

Lehrveranstaltung MB1-PR1 (Programmieren 1) an der Beuth Hochschule für Technik Berlin

Datei 14WS\_PR1\_Uebungen

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

### **Inhaltsverzeichnis**

Übung 1: Zahlensysteme.....	2
Übung 2: Gleitpunktzahlen.....	3
Übung 3: if.....	6
Übung 4: Ausführen.....	8
Übung 5: Schleifen A (ausführen).....	9
Übung 6: Schleifen B (programmieren).....	10
Übung 7: Schleifen C (programmieren).....	11
Übung 8: Methoden A.....	14
Übung 9: Methoden B.....	16
Übung 10: Reihungen A.....	17
Übung 11: Reihungen B.....	19
Übung 12: Reihungen C (mehrstufige, als Bojen).....	20
Übung 13: Sammlungen.....	21
Übung 14: Bojen A.....	22
Übung 15: Bojen B (von Reihungen).....	23
Übung 16: Klassen A.....	24
Übung 17: Klassen B.....	26
Übung 18: Klassen C.....	27
Übung 19: Klassen D.....	30
Übung 20: Strings.....	31
Übung 21: Ausnahmen A.....	32
Übung 22: Ausnahmen B.....	34
Übung 23: E01Punkt etc.....	35
Übung 24: Punkt, Quadrat, Rechteck, Kreis.....	36
Übung 25: Oberklassen/Unterklassen.....	37
Übung 26: Bitfummeln.....	38
Übung 27: Einfach boolean.....	39
Übung 28: Histogramme.....	40

## Übung 1: Zahlensysteme

1. Füllen Sie die folgende Tabelle aus:

als 10-er-Zahl	als 2-er-Zahl	als 5-er-Zahl	als 9-er-Zahl	als 12-er-Zahl
$10_{10}$				
$25_{10}$				
$37_{10}$				

2. Tragen Sie in die folgende Tabelle die *Stellenwerte* der Stellen zwischen +2 und -2 für die Zahlensysteme mit den Basen 2, 3, 9 und 12 ein (natürlich als Dezimalzahlen). Die Stellenwerte für die *negativen Stellen* ("rechts vom Punkt") dürfen Sie als *Stammbrüche* (z.B.  $1/2$ ,  $1/27$ ,  $1/144$  etc.) notieren (statt als Dezimalbrüche wie  $0.5_{10}$  oder  $0.0370_{10}$  etc.).

Stelle System	+2	+1	0	-1	-2
2-er-System					
3-er-System					
9-er-System					
12-er-System					

3. Wenn man beim Dividieren oder Modulieren ("die mod-Operation anwenden") von Ganzzahlen ein Ganzzahl-Ergebnis herausbekommen will, hat man meistens eine Wahl zwischen zwei Ergebnissen. Tragen Sie beide Lösungen in die folgende Tabelle ein:

dend	dor	dend div dor		dend mod dor	
		1. Lösung	2. Lösung	1. Lösung	2. Lösung
+7	+3				
+7	-3				
-7	+3				
-7	-3				

4. Sei  $n$  eine nicht-negative Ganzzahl.  
Stellen Sie sich  $n$  als 10-er-Zahl dargestellt vor.

Welcher Ausdruck bezeichnet die Ziffer an der Stelle 0 von $n$ ?	
Welche Zuweisung an $n$ entfernt die Ziffer an der Stelle 0 von $n$ ?	

5. Stellen Sie sich  $n$  als  $b$ -er-Zahl dargestellt vor  
(wobei  $b$  irgendeine Basiszahl wie z.B. 2 oder 10 oder 16 etc. ist).

Welcher Ausdruck bezeichnet die Ziffer an der Stelle 0 von $n$ ?	
Welche Zuweisung an $n$ entfernt die Ziffer an der Stelle 0 von $n$ ?	

## Übung 2: Gleitpunktzahlen

Einem Windows-Rechner erlauben, Java-Applets von einer vertrauenswürdigen Seite auszuführen:

**Start-Knopf / Alle Programme / Java / Configure Java** (oder: **Java konfigurieren**)

Dadurch sollte das Java Control Panel geöffnet werden.

