTo(BigInteger.ZERO) == 0) break;
17             p ("D BigInteger BI2? ");
18             BigInteger bi2 = EM.liesBigInteger();
19
20             pln("E BI1:          " + bi1);
21             pln("F BI2:          " + bi2);
22
23             pln("G BI1 + BI2:      " + bi1.add(bi2));
24             pln("H BI1 - BI2:      " + bi1.subtract(bi2));
25             pln("I BI1 * BI2:      " + bi1.multiply(bi2));
26             pln("J BI1 / BI2:      " + bi1.divide(bi2));
27
28             // Die Methode divideAndRemainder liefert eine Reihung mit zwei
29             // Komponenten vom Typ BigInteger: den Quotienten und den Rest.
30             BigInteger[] bir = bi1.divideAndRemainder(bi2);
31             pln("K BI1 d BI2:      " + bir[0]); // d wie divide
32             pln("L BI1 r BI2:      " + bir[1]); // r wie remainder
33
34             // Die Methode mod wirft eine Ausnahme ArithmeticException, wenn
35             // der zweite Operand (der Modulus) negativ ist. Man beachte den
36             // subtilen Unterschied zwischen den beiden Restfunktionen rem
37             // (d.h. divideAndRemainder) und mod!
38             pln("M BI1 mod BI2:      " + bi1.mod(bi2));
39         } // while
40         pln("N BigInteger01: Das war's erstmal!");
41     } // end main
```



**Lösung Klassen 2:**

1. Nach Ausführung aller Befehle haben die Komponenten der Reihung otto folgende Werte:  
10, 15, 24, 30, 14, 27.

Anzahl und Namen verschiedener Elemente der Klasse Lesen:

1. Klassenattribute: 2 Stück: Inge, Bernd
2. Klassenmethoden: 4 Stück: liesString, liesInt, liesFloat, liesBoolean
3. Objektattribute: 0
4. Objektmethoden: 0
5. Die zusätzlichen Klassenmethoden für die Klasse Lesen:

```

1 // -----
1 static public short  liesShort() throws IOException {
2     return Short.parseShort(Bernd.readLine());
3 }
4 // -----
5 static public long   liesLong() throws IOException {
6     return Long.parseLong(Bernd.readLine());
7 }
8 // -----
9 static public double liesDouble() throws IOException {
10    return Double.parseDouble(Bernd.readLine());
11 }

```

**6. Die Klasse IntRech02:**

```

12 // Datei IntRech02.java
13 // -----
14 // Die Klasse IntRech01 wird um Methoden zum sicheren Multiplizieren und
15 // Dividieren erweitert zur Klasse IntRech02.
16 // -----
17 class IntRech02 extends IntRech01 {
18     // -----
19     static public int mul(int i1, int i2) throws ArithmeticException {
20         long n1 = i1;
21         long n2 = i2;
22         long erg = n1 * n2; // Hier wird sicher multipliziert!
23         pruefeObInt(erg);   // Falls noetig wird eine Ausnahme ausgeloeset!
24         return (int) erg;
25     } // mul
26     // -----
27     static public int div(int i1, int i2) throws ArithmeticException {
28         long n1 = i1;
29         long n2 = i2;
30         long erg = n1 / n2; // Hier wird sicher dividiert!
31         pruefeObInt(erg);   // Falls noetig wird eine Ausnahme ausgeloeset!
32         return (int) erg;
33     } // div
34     // -----
35 } // class IntRech02
36

```

## 7. Hier das Programm BigInteger03:

```
1 // Datei BigInteger03.java
2 // -----
3 // Liest eine Ganzzahl vom Typ int ein und gibt deren Fakultaet aus (als
4 // Zahl vom Typ BigInteger). Gibt ausserdem die Laenge des Ergebnisses (die
5 // Anzahl der Dezimalziffern und die Anzahl der noetigen Binaerziffern) aus.
6 // -----
7 import java.math.BigInteger;
8
9 class BigInteger03 {
10     static public void main(String[] sonja) throws java.io.IOException {
11         pln("BigInteger03: Jetzt geht es los!");
12
13         while (true) {
14             p ("Eine Ganzzahl n (0 zum Beenden): ");
15             int n = EM.liesInt();
16             if (n == 0) break;
17             if (n < 0) {
18                 pln("Die Zahl " + n + " ist zu klein!");
19                 continue;
20             }
21
22             BigInteger erg = new BigInteger("1");
23             BigInteger faktor = new BigInteger("1");
24
25             for (int i=1; i < n; i++) {
26                 faktor = faktor.add(BigInteger.ONE);
27                 erg = erg.multiply(faktor);
28             }
29
30             String ergString = erg.toString();
31             p ("Die Fakultaet von " + n + " ist gleich: ");
32             pln(ergString);
33             int laengeD = ergString.length();
34             int laengeB = erg.bitLength();
35             pln("Laenge (in Dezimalziffern)      : " + laengeD);
36             pln("Laenge (in Binaerziffern)         : " + laengeB);
37         } // while
38
39         pln("BigInteger03: Das war's erstmal!");
40     } // main
41 } // class BigInteger03
42 /* -----
```

## Übung Klassen 3:

**Vererbung** zwischen Klassen wird im **Kapitel 12** des Buches behandelt.

1. Betrachten Sie die folgenden Vereinbarungen der drei Klassen A, B und C:

```
1 class A          {int    n; ...}
2 class B extends A {float  f; ...}
3 class C extends A {double d; ...}
```

Welche der folgenden Sätze sind wahr (richtig, korrekt) und welche sind falsch?

- 1.01. A ist *eine* Unterklasse von B.      1.02. A ist *die* Unterklasse von B.
- 1.03. B ist *eine* Unterklasse von A.      1.04. B ist *die* Unterklasse von A.
- 1.05. A ist *eine* direkte Oberklasse von B.      1.06. A ist *die* direkte Oberklasse von B.
- 1.07. B ist *eine* direkte Oberklasse von A.      1.08. B ist *die* direkte Oberklasse von A.
- 1.09. Die Klasse A *enthält mehr Elemente* als die Klasse B.
- 1.10. Die Klasse B *enthält mehr Elemente* als die Klasse A.
- 1.11. Jedes Objekt der Klasse A ist auch ein Objekt der Klasse B.
- 1.12. Jedes Objekt der Klasse B ist auch ein Objekt der Klasse A.
- 1.13. Zur Klasse A gehören immer (gleich viel oder) mehr Objekte als zu B.
- 1.14. Zur Klasse B gehören immer (gleich viel oder) mehr Objekte als zu A.

2. Diese Aufgabe bezieht sich auf die Klassen E01Punkt, E01Quadrat etc. aus dem Abschnitt 12.3 des Buches.

