

**Inhaltsverzeichnis**

Übung if:.....	2
Übung ÜberAus 1:.....	4
Übung Schleifen (ausführen) 1:.....	5
Übung Schleifen (programmieren) 2:.....	6
Übung Schleifen (ausführen) 3:.....	7
Übung Methoden 1:.....	8
Übung Reihungen 1:.....	9
Übung Reihungen 2:.....	10
Übung Reihungen 3:.....	11
Übung Bojen (ausführliche und vereinfachte) 1:.....	12
Übung Bojen (von Reihungen) 2:.....	13
Übung Klassen 1:.....	14
Übung Deutsch 1:.....	16
Übung Klassen 2:.....	17
Übung Klassen 3:.....	20
Übung Strings 1:.....	21
Übung Ausnahmen 1:.....	22
Übung Ausnahmen 2:.....	24
Übung Methoden 2:.....	25
Übung Punkt, Quadrat, Rechteck, Kreis.....	26
Übung Oberklassen/Unterklassen.....	27
Übung Bitfummeln.....	28
Übung Einfach boolean.....	29

In den Übungen wird manchmal auf das folgende Buch Bezug genommen:

[JASP]

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

**Übung if:**

Machen Sie sich mit den verschiedenen Varianten der `if`-Anweisung vertraut (siehe z. B. [JASP], Abschnitt 4.2.2). 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.
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-Anweisung 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
```

**Übung ÜberAus 1:**

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*, *Ausdrück* oder *Vereinbarung*) 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, Bleistift und Radiergummi) 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);
```

## Übung Schleifen (ausführen) 1:

Führen Sie die folgenden Befehlsfolgen mit Papier, Bleistift und Radiergummi aus. Geben Sie genau an, welche Variablen erzeugt wurden, welche Werte in jeder Variablen (nacheinander) standen und welche Zeichen und Zeilen zur *Standardausgabe* (d.h. zum Bildschirm) ausgegeben wurden (wie im Kapitel 3 des Buches "Java ist eine Sprache" beschrieben). Beachten Sie dabei den Unterschied zwischen `print` (der Bildschirmzeiger bleibt unmittelbar hinter den ausgegeben Zeichen stehen) und `println` (der Bildschirmzeiger rückt zum Anfang der nächsten Zeile vor). Ist eine Schleife wie `for (int i=1; ...) { ... }` fertig ausgeführt, so wird die Schleifenvariable `i` *zerstört*.

### 1. Die anna-Schleife:

```
19 int anna = 7;
20 while (anna > 0) {
21     p(anna + " ");
22     anna /= 2;
23 }
24 println();
```

### 2. Die berta-Schleife:

```
25 int berta = 7;
26 do {
27     berta /= 2;
28     System.out .print(" -> " + berta);
29 } while (berta > 0);
30 println();
```

### 3. Die celia-Schleife:

```
31 for (int celia = -3; celia < 5; celia += 2) {
32     p(2 * celia + 4 + " ");
33 }
34 println();
```

### 4. Die dora-Schleife:

```
35 for (int dora = 3; 2*dora > -3; dora--) {
36     p(dora + " ");
37 }
38 println();
```

### 5. Die MAX1-Schleife:

```
39 final int MAX1 = 3;
40 for (int i1 = 1; i1 <= MAX1; i1++) {
41     for (int i2 = 1; i2 <= 2*MAX1; i2++) {
42         p("**");
43     }
44     println();
45 }
```

### 6. Die MAX2-Schleife:

```
46 final int MAX2 = 5;
47 for (int i1 = 1; i1 <= MAX2; i1++) {
48     for (int i2 = 1; i2 <= i1; i2++) {
49         p(++);
50     }
51     println();
52 }
```

**Übung Schleifen (programmieren) 2:**

1. Programmieren Sie eine Schleife, die die Ganzzahlen von 1 bis 10 (alle auf einer Zeile, durch je ein Blank voneinander getrennt) zur Standardausgabe (d.h. 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?