### Sicherheit / Siteliste bearbeiten

Unter **Verzeichnis** das betreffende Verzeichnis

(z.B. <http://public.beuth-hochschule.de/~grude/>)

eintragen (am besten kopieren) und auf **Hinzufügen** klicken.

Starten Sie mit einem Browser auf der Netzseite <http://public.beuth-hochschule.de/~grude/>

das `GleitBitsFloatApplet`. Machen Sie sich mit der Struktur der GRABO (engl. GUI) vertraut:

**Oben:** Die 32 Bits eines float-Wertes (1 Bit *Vorzeichen*, 8 Bits *Charakteristik*, 23 Bits *Mantisse*), weißes/schwarzes Quadrat: *Bitwert 0/1*, verändern durch Anklicken.

**Mitte:** 6 Datenfelder: **Vorzeichen** bis **Die Gleitpunktzahl im Ganzen** und 2x6 Knöpfe: **Alle Bits an** bis **Durch 0.1**.

**Unten:** Ein Datenfeld **Alle Ziffern der Gleitpunktzahl**.

**Graue** Datenfelder: Nur für Anzeige.

**Weiß**e Datenfelder: Für Anzeige und Eingabe.

Hier werden folgende **Abkürzungen** verwendet:

**Gleitpunktzahl** für Die Gleitpunktzahl im Ganzen (Datenfeld für Eingaben)

**Alle Ziffern** für Alle Ziffern der Gleitpunktzahl (Datenfeld, nur Anzeige)

**Nächste** für Nächste float-Zahl (Knopf zum Anklicken)

**Vorige** für Vorige float-Zahl (Knopf zum Anklicken)

**Hinweis:** In einem Java-Programm sind `0.1`, `12.345`, `3e5` etc. Literale des Typs `double`. Literale des Typs `float` müssen so notiert werden: `0.1F`, `12.345F`, `3e5F` etc. (oder mit `f` statt `F`).

### 1. Der wahre Wert des float-Literals 0.1F

Eingabe in **Gleitpunktzahl**: `0.1` (mit Return abschließen!)

In **Gleitpunktzahl** steht die übliche Darstellung, in **Alle Ziffern** der wahre Wert.

Frage 1.1: Aus wie vielen Ziffern besteht die Darstellung des wahren Wertes?

Frage 1.2: Ist der wahre Wert kleiner oder größer als ein Zehntel?

### 2. Wie weit liegen float-Werte auseinander?

Eingabe in **Gleitpunktzahl**: `4e6` (mit Return abschließen!)

**Alle Ziffern** lesen, nacheinander **Nächste** und **Vorige** anklicken, evtl. mehrmals.

Frage 2.1: Um wie viel unterscheidet sich der Wert `4e6F` vom nächsten Wert?

Stellen Sie das an Hand der genauen Anzeige in **Alle Ziffern** fest, nicht an Hand der üblichen, aber ungenauen Anzeige in **Gleitpunktzahl**.

Ebenso mit Eingabe von `16e6` und `16e9` in **Gleitpunktzahl**.

Frage 2.2: Um wie viel unterscheidet sich der Wert `16e6F` vom nächsten Wert?

Frage 2.3: Um wie viel unterscheidet sich der Wert `16e9F` vom nächsten Wert?

Frage 2.4: Welchen `float`-Wert bezeichnet das Literal `16000000500.0F`?

Frage 2.5: Welchen `float`-Wert bezeichnet das Literal `16000000600.0F`?

### 3. Kann man float-Berechnungen "rückgängig machen"?

Eingabe in <b>Gleitpunktzahl</b>	Hin-Aktion	Zurück-Aktion	Anzeige in <b>Gleitpunktzahl?</b>
0.1	Knopf <b>Mal 2.0</b> (5 Mal)	Knopf <b>Durch 2.0</b> (5 Mal)	
0.1	Knopf <b>Mal 0.1</b> (5 Mal)	Knopf <b>Durch 0.1</b> (5 Mal)	
0.1	Knopf <b>Mal 0.1</b> (1 Mal)	Knopf <b>Durch 0.1</b> (1 Mal)	
0.1	Knopf <b>Mal 0.1</b> (11 Mal)	Knopf <b>Durch 0.1</b> (11 Mal)	

### 4. Mit wie vielen Ziffern werden float-Werte üblicherweise (z.B. von pln) dargestellt?

Eingabe in **Gleitpunktzahl**:  $1e36$  (mit Return abschließen). Welcher Wert wird in **Gleitpunktzahl** angezeigt? Welcher wahre Wert ist damit gemeint (siehe **Alle Ziffern**)? Tragen Sie diese beiden Anzeigen in die Zeile 0 der folgenden Tabelle ein (von der Anzeige in **Alle Ziffern** genügen die ersten 12 Ziffern).