```
4 package Erben;
5 class Test1 {
6     static public main main(String[] sonja) {
7         E01Punkt p1 = new Quadrat(0.0, 0.0, 1.0);
8         ...
9         p1 = new E01Rechteck(0.5, 1.5, 2.5, 4.0);
10        ...
11        p1 = new E01Punkt(1.0, 0.0);
12        ...
13        p1 = new E01Kreis(0.0, 1.0, 2.0);
14        ...
15    }
```

2.1. Welchen *vereinbarten* Typ hat die Variable p1?

2.2. Welchen *momentanen* Typ hat die Variable p1, wenn der Ausführer gerade damit fertig ist, die **Zeile 4** auszuführen?

2.3. Ebenso für **Zeile 6**.    2.4. Ebenso für **Zeile 8**.    2.5. Ebenso für **Zeile 10**.

2.6. In welche der Zeilen 5, 7, 9 und 11 dürfte man die folgende Vereinbarung schreiben:

```
String t = p1.text();
```

und welchen *Zielwert* (nicht Wert!) hätte t jeweils?

2.7. In welche der Zeilen 5, 7, 9 und 11 dürfte man die folgende Vereinbarung schreiben:

```
String s = p1.toString();
```

und welchen *Zielwert* (nicht Wert!) hätte s jeweils?

2.8. In welche der Zeilen 5, 7, 9 und 11 dürfte man die folgende Vereinbarung schreiben:

```
double u = ((Quadrat) p1).getUmfang();
```

und welchen *Wert* (nicht Zielwert!) hätte u jeweils?

**Lösung Klassen 3:**

1. Nur die folgenden Sätze sind wahr (richtig, korrekt):

1.03. B ist *eine* Unterklasse von A.

1.06. A ist *die* direkte Oberklasse von B.

1.10. Die Klasse *B* enthält mehr Elemente als die Klasse A.

1.12. Jedes Objekt der Klasse *B* ist auch ein Objekt der Klasse A.

1.13. Zur Klasse A gehören immer (gleich viel oder) mehr Objekte als zu B.

2.1. Die Variable *p1* hat den vereinbarten Typ *Punkt*.

2.2. Nach Zeile 4 hat *p1* den momentanen Typ *Quadrat*.

2.3. Nach Zeile 6 hat *p1* den momentanen Typ *Rechteck*.

2.4. Nach Zeile 8 hat *p1* den momentanen Typ *Punkt*.

2.5. Nach Zeile 10 hat *p1* den momentanen Typ *Kreis*.

2.6.

Vereinbart in Zeile 5 hätte *t* den Zielwert "(0.0, 0.0)"

Vereinbart in Zeile 7 hätte *t* den Zielwert "(0.5, 1.5)"

Vereinbart in Zeile 9 hätte *t* den Zielwert "(1.0, 0.0)"

Vereinbart in Zeile 11 hätte *t* den Zielwert "(0.0, 1.0)"

2.7.

Vereinbart in Zeile 5 hätte *s* den Zielwert

"Quadrat, Mittelpunkt bei: (0.0, 0.0), Seitenlaenge: 2.0"

Vereinbart in Zeile 7 hätte *s* den Zielwert

"Rechteck, Mittelpunkt bei: (0.5, 1.5), x-Seite: 5.0, y-Seite: 8.0"

Vereinbart in Zeile 9 hätte *s* den Zielwert

"Punkt: (1.0, 0.0)"

Vereinbart in Zeile 11 hätte *s* den Zielwert

"Kreis, Mittelpunkt bei: (0.0, 1.0), Radius: 2.0"

2.8.

Vereinbart in Zeile 5 hätte *u* den Wert 8.0

Vereinbart in Zeile 7 hätte *u* den Wert 26.0 (und nicht etwa  $8 \cdot 2.5$  gleich 20.0!)

In den Zeile 9 und 11 darf *u* nicht vereinbart werden, weil *E01Punkt* und *E01Kreis* keine Untertypen von *E01Quadrat* sind.

## Übung Strings 1:

Die Klasse `String` wird im Abschnitt 10.1 des Buches behandelt.

1. Wieviele Konstruktoren hat die Klasse `StringBuilder`? Schauen Sie in ihrer Lieblingsdokumentation der Java-Standardklassen nach.
2. Wieviele Konstruktoren hat die Klasse `String`?
3. In der Klasse `String` gibt es eine Methode `String valueOf(char[] data)`. Geben Sie eine Befehlsfolge an, in der diese Methode aufgerufen wird.
4. Ebenso für die Methode `char charAt(int index)`.
5. Welche Zielwerte haben die `String`-Variablen `s1` bis `s3` nach den folgenden Vereinbarungen:

```
1 String sonja = new String("Hallo!");
2 String s1    = sonja.substring(0, sonja.length());
3 String s2    = sonja.substring(1, 6);
4 String s3    = sonja.substring(1, 1);
```

6. Ergänzen Sie das folgende "Skelett" zu einer Methode:

```
5 String wiederhole(char zeichen, int anzahl) {
6 // Liefert einen String, der aus anzahl vielen zeichen besteht.
7 // Falls anzahl kleiner oder gleich 0 ist, wird ein leerer String
8 // als Ergebnis geliefert.
9     ...
10 } // wiederhole
```

7. Schreiben Sie drei Methoden, die den folgenden Skeletten entsprechen:

```
11 static public String linksBuendig(String s, int len) {
12 // Falls s.length groesser oder gleich len ist, wird s unveraendert
13 // als Ergebnis geliefert. Sonst werden an der rechten Seite von s
14 // soviele Blanks angehaengt, dass ein String der Laenge len entsteht.
15 // Dieser wird als Ergebnis geliefert.
16     ...
17 }
18
19 static public String rechtsBuendig(String s, int len) {
20 // Falls s.length groesser oder gleich len ist, wird s unveraendert
21 // als Ergebnis geliefert. Sonst werden an der linken Seite von s
22 // soviele Blanks angehaengt, dass ein String der Laenge len entsteht.
23 // Dieser wird als Ergebnis geliefert.
24     ...
25 }
26
27 static public String zentriert(String s, int len) {
28 // Falls s.length groesser oder gleich len ist, wird s unveraendert
29 // als Ergebnis geliefert. Sonst werden links und rechts von s soviele
30 // Blanks angehaengt, dass ein String der Laenge len entsteht. Dieser
31 // wird als Ergebnis geliefert. Falls moeglich werden links und rechts von
32 // s gleich viel Blanks angehaengt, und sonst rechts eins mehr als links.
33     ...
34 }
```

## Lösung Strings 1:

1. Die Klasse `StringBuilder` hat **4 Konstruktoren**.

2. Die Klasse `String` hat **13 Konstruktoren**, davon zwei "Auslaufmodelle" (deprecated), die man nicht mehr verwenden soll.

3. Eine Befehlsfolge mit `valueOf`:

```
1 char[] otto = {'H', 'a', 'l', 'l', 'o', '!'};
2 String emil = String.valueOf(otto);
```

4. Eine Befehlsfolge mit `charAt`:

```
3 String sonja = new String("Hallo!");
4 char c = sonja.charAt(1);
```

5. Zielwerte der Stringvariablen: `s1` hat den Zielwert "Hallo!"  
`s2` hat den Zielwert "allo!"  
`s3` hat den Zielwert ""

6. Die Methode wiederhole:

```
5 static String wiederhole(char zeichen, int anzahl) {
6     // Liefert einen String, der aus anzahl vielen zeichen besteht.
7     // Falls anzahl kleiner oder gleich 0 ist, wird ein leerer
8     // String als Ergebnis geliefert.
9
10    if (anzahl <= 0) return "";
11    char[] erg = new char[anzahl];
12    for (int i=0; i<erg.length; i++) erg[i] = zeichen;
13    // Oder:
14    // Arrays.fill(erg, zeichen);
15    return new String(erg);
16 }
```

7. Die Methoden `linksBuendig`, `rechtsBuendig` und `zentriert`:

```
17 static String linksBuendig(String s, int len) {
18     // Liefert eine Kopie von s, die die Laenge len hat. Falls noetig
19     // werden rechts Blanks angehaengt. Falls s.length() groesser oder
20     // gleich len ist, wird eine (unveraenderte) Kopie von s geliefert.
21     if (s.length() >= len) return s;
22     String blanks = wiederhole(' ', len - s.length());
23     return s + blanks;
24 }
25
26 static String rechtsBuendig(String s, int len) {
27     // Liefert eine Kopie von s, die die Laenge len hat. Falls noetig
28     // werden links Blanks angehaengt. Falls s.length() groesser oder
29     // gleich len ist, wird eine (unveraenderte) Kopie von s geliefert.
30     if (s.length() >= len) return s;
31     String blanks = wiederhole(' ', len - s.length());
32     return blanks + s;
33 }
34 static String zentriert(String s, int len) {
35     // Liefert eine Kopie von s, die die Laenge len hat. Falls noetig
36     // werden links und rechts Blanks angehaengt (im Falle einer ungeraden
37     // Anzahl rechts eins mehr als links). Falls s.length() groesser oder
38     // gleich len ist, wird eine (unveraenderte) Kopie von s geliefert.
39     if (s.length() >= len) return s;
40     String blanks = wiederhole(' ', len - s.length());
41     int halb = blanks.length() / 2;
42     String linksb = blanks.substring(0, halb);
43     String rechtsb = blanks.substring(halb);
44     return linksb + s + rechtsb;
45 }
```

## Übung Ausnahmen 1:

**Ausnahmen** werden im **Kapitel 15** des Buches behandelt.

1. Führen Sie das Programm `Ausnahmen20` (siehe nächste Seite) mit Papier und Bleistift aus. Nehmen Sie dabei an, dass der Benutzer die Zahl 6 eingibt (für den Lesebefehl in Zeile 30 bzw. in Zeile 17). Geben Sie alle Zeilen an, die in diesem Fall zur Standardausgabe (zum Bildschirm) ausgegeben werden.
2. Ebenso wie 1., aber mit der Zahl 7 als Eingabe.
3. Ebenso wie 1., aber mit der Eingabe `abc`.
4. Führen Sie die folgende Befehlsfolge mit Papier und Bleistift aus und geben Sie an, was zur Standardausgabe (zum Bildschirm) ausgegeben wird:

```
1  for (int i1=3; i1<7; i1++) {
2      for (int i2=2; i2<=4; i2++) {
3          if ((i1 % 2) == (i2 % 2)) {
4              p("(" + i1 + ", " + i2 + ") ");
5          }
6      }
7  }
8  pln();
```

5. Betrachten Sie die folgende Vereinbarung einer Klasse:

```
9 class Carl {
10     static int    stefan    = 17;
11     static float  stanislaus = 1.5;
12     int    inga    = 27;
13     float  irene   = 3.8;
14 }
```

- 5.1. Welche Elemente der Klasse `Carl` gehören zum Modul `Carl`?
- 5.2. Welche Elemente der Klasse `Carl` gehören zum Bauplan (zum Typ) `Carl`?
- 5.3. Zeichnen Sie zwei Objekte der Klasse `Carl` als Bojen. Diese Objekte sollen `otto` bzw. `emil` heissen. Zeichnen Sie in beiden Objekten auch das `this`-Element ein.
6. Ergänzen Sie das folgenden Methoden-Skelett um einen geeigneten Methoden-Rumpf:

```
15 public String nurUngerade(String s) {
16     // Liefert einen String, der nur die Zeichen von s enthaelt, die einen
17     // ungeraden Index haben. Beispiel:
18     // nurUngerade("ABCDEFGH") ist gleich "BDF"
19     ...
20 }
```

**Hinweis:** Eine Ausnahme des Typs `NumberFormatException` enthält eine Meldung der folgenden Form: "For input string xyz" (wobei "xyz" der String ist, der nicht umgewandelt werden konnte).

## Das Programm Ausnahmen20:

```

1 // Vereinbarung einer geprüften Ausnahmeklasse:
2 class ZahlIstUngerade extends Exception {
3     public ZahlIstUngerade(String s, int n) {super(s); zahl = n;}
4     public int getZahl() {return zahl;}
5     private int zahl = 0;
6 } // class ZahlIstUngerade
7 // -----
8 public class Ausnahmen03 { // Die Hauptklasse dieses Programms
9     // -----
10    static public int liesEineGeradeZahl() throws
11        java.io.IOException, // Wenn die Tastatur Feuer faengt
12        ZahlIstUngerade // Wenn die Eingabe eine ungerade Zahl ist
13        // Liest eine gerade Ganzzahl von der Standardeingabe ein und
14        // liefert sie als Ergebnis.
15    {
16
17        String einGabeS = EM.liesString();
18        // Die Methode Integer.decode wirft evtl. eine NumberFormatException:
19        int einGabeI = Integer.decode(einGabeS);
20        if (einGabeI % 2 != 0) throw // <--- throw
21            new ZahlIstUngerade("Verflixst!!!", einGabeI);
22        return einGabeI;
23    } // liesEineGeradeZahl
24    // -----
25    static public void main(String[] sonja) {
26        p("Ausnahmen20: Eine gerade Zahl? ");
27        int zahl = 0;
28
29        try { // <--- try
30            zahl = liesEineGeradeZahl();
31            pln(zahl + " ist eine sehr gute Eingabe!");
32        }
33        catch (ZahlIstUngerade Ziu) { // <--- catch
34            pln("Ihre Eingabe " + Ziu.getZahl() + " ist ungerade!");
35            pln(Ziu.getMessage());
36        }
37        catch (java.io.IOException IOEx) { // <--- catch
38            pln("Eine Ausnahme des Typs java.io.IOException trat auf!");
39            pln(IOEx.getMessage());
40        }
41        catch (java.lang.NumberFormatException NFE) { // <--- catch
42            pln("NumberFormatException, Meldung: " + NFE.getMessage());
43            pln("Diese Ausnahme wird propagiert!");
44            throw NFE; // Die Ausnahme NFE wird propagiert // <--- throw
45        }
46        finally { // <--- finally
47            pln("Diese Meldung erscheint auf jeden Fall!");
48        }
49
50        // Die folgende Meldung wird nicht ausgegeben wenn das Programm mit
51        // einer NumberFormatException abgebrochen wird:
52        pln("Ausnahmen20: Das war's erstmal!");
53    } // main
54 } // class Ausnahmen20

```



## Lösung Ausnahmen 1:

### 1. Ein Dialog mit dem Programm Ausnahmen20:

```
1 Ausnahmen20: Eine gerade Zahl? 6
2 6 ist eine sehr gute Eingabe!
3 Diese Meldung erscheint auf jeden Fall!
4 Ausnahmen20: Das war's erstmal!
```

### 2. Noch ein Dialog mit dem Programm Ausnahmen20:

```
5 Ausnahmen20: Eine gerade Zahl? 7
6 Ihre Eingabe 7 ist ungerade!
7 Verflixt!!!
8 Diese Meldung erscheint auf jeden Fall!
9 Ausnahmen20: Das war's erstmal!
```

### 3. Noch ein Dialog mit dem Programm Ausnahmen20:

```
10 Ausnahmen20: Eine gerade Zahl? abc
11 NumberFormatException, Meldung: For input string: "abc"
12 Diese Ausnahme wird propagiert!
13 Diese Meldung erscheint auf jeden Fall!
```

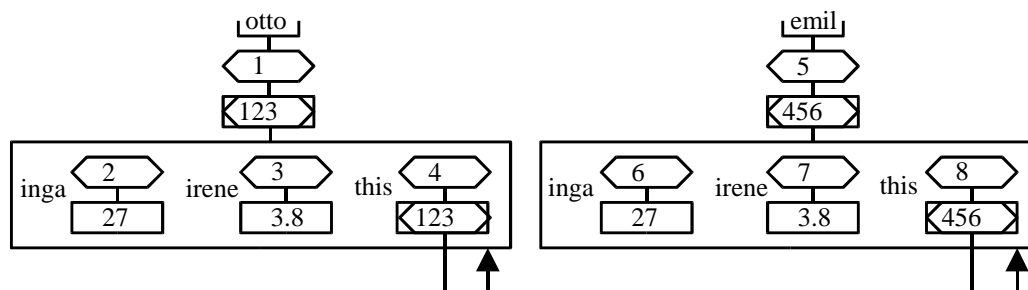
### 4. Die Ausgabe der doppelten Schleife:

```
14 (3, 3) (4, 2) (4, 4) (5, 3) (6, 2) (6, 4)
```

#### 5.1. Die Elemente stefan und stanislaus gehören zum Modul Carl.

#### 5.2. Die Elemente inga und irene gehören zum Bauplan (oder: zum Typ) Carl.

#### 5.3. Die Carl-Objekte namens otto und emil als Bojen:



### 6. Die Methode nurUngerade:

```
1 static public String nurUngerade(String s) {
2     // Liefert einen String, der nur aus den Zeichen von s besteht,
3     // die einen ungeraden Index haben.
4
5     StringBuilder sb = new StringBuilder();
6     for (int i=1; i < s.length(); i += 2) {
7         sb.append(s.charAt(i));
8     }
9     return sb.toString();
10 }
```

**Übung Ausnahmen 2:**

**Ausnahmen** werden im Kapitel 15 des Buches behandelt.

Betrachten Sie das folgende Java-Programm:

```

1 class Ausnahmen07 {
2     // -----
3     static public void main(String[] sonja) {
4
5         // Ein try-Block ohne catch-Block, aber mit finally-Block;
6         try {
7             AM.pln("Ausnahmen07: Jetzt geht es los!");
8         } finally {
9             AM.pln("Ausnahmen07: Kein catch-, aber ein finally-Block!");
10        } // try/finally
11
12        // Die methode01 wird wiederholt aufgerufen:
13        while (true) {
14            try {
15                AM.pln("A In der main-Methode wird methode01 aufgerufen!");
16                methode01();
17            } catch(Throwable t) {
18                AM.pln("D " + t);
19            }
20        } // while
21    } // main
22    // -----
23    static void methode01() throws Exception {
24        // Wird auf verschiedene Weise beendet oder aufgrund einer Ausnahme
25        // abgebrochen:
26        int n = 0;
27        while (true) {
28            try {
29                AM.p("----- methode01(): Eine Ganzzahl (1 bis 4)? ");
30                n = EM.liesInt();
31                switch (n) {
32                    case 1: throw new Exception("1 ist zu klein!");
33                    case 2: throw new Exception("2 ist mickrig!");
34                    case 3: throw new Exception("3 reicht beinahe!");
35                    case 4: throw new Exception("4 beendet alles!");
36                    default: throw new Exception(n + " ist falsch!");
37                } // switch
38
39            } catch(Throwable t) {
40                AM.pln("B " + t);
41
42                Exception e = new Exception(
43                    "In einem catch-Block wurde eine Ausnahme geworfen!");
44                );
45                if (n == 1) break; // Die while-Schleife beenden
46                if (n == 2) return; // Die methode01 beenden
47                if (n == 3) throw e; // Eine Ausnahme auslösen
48                if (n == 4) {
49                    AM.pln("Der finally-Block wurde *nicht* ausgeführt!");
50                    System.exit(1); // Das ganze Programm Ausnahmen01 beenden
51                } // if
52            } finally {
53                AM.pln("C Der finally-Block wird *fast* immer ausgeführt!");
54            } // try/catch/finally
55        } // while
56    } // methode01
57    // -----
58 } // class Ausnahmen07

```

Was gibt dieses Programm zur Standardausgabe (zum Bildschirm) aus, wenn der Benutzer (für den Le-sebefehl in Zeile 30) der Reihe nach folgende Zahlen eingibt: 7, 1, 2, 3, 4?

**Lösung Ausnahmen 2:**