3. Programmieren Sie für jede der folgenden Zahlenfolgen eine Schleife, die die Zahlenfolge zur Standardausgabe ausgibt:

```
3.1. -5 -2 1 4 7 10 13 16 19
```

```
3.2. 1 2 4 8 16 32 64 128 256 512 1024 2048 4096
```

```
3.3. 3 4 6 10 18 34 66 130 258 514 1026 2050 4098
```

```
3.4. 1 2 4 7 11 16 22 29 37 46 56 67 79 92
```

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

```
int norbert = EM.liesInt();
```

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

5. Befehlen Sie dem Ausführer, den kleinsten Teiler von `norbert` (der grösser als 1 ist) auszugeben. Falls `norbert` eine Primzahl ist, soll als kleinster Teiler der Wert von `norbert` selbst ausgegeben werden.

6. 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.) finden Sie z. B. im Abschnitt 10.1 von [JASP] und im Beispielprogramm `String02.java`.

7. 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.

8. 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.).

9. 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.

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

11. 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.

**Übung Schleifen (ausführen) 3:**

Führen Sie die folgenden Schleifen mit Papier, Bleistift und Radiergummi aus. Geben Sie für jede Schleife an, welche Variablen erzeugt werden, welche Werte (nacheinander) in jeder Variablen standen und was zum Bildschirm ausgegeben wurde (wie im Kapitel 3 des Buches "Java ist eine Sprache" beschrieben).

```
1      final int z = 3;
2
3      // Schleife 1:
4      for (int i=1; i<=z; i++) {
5          for (int j=z-i; j>=1;      j--) p("00");
6          for (int j=1;      j<=i+(i-1); j++) p("XX");
7          for (int j=z-i; j>=1;      j--) p("00");
8          pln();
9      }
10
11     // Schleife 2:
12     for (char c1='0'; c1<'2'; c1++) {
13         for (char c2='C'; c2>='A'; c2--) {
14             p(" " + c2 + c1 + " ");
15         }
16         pln();
17     }
18
19     // Schleife 3:
20     for (int i=1; i<=3; i++) {
21         switch (i) {
22             case 2 : p("A");
23             default : p("B");
24             case 1 : p("C"); break;
25         }
26     }
27     pln();
28
29     // Schleife 4:
30     int a = 1;
31     int b = 2;
32     int c = 3;
33     for (int i=1; i<=5; i++) {
34         pln(a + b + c);
35         a++;
36         b += a;
37         c += a + b;
38     }
39
40     // Schleife 5:
41     for (int i=1; i<=3; i++) {
42         for (int j=3-i; j>=1;      j--) p("A");
43         for (int j=1;      j<=i+(i-1); j++) p("B");
44         for (int j=3-i; j>=1;      j--) p("A");
45         pln();
46     }
```

## Übung Methoden 1:

Zur Erinnerung: Methoden mit dem pseudo-Rückgabety `void` werden auch als *Prozeduren* bezeichnet. Alle anderen Methoden ("Methoden mit einem richtigen, von `void` verschiedenen Rückgabety") 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 }
5
```

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 ein `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"  $a^2 = b^2 + c^2$ .

**Übung Reihungen 1:**

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

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

**Übung Reihungen 2:****Ergänzen Sie folgenden Methoden-Skelette zu richtigen Methoden:**

```
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 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 // -----
```

**Übung Reihungen 3:**

Betrachten Sie die folgenden fünf Vereinbarungen von zweistufigen Reihungsvariablen:

```
1 int[][] irrA = null; // Keine Reihung
2 int[][] irrB = new int[3][]; // 1-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*.

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

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.

**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 }
```

## Übung Klassen 1:

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) {pln(ob);}
66     static void pln()         {pln(); }
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.

**Übung Deutsch 1:**

Betrachten Sie 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.

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

```

**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 // Verfügung 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 Verfügung, 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
44

```

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 `EM` 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 und bei den Beispielprogrammen). 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 (im 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`:

```

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

```

```

73     pln("B int02 + 12345: " + add(int02, 12345));
74     pln("C int02 + int01: " + add(int02, int01));
75 } // main
76 // -----
77 } // class IntRech02

```

### Das Programms BigInteger01:

```

78 import java.math.BigInteger;
79
80 public class BigInteger01 {
81     // -----
82     static public void main(String[] emil) throws
83         java.io.IOException,
84         java.lang.ArithmeticException,
85         java.lang.NumberFormatException
86     {
87         pln("A BigInteger01: Jetzt geht es los!");
88
89         while (true) {
90             pln("B Zum Beenden bitte 0 eingeben!");
91             p ("C BigInteger BI1? ");
92             BigInteger bi1 = EM.liesBigInteger();
93             if (bi1.compareTo(BigInteger.ZERO) == 0) break;
94             p ("D BigInteger BI2? ");
95             BigInteger bi2 = EM.liesBigInteger();
96
97             pln("E BI1:          " + bi1);
98             pln("F BI2:          " + bi2);
99
100            pln("G BI1 + BI2:      " + bi1.add(bi2));
101            pln("H BI1 - BI2:      " + bi1.subtract(bi2));
102            pln("I BI1 * BI2:      " + bi1.multiply(bi2));
103            pln("J BI1 / BI2:      " + bi1.divide(bi2));
104
105            // Die Methode divideAndRemainder liefert eine Reihung mit zwei
106            // Komponenten vom Typ BigInteger: den Quotienten und den Rest.
107            BigInteger[] bir = bi1.divideAndRemainder(bi2);
108            pln("K BI1 d BI2:    " + bir[0]); // d wie divide
109            pln("L BI1 r BI2:    " + bir[1]); // r wie remainder
110
111            // Die Methode mod wirft eine Ausnahme ArithmeticException, wenn
112            // der zweite Operand (der Modulus) negativ ist. Man beachte den
113            // subtilen Unterschied zwischen den beiden Restfunktionen rem
114            // (d.h. divideAndRemainder) und mod!
115            pln("M BI1 mod BI2:  " + bi1.mod(bi2));
116        } // while
117        pln("N BigInteger01: Das war's erstmal!");
118    } // end main

```

### Übung Klassen 3:

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 [JASP].

```

1 class Ueb_Klassen3 {
2     static public main main(String[] sonja) {
3         E01Punkt p1 = new E01Quadrat(0.0, 0.0, 1.0);
4         ...
5         p1 = new E01Rechteck(0.5, 1.5, 2.5, 4.0);
6         ...
7         p1 = new E01Punkt(1.0, 0.0);
8         ...
9         p1 = new E01Kreis(0.0, 1.0, 2.0);
10        ...

```

2.1. Welchen *Typ* hat die Variable p1?

2.2. Welchen *Zieltyp* hat die Variable p1, wenn der Ausführer gerade damit fertig ist, die *Zeile 3* auszuführen?

2.3. Ebenso für *Zeile 5*.    2.4. Ebenso für *Zeile 7*.    2.5. Ebenso für *Zeile 9*.

2.6. In welche der Zeilen 4, 6, 8 und 10 dürfte man die folgende Vereinbarung schreiben:

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

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

2.7. In welche der Zeilen 4, 6, 8 und 10 dürfte man die folgende Vereinbarung schreiben:

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

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

2.8. In welche der Zeilen 4, 6, 8 und 10 dürfte man die folgende Vereinbarung schreiben:

```
double u = ((E01Rechteck) p1).getUmfang();
```

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

**Übung Strings 1:**

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);
5
```

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

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

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

```
12 static public String linksBuendig(String s, int len) {
13 // Falls s.length groesser oder gleich len ist, wird s unveraendert
14 // als Ergebnis geliefert. Sonst werden an der rechten Seite von s
15 // soviele Blanks angehaengt, dass ein String der Laenge len entsteht.
16 // Dieser wird als Ergebnis geliefert.
17 ...
18 }
19
20 static public String rechtsBuendig(String s, int len) {
21 // Falls s.length groesser oder gleich len ist, wird s unveraendert
22 // als Ergebnis geliefert. Sonst werden an der linken Seite von s
23 // soviele Blanks angehaengt, dass ein String der Laenge len entsteht.
24 // Dieser wird als Ergebnis geliefert.
25 ...
26 }
27
28 static public String zentriert(String s, int len) {
29 // Falls s.length groesser oder gleich len ist, wird s unveraendert
30 // als Ergebnis geliefert. Sonst werden links und rechts von s soviele
31 // Blanks angehaengt, dass ein String der Laenge len entsteht. Dieser
32 // wird als Ergebnis geliefert. Falls moeglich werden links und rechts von
33 // s gleich viel Blanks angehaengt und sonst rechts eins mehr als links.
34 ...
35 }
```