Zeilen-Nr.	Anzeige aus <b>Gleitpunktzahl</b>	Anzeige aus <b>Alle Ziffern</b> (die ersten 12 Ziffern)
-2		
-1		
0		
+1		
+2		

Ermitteln Sie (mit Hilfe der Knöpfe **Vorige** und **Nächste**) die zwei Vorgänger und die zwei Nachfolger des eingetragenen Wertes und tragen Sie diese ebenfalls in die Tabelle ein (die Vorgänger in die Zeilen -1 und -2, die Nachfolger in die Zeilen +1 und +2).

Vergleichen Sie die genaue Darstellung in **Alle Ziffern** mit der ungenauen, gerundeten Darstellung in **Gleitpunktzahl**. Beschreiben Sie die Regel, nach der die ungenaue Darstellung mal aus mehr und mal aus weniger Ziffern besteht.

### 5. Welche und wie viele Bitkombinationen stellen NaN-Werte dar?

Eingabe in **Gleitpunktzahl**: `nan` (mit Return abschließen!). Welche der 32 Bits sind an bzw. aus?

Eingabe in **Gleitpunktzahl**: `nann` (mit Return abschließen!). Welche der 32 Bits sind an bzw. aus?

Das waren 2 "extreme NaN-Werte". Welche Bits kann man beliebig an- bzw. ausmachen ohne den Bereich der NaN-Werte zu verlassen? Was passiert, wenn ein NaN-Wert angezeigt wird und man dann das Vorzeichenbit verändert?

Frage 5.1: Welche Bitkombinationen stellen NaN-Werte dar?

Die folgende Antwort ist falsch, aber Ihre richtige Antwort sollte eine ähnliche *Form* haben:

Falsche Antwort 5.1: Die Mantissen-Bits müssen alle an sein, von den Bits der Charakteristik müssen mindestens drei aus sein, das Vorzeichenbit kann an oder aus sein.

Frage 5.2: Wie viele Bitkombinationen stellen NaN-Werte dar? Geben Sie die genaue Formel und einen ungefähren Wert an (z.B. genaue Formel:  $2^{10} - 5$ , ungefährender Wert: 1 Tausend).

## 6. Normalisierte und nicht-normalisierte float-Zahlen

Um die Beschreibung etwas zu vereinfachen, beschränken wir uns hier erst mal auf `float`-Zahlen mit positivem Vorzeichen (das Vorzeichenbit **VZ** ist aus, d.h. weiß). Für negative `float`-Zahlen gilt dann alles "ganz entsprechend".

**Normalisierte Zahlen** erkennt man daran, dass mindestens *ein* Bit der Charakteristik an ist. Bei solchen Zahlen gelten alle Bits der Mantisse als *Nachpunktstellen* (im 2-er-System), vor denen man sich eine 1. denken sollte. Prüfen Sie das anhand der Zahlen `1.0`, `1.5`, `1.25`, `1.75` nach.

Besonders kleine (nahe bei 0.0 gelegene) `float`-Werte werden als **nicht-normalisierte Zahlen** dargestellt. Alle Bits der Charakteristik sind aus und den Punkt sollte man sich zwischen dem ersten und zweiten Bit der Mantisse (d.h. zwischen Bit 22 und 21) denken.

Im **GleitBitsFloatApplet** steht im Datenfeld **Zahl ist normalisiert** `true` bzw. `false`, je nachdem ob die aktuelle Zahl normalisiert ist oder nicht.

Die *kleinste normalisierte Zahl* wird angezeigt, wenn man in **Gleitpunktzahl** den Namen `min` eingibt. Drückt man dann auf **Vorige**, sieht man die *größte nicht-normalisierte Zahl*. Gibt man den Namen `min` ein, so wird *die kleinste float-Zahl* angezeigt (und die ist *nicht-normalisiert*).