```
1 Ein Dialog mit dem Programms Ausnahmen07:
2
3 Ausnahmen07: Jetzt geht es los!
4 Ausnahmen07: Kein catch-, aber ein finally-Block!
5 A In der main-Methode wird methode01 aufgerufen!
6 ----- methode01(): Eine Ganzzahl (1 bis 4)? 7
7 B java.lang.Exception: 7 ist falsch!
8 C Der finally-Block wird *fast* immer ausgeführt!
9 ----- methode01(): Eine Ganzzahl (1 bis 4)? 1
10 B java.lang.Exception: 1 ist zu klein!
11 C Der finally-Block wird *fast* immer ausgeführt!
12 A In der main-Methode wird methode01 aufgerufen!
13 ----- methode01(): Eine Ganzzahl (1 bis 4)? 2
14 B java.lang.Exception: 2 ist mickrig!
15 C Der finally-Block wird *fast* immer ausgeführt!
16 A In der main-Methode wird methode01 aufgerufen!
17 ----- methode01(): Eine Ganzzahl (1 bis 4)? 3
18 B java.lang.Exception: 3 reicht beinahe!
19 C Der finally-Block wird *fast* immer ausgeführt!
20 D java.lang.Exception: In einem catch-Block wurde eine Ausnahme geworfen!
21 A In der main-Methode wird methode01 aufgerufen!
22 ----- methode01(): Eine Ganzzahl (1 bis 4)? 4
23 B java.lang.Exception: 4 beendet alles!
24 Der finally-Block wurde *nicht* ausgeführt!
```

## Übung Methoden 2:

### 1. Ergänzen Sie das folgende "Skelett" zu einer Methode:

```
1  static public char[] a_nach_b(char[] r) {
2      // Liefert eine Kopie von r, in der alle Vorkommen von 'a' durch
3      // 'b' ersetzt wurden.
4      ...
5  } // a_nach_b
```

### 2. Ergänzen Sie das folgende "Skelett" zu einer Methode:

```
6  static public char[] a_nach_bb(char[] r) {
7      // Liefert eine Kopie von r, in der alle Vorkommen von 'a' durch
8      // zwei 'b' ersetzt wurden.
9      ...
10 } // a_nach_bb
```

**Hinweis:** Falls das Zeichen 'a' in der Reihung *r* (ein oder mehrmals) vorkommt, ist das Ergebnis der Methode eine Reihung, die *länger* ist als der Parameter *r*. Achten Sie darauf, dass diese Ergebnis-Reihung "genau die richtige Länge" hat, und nicht etwa "ein bisschen zu lang" ist.

### 3. Ergänzen Sie das folgende "Skelett" zu einer Methode:

```
11 static public char[] aa_nach_bbb(char[] r) {
12     // Liefert eine Kopie von r, in der alle Vorkommen von zwei unmittel-
13     // bar nacheinander liegenden 'a' durch drei 'b' ersetzt wurden.
14     ...
15 } // aa_nach_bbb
```

Der Hinweis zu der vorigen Aufgabe gilt entsprechend auch für diese Aufgabe. Ausserdem soll gelten: Eine Reihung {'a', 'a', 'a'} soll in eine Reihung der Länge 4 {'b', 'b', 'b', 'a'} übersetzt werden (und nicht in eine Reihung der Länge 6 {'b', 'b', 'b', 'b', 'b', 'b'}).

4. Die Datei `UebLoes.java` kann dazu benutzt werden, die Methoden `a_nach_b`, `a_nach_bb` und `aa_nach_bbb` (und einige weitere Methoden) zu testen. Eine kleine Bedienungsanleitung findet man am Anfang der Datei `UebLoes.java`.

### 5. Schreiben Sie drei Funktionen (ohne die Methode `replace` der Klasse `String` zu verwenden, zur Übung!):

```
16 static public String c_nach_d(String s) {
17     // Liefert eine Kopie von s, in der alle Vorkommen von 'c' durch
18     // 'd' ersetzt wurden.
19     ...
20 } // c_nach_d
21
22 static public String c_nach_dd(String s) {
23     // Liefert eine Kopie von s, in der alle Vorkommen von 'c' durch
24     // zwei 'd' ersetzt wurden.
25     ...
26 } // c_nach_dd
27
28 static public String cc_nach_ddd(String s) {
29     // Liefert eine Kopie von s, in der alle Vorkommen von zwei unmittel-
30     // bar nacheinander liegenden 'c' durch drei 'd' ersetzt wurden.
31     ...
32 } // cc_nach_ddd
```

Entwickeln Sie die Lösungen mit Papier und Bleistift (wie in der Klausur). Anschliessend können Sie auch diese Lösungen mit dem Programm `UebLoes` testen.

### 6. Programmieren Sie auch die anderen Methoden, die in der Datei `UebLoes` spezifiziert sind.

**Lösung Methoden 2:****Lösungen der Aufgaben 1., 2. und 3. :**