## Übung Ausnahmen 1:

1. Führen Sie das Programm `Ausnahmen03` (siehe nächste Seite und bei den Beispielprogrammen) 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 Ausnahmen03:

```

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("Verflixt!!!", einGabeI);
22        return einGabeI;
23    } // liesEineGeradeZahl
24    // -----
25    static public void main(String[] sonja) {
26        p("Ausnahmen03: 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("Ausnahmen03: Das war's erstmal!");
53    } // main

```

**Übung Ausnahmen 2:**

Betrachten Sie das folgende Java-Programm:

```

1 class Ausnahmen07 {
2     // -----
3     static public void main(String[] sonja) {
4
5         // Ein try-Block ohne catch-Blocke, 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?

## 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 {'b', 'b', 'b', 'a'} übersetzt werden (und nicht in eine Reihung {'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 benutzen, 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.

### Übung Punkt, Quadrat, Rechteck, Kreis

Betrachten Sie in [JASP] die Klassen `E01Punkt`, `E01Quadrat`, `E01Rechteck` und `E01Kreis` (im Abschnitt 12.3) 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	

## Übung Oberklassen/Unterklassen

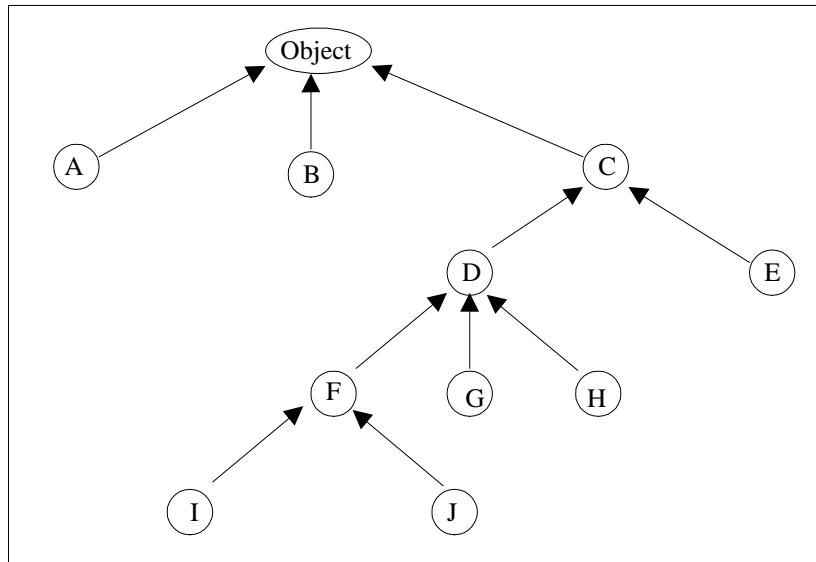
Betrachten Sie die folgende Klassenhierarchie (mit den Klassen Object, A, B, C, ...). Ein Pfeil von K2 nach K1 bedeutet:

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. *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?

## Übung 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		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-System (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
<code>1 &lt;&lt; 1</code>			<code>-1 &gt;&gt;&gt; 1</code>		
<code>1 &lt;&lt; 2</code>			<code>-1 &gt;&gt;&gt; 2</code>		
<code>1 &lt;&lt; 3</code>			<code>-1 &gt;&gt;&gt; 3</code>		
<code>1 &lt;&lt; 4</code>			<code>-1 &gt;&gt;&gt; 4</code>		
<code>1 &lt;&lt; 5</code>			<code>-1 &gt;&gt;&gt; 5</code>		
<code>1 &lt;&lt; 6</code>			<code>-1 &gt;&gt;&gt; 6</code>		
<code>1 &lt;&lt; 7</code>			<code>-1 &gt;&gt;&gt; 7</code>		
<code>1 &lt;&lt; 8</code>			<code>-1 &gt;&gt;&gt; 8</code>		
<code>-1 &gt;&gt; 1</code>			<code>-1 / 2</code>		
<code>-1 &gt;&gt; 2</code>					
<code>-1 &gt;&gt; 17</code>					

## Übung Einfach boolean

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`, `vergleich`. Gute Namen: `istDreiseit`, `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:

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

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

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