```

1  // -----
2  static public char[] a_nach_b(char[] r) {
3      // Liefert eine Kopie von r, in der alle Vorkommen von 'a' durch
4      // 'b' ersetzt wurden.
5      char[] q = new char[r.length]; // Eine qopie von r
6      for (int i=0; i<q.length; i++) {
7          if (r[i] == 'a') {
8              q[i] = 'b';
9          } else {
10             q[i] = r[i];
11         }
12     }
13     return q;
14 } // a_nach_b
15 // -----
16 static public char[] a_nach_bb(char[] r) {
17     // Liefert eine Kopie von r, in der jedes Vorkommen von 'a' durch
18     // je zwei 'b' ersetzt wurden.
19
20     // Zaehlen, wie viele 'a' in r vorkommen:
21     int anza = 0;
22     for (int i=0; i<r.length; i++) {
23         if (r[i]=='a') anza++;
24     }
25     char[] q = new char[r.length + anza]; // Ergebnis dieser Methode
26     int iq = 0; // erster freier Index von q
27     for (char c: r) {
28         if (c == 'a') {
29             q[iq++] = 'b';
30             q[iq++] = 'b';
31         } else {
32             q[iq++] = c;
33         }
34     } // for
35
36     return q;
37 } // a_nach_bb
38 // -----
39 static public char[] aa_nach_bbb(char[] r) {
40     // Liefert eine Kopie von r, in der jedes Vorkommen von zwei unmittel-
41     // bar nacheinander liegenden 'a' durch drei 'b' ersetzt wurde.
42     // Nur nicht-ueberlappende Vorkommen von "Doppel-a" werden ersetzt.
43
44     // Zaehlen, wie viele Doppel-a in r vorkommen:
45     int anzaa = 0;
46     int ir = 0; // naechster Index von r
47     while (ir < r.length-1) {
48         if (r[ir] == 'a' && r[ir+1] == 'a') {
49             anzaa++; // Ein Doppel-a zaehlen.
50             ir += 2; // Weil zwei Zeichen "erledigt" sind.
51         } else {
52             ir += 1; // Ein Zeichen erledigt
53         } // if
54     }
55
56     // Die Ergebnis-Reihung q vereinbaren und r nach q kopieren (dabei
57     // jedes Doppel-a durch drei 'b' ersetzen:
58     char[] q = new char[r.length + anzaa]; // Ergebnis dieser Methode
59     int iq = 0; // naechster freier Index von q
60     ir = 0; // naechster Index von r
61     while (ir < r.length-1) {
62         if (r[ir] == 'a' && r[ir+1] == 'a') {
63             q[iq++] = 'b';
64             q[iq++] = 'b';
65             q[iq++] = 'b';
66             ir += 2; // Weil zwei Zeichen "erledigt" sind.
67         } else {

```

```

68         q[iq++] = r[ir];
69         ir     += 1;    // Ein Zeichen erledigt
70     } // if
71 } // for
72 // Das letzte Zeichen von r wurde evtl. noch nicht nach q kopiert:
73 if (ir == r.length-1) q[iq++] = r[ir];
74 return q;
75 } // aa_nach_bbb
76 // -----

```

**Zur Aufgabe 4** dieser Übung gibt es keine Musterlösung.

**Lösungen zur Aufgabe 5 :**

```

77 // -----
78 static public String c_nach_d(String s) {
79     // Liefert eine Kopie von s, in der alle Vorkommen von 'c' durch
80     // 'd' ersetzt wurden.
81
82     StringBuilder erg = new StringBuilder(s);
83     for (int i=0, laenge=erg.length(); i<laenge; i++) {
84         if (erg.charAt(i) == 'c') erg.setCharAt(i, 'd');
85     }
86     return erg.toString();
87 } // c_nach_d
88 // -----
89 static public String c_nach_dd(String s) {
90     // Liefert eine Kopie von s, in der jedes Vorkommen von 'c' durch
91     // je zwei 'd' ersetzt wurde.
92
93     StringBuilder erg = new StringBuilder(s);
94     for (int i=erg.length()-1; i >= 0; i--) {
95         if (erg.charAt(i) == 'c') erg.replace(i, i+1, "dd");
96     }
97     return erg.toString();
98 } // c_nach_dd
99 // -----
100 static public String cc_nach_ddd(String s) {
101     // Liefert eine Kopie von s, in der jedes Vorkommen von zwei unmit-
102     // telbar nacheinander liegenden 'c' durch drei 'd' ersetzt wurden.
103     // Nur nicht-ueberlappende Vorkommen von "Doppel-ces" werden ersetzt.
104
105     StringBuilder erg = new StringBuilder(s);
106     for (int i=0, laenge=erg.length(); i<=laenge - 2; i++) {
107         if (erg.substring(i, i+2).equals("cc")) {
108             erg.replace(i, i+2, "ddd");
109             i += 2;
110         }
111     }
112     return erg.toString();
113 } // cc_nach_ddd
114 // -----
115

```

**6. Für die anderen in UebLoes spezifizierten Methoden müssen Sie selbst Musterlösungen entwickeln.**

## Übung Punkt, Quadrat, Rechteck, Kreis

Betrachten Sie im **Abschnitt 12.3 des Buches** die Klassen `E01Punkt`, `E01Quadrat`, `E01Rechteck` und `E01Kreis` und füllen Sie dann die folgenden Tabellen aus. Geben Sie jeweils die **Anzahl** der betreffenden Elemente und die **Namen** der Elemente an. Die erste Tabelle (für die Klasse `E01Punkt`) ist als Beispiel bereits ausgefüllt. Um die Übung zu vereinfachen, wurden dabei die Elemente, die die Klasse `Punkt` von der Klasse `Object` erbt, vernachlässigt und nicht erwähnt.

### Klasse Punkt:

Objektattribute, geerbt	-- (die von <code>Object</code> geerbten Elemente werden hier vernachlässigt)
Objektattribute, neu	2, <code>x</code> , <code>y</code>
Objektmethoden, geerbt	-- (die von <code>Object</code> geerbten Elemente werden hier vernachlässigt)
Objektmethoden, neu	4, <code>urAbstand</code> , <code>urSpiegeln</code> , <code>text</code> , <code>toString</code>

### Klasse Rechteck:

Objektattribute, geerbt	
Objektattribute, neu	
Objektmethoden, geerbt	
Objektmethoden, neu	

### Klasse Quadrat:

Objektattribute, geerbt	
Objektattribute, neu	
Objektmethoden, geerbt	
Objektmethoden, neu	

### Klasse Kreis:

Objektattribute, geerbt	
Objektattribute, neu	
Objektmethoden, geerbt	
Objektmethoden, neu	

**Lösung Punkt, Quadrat, Rechteck, Kreis**

Hier die ausgefüllten Tabellen:

**Klasse Punkt:**

Objektattribute, geerbt	-- (die von Object geerbten Elemente werden hier vernachlässigt)
Objektattribute, neu	2, x, y
Objektmethoden, geerbt	-- (die von Object geerbten Elemente werden hier vernachlässigt)
Objektmethoden, neu	4, urAbstand, urSpiegeln, text, toString

**Klasse Rechteck:**

Objektattribute, geerbt	2, x, y
Objektattribute, neu	2, deltax, deltay
Objektmethoden, geerbt	4, urAbstand, urSpiegeln, text, toString
Objektmethoden, neu	3, toString, getUmfang, getFlaeche

**Klasse Quadrat:**

Objektattribute, geerbt	4, x, y, deltax, deltay
Objektattribute, neu	0
Objektmethoden, geerbt	4, urAbstand, urSpiegeln, text, toString (5.toString)
Objektmethoden, neu	3, toString, getUmfang, getFlaeche

**Klasse Kreis:**

Objektattribute, geerbt	2, x, y
Objektattribute, neu	1, radius
Objektmethoden, geerbt	4, urAbstand, urSpiegeln, text, toString
Objektmethoden, neu	3, toString, getUmfang, getFlaeche



## Übung Oberklassen/Unterklassen

*Oberklassen* und *Unterklassen* etc. werden im **Kapitel 12** des Buches behandelt.

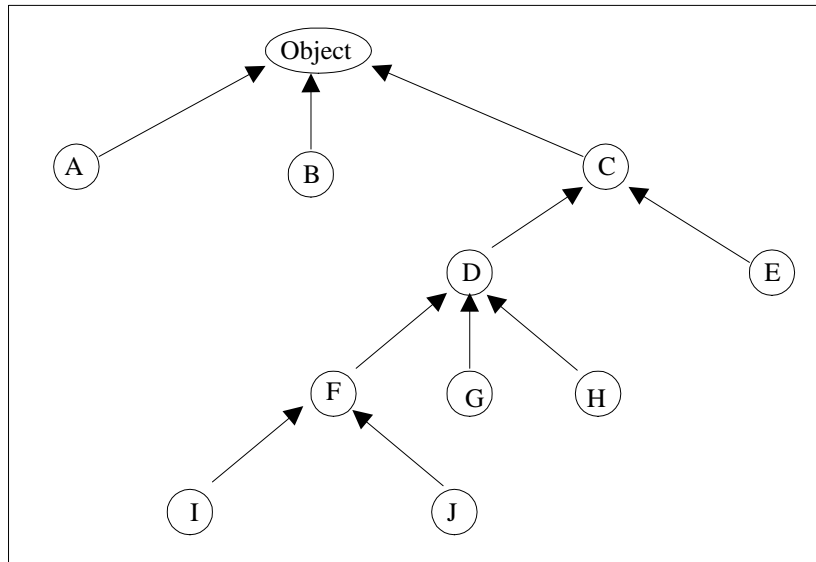
Betrachten Sie die folgende Klassenhierarchie (mit den Klassen Object, A, B, C, ...). Ein Pfeil von K2 nach K1 bedeutet:

die Klasse K2 *erbt* von der Klasse K1, oder:

die Klasse K2 *ist eine Erweiterung* der Klasse K1, oder:

die Klasse K2 *ist eine direkte Unterklasse* von K1, oder:

die Klasse K1 *ist die direkte Oberklasse* von K2.



Geben Sie die folgenden Klassen an:

1. **Die** direkte Oberklasse von F?
2. **Eine** indirekte Oberklasse von F?
3. Alle Oberklassen von F?
4. **Eine** direkte Unterklasse von C?
5. **Eine** indirekte Unterklasse von C?
6. Alle Unterklassen von C?

## Lösung Oberklassen/Unterklassen

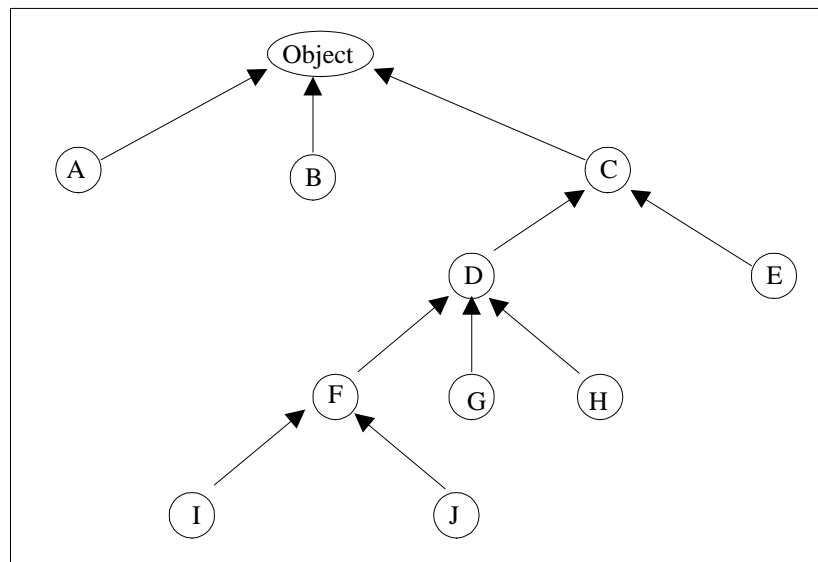
Betrachten Sie die folgende Klassenhierarchie (mit den Klassen Object, A, B, C, ...). Ein Pfeil von K2 nach K1 bedeutet:

die Klasse K2 *erbt* von der Klasse K1, oder:

die Klasse K2 *ist eine Erweiterung* der Klasse K1, oder:

die Klasse K2 *ist eine direkte Unterklasse* von K1, oder:

die Klasse K1 *ist die direkte Oberklasse* von K2.



Geben Sie die folgenden Klassen an:

- |   |                               |
|---|-------------------------------|
| 1. <b>Die</b> direkte Oberklasse von F?     | D                             |
| 2. <b>Eine</b> indirekte Oberklasse von F?  | C oder Object                 |
| 3. Alle Oberklassen von F?                  | D, C, Object                  |
| 4. <b>Eine</b> direkte Unterklasse von C?   | D oder E                      |
| 5. <b>Eine</b> indirekte Unterklasse von C? | F oder G oder H oder I oder J |
| 6. Alle Unterklassen von C?                 | D, E, F, G, H, I, J           |

## Übung Bitfummeln

*Operatoren* werden im Abschnitt 6.2 des Buches behandelt (aber die Operatoren zum Bitfummeln werden dort nur sehr kurz dargestellt (siehe Seite 145 bis 147).

1. Geben Sie für jed Ziffer des 16-er-Systems die entsprechende vierstellige Zahl im 2-er-System an:

16-er	2-er	16-er	2-er	16-er	2-er	16-er	2-er
0	0000	4		8		C	
1	0001	5		9		D	
2	0010	6		A		E	
3	0011	7		B		F	

2. Die folgende Tabelle enthält `int`-Literele im 16-er-System und im 10-er-Syste (und zwei Attributnamen). Geben Sie jeweils das fehlende Literal an:

16-er	10-er	16-er	10-er
0x01			11
0x0A			15
0xA0			16
0xAA			-16
0xFF			<code>Integer.MIN_VALUE</code>
0xFFFFFFFF			<code>Integer.MAX_VALUE</code>
0xFFFFFFFFE			

Geben Sie die Werte der folgenden `int`-Ausdrücke im 16-er und im 10-er-System an:

int-Ausdruck	16-er	10-er	int-Ausdruck	16-er	10-er
1 << 1			-1 >>> 1		
1 << 2			-1 >>> 2		
1 << 3			-1 >>> 3		
1 << 4			-1 >>> 4		
1 << 5			-1 >>> 5		
1 << 6			-1 >>> 6		
1 << 7			-1 >>> 7		
1 << 8			-1 >>> 8		
-1 >> 1			-1 / 2		
-1 >> 2					
-1 >> 17					

**Lösung Bitfummeln**

1. Geben Sie für jed Ziffer des 16-er-Systems die entsprechende vierstellige Zahl im 2-er-System an:

16-er	2-er	16-er	2-er	16-er	2-er	16-er	2-er
0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

2. Die folgende Tabelle enthält `int`-Literele im 16-er-System und im 10-er-Syste (und zwei Attributnamen). Geben Sie jeweils das fehlende Literal an:

16-er	10-er	16-er	10-er
0x00000001	1	0x0000000B	11
0x0000000A	10	0x0000000F	15
0x000000A0	160	0x00000010	16
0x000000AA	170	0xFFFFFFFF0	-16
0x000000FF	255	0x80000000	<code>Integer.MIN_VALUE</code>
0xFFFFFFFF	-1	0x7FFFFFFF	<code>Integer.MAX_VALUE</code>
0xFFFFFFFFE	-2		

Geben Sie die Werte der folgenden `int`-Ausdrücke im 16-er und im 10-er-System an:

int-Ausdruck	16-er	10-er	int-Ausdruck	16-er	10-er
1 << 1	0x00000002	2	-1 >>> 1	0x7FFFFFFF	2147483647
1 << 2	0x00000004	4	-1 >>> 2	0x3FFFFFFF	1073741823
1 << 3	0x00000008	8	-1 >>> 3	0x1FFFFFFF	536870911
1 << 4	0x00000010	16	-1 >>> 4	0x0FFFFFFF	268435455
1 << 5	0x00000020	32	-1 >>> 5	0x07FFFFFF	134217727
1 << 6	0x00000040	64	-1 >>> 6	0x03FFFFFF	67108863
1 << 7	0x00000080	128	-1 >>> 7	0x01FFFFFF	33554431
1 << 8	0x00000100	256	-1 >>> 8	0x00FFFFFF	16777215
-1 >> 1	0xFFFFFFFF	-1	-1 / 2	0x00000000	0
-1 >> 2	0xFFFFFFFF	-1			
-1 >> 17	0xFFFFFFFF	-1			

## Übung Einfach boolean

Im Buch wird (leider) nicht erläutert, wie man Namen von `boolean`-Variablen wählen sollte und wie man Bedingungen in `if`-Anweisungen und `boolean`-Ausdrücke in `return`-Anweisungen vereinfachen kann. Hier folgt wenigstens eine Übung über dieses Gebiet.

Variablen des Typs `boolean` und Funktionen mit dem Ergebnistyp `boolean` sollten immer Namen haben, aus denen eindeutig hervorgeht, was der Wert (bzw. das Ergebnis) `true` (bzw. `false`) bedeutet. Schlechte Namen: `test`, `pruefen`, `vergleichen`. Gute Namen: `istDreieck`, `hat4Ecken`, `passtRein`.

1. Schlagen Sie bessere Namen vor für die folgenden Variablen und Funktionen:

```
1 boolean b1 = n > 0;
2 boolean b2 = n <= 0;
3 boolean b3 = n%2 == 0;
4 boolean b4 = n%2 != 0;
5
6 boolean f1(char c) {
7     return ('A' <= c && c <= 'Z') || ('a' <= c && c <= 'z');
8 }
9
10 boolean f2(int n) {
11     n = Math.abs(n);
12     if (n <= 1) return false;
13     if (n == 2) return true;
14     if (n%2 == 0) return false;
15
16     final int MAX = (int) Math.sqrt(n);
17     for (int i=3; i<MAX; i+=2) {
18         if (n%i == 0) return false;
19     }
20     return true;
21 }
```

2. Vereinfachen Sie die folgenden `if`-Anweisungen:

```
1 if ((a<b) == true) ...
2 if ((a<=b) == false) ...
3 if ((a<b) != true) ...
4 if ((a<=b) != false) ...
5 if ((a<b) && (c<d) && (e<f) == true) ...
```

3. Vereinfachen Sie die folgenden `if`-Anweisungen mit `return`-Anweisungen darin:

```
6 if (a<b) {
7     return true;
8 } else {
9     return false;
10 }
11
12 if (a<b) {
13     return false;
14 } else {
15     return true;
16 }
17
18 if ((a<b) && (c<d) && (e<f) == true) {
19     return true;
20 } else {
21     return false;
22 }
```

## Lösung Einfach boolean

1. Schlagen Sie bessere Namen vor für die folgenden Variablen und Funktionen vor

für b1: nIstPositiv **oder** istPositiv **oder** positiv

für b2: nIstNichtPositiv **oder** kleinerGleichNull **oder** nichtPositiv

für b3: nIstGerade **oder** istGerade

für b4: nIstUngerade **oder** istUngerade

für f1: istBuchstabe

für f2: istPrim

2. Vereinfachen Sie die folgenden if-Anweisungen:

### Vorgegebene Form:

```
23 if ((a<b) == true) ...
24 if ((a<=b) == false) ...
25 if ((a<b) != true) ...
26 if ((a<=b) != false) ...
27 if ((a<b) && (c<d) && (e<f) == true) ...
```

### Vereinfachte Form:

```
if (a<b) ...
if (a>b) ...
if (a>=b) ...
if (a<=b) ...
if ((a<b) && (c<d) && (e<f)) ...
```

3. Vereinfachen Sie die folgenden if-Anweisungen mit return-Anweisungen darin:

### Vorgegebene Form:

```
28 if (a<b) {
29     return true;
30 } else {
31     return false;
32 }
33
34 if (a<b) {
35     return false;
36 } else {
37     return true;
38 }
39
40 if ((a<b) && (c<d) && (e<f) == true) {
41     return true;
42 } else {
43     return false;
44 }
```

### Vereinfachte Form:

```
return a<b;

return !(a<b);
oder:
return b<=a;

return a<b && c<d && e<f;
```

Java ist eine Sprache

Java lesen, schreiben und ausführen — Eine präzise  
und verständliche Einführung

Grude, U.

2005, XII, 604 S. 612 Abb., Softcover

ISBN: 978-3-528-05914-9