Frage 6.1: Welche Bits sind an bzw. aus bei der *kleinsten normalisierten Zahl*?

Frage 6.2: Welche Bits sind an bzw. aus bei der *größten nicht-normalisierten Zahl*?

Frage 6.3: Welche Bits sind an bzw. aus bei der *kleinsten (nicht-normalisierten) Zahl*?

Frage 6.4: Welche `float`-Zahl liegt unmittelbar vor der kleinsten Zahl?

## 7. float-Werte mit Namen

In das Datenfeld **Gleitpunktzahl** darf man nicht nur Gleitpunkt-Literale wie z.B. `1.0` oder `-12.3456` oder `+123.4e-7` etc. eingeben, sondern auch bestimmte *Namen* für bestimmte `float`-Werte. Einige dieser Namen wurden bereits in den vorangehenden Teilen dieser Übung erwähnt.

Frage 7.1: Geben Sie die vollständige Liste aller Namen an, die man in das Datenfeld **Gleitpunktzahl** eingeben darf. Zur Beantwortung dieser Frage dürfen Sie sich die *Bedienungsanleitung* des `GleitBitsFloatApplets` ansehen (was sonst natürlich streng verboten ist :-).

Frage 7.2: Aus wie vielen Ziffern besteht der Dezimalbruch, der der kleinsten `float`-Zahl (`min`) exakt entspricht?

### Übung 3: 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*).

Wenn Sie Ihre Lösungen in die Testdatei **Ueb\_If.java** eintragen, können Sie sie gleich testen.

Angenommen, Sie haben die folgenden Vereinbarungen schon geschrieben:

```
1 int n1 = EM.liesInt(); // Liest einen int-Wert von der Tastatur in die Variable n1
2 int n2 = EM.liesInt(); // Liest einen int-Wert von der Tastatur in die Variable n2
3 int n3 = EM.liesInt(); // Liest einen int-Wert von der Tastatur in die Variable n3
4 int n4 = EM.liesInt(); // Liest einen int-Wert von der Tastatur in die Variable n4
5 int n5 = EM.liesInt(); // Liest einen int-Wert von der Tastatur in die Variable n5
6 int max = 17;
```

Befehlen Sie jetzt dem Ausführer mit `if`-Anweisungen, 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 `n1` einen *positiven* Wert enthält, soll ihr Wert *um 13 Prozent erhöht* werden (mit Ganzzahlrechnung, ohne Bruchzahlen!).
3. Falls die Variable `n1` einen *geraden* Wert enthält, soll ihr Wert *halbiert* werden. Sonst soll ihr Wert *verdoppelt* werden. Um festzustellen, ob `n1` gerade ist oder nicht, sollen Sie die Restoperation (Modulo-Operation) `%` verwenden.
4. Lösen Sie 3. noch einmal, aber diesmal ohne Restoperation `%` (nur mit den Operationen `+`, `-`, `*`, `/`). Was versteht man eigentlich unter dem *Rest* einer Ganzzahldivision?
5. 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`).
6. Falls die Variable `n1` 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`).
7. Die größere der zwei Zahlen `n1` und `n2` soll der Variablen `max` zugewiesen werden.
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 `n1` enthält einen Rechnungsbetrag. Falls 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 einfacher 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 4: Ausführen

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

```
L 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);
```



## Übung 5: Schleifen A (ausführen)

Einfache Schleifen werden in den **Abschnitten 4.2.4 bis 4.2.7** des Buches behandelt. Wie man Programme mit Papier und Bleistift ausführt steht 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 den Ausgabebefehlen `p` (der Bildschirmzeiger bleibt unmittelbar hinter den ausgegeben Zeichen stehen) und `pln` (der Bildschirmzeiger rückt nach der Ausgabe zum Anfang der nächsten Zeile vor). Mit `final int ...` wird eine *unveränderbare Variable* („Konstante“) vom Typ `int` vereinbart. Wenn der Ausführer eine Schleife wie `for (int i=1; ...) { ... }` fertig ausgeführt hat, dann *zerstört* er die Schleifen-Variable `i`.

### 5.1. Die anna-Schleife:

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

### 5.2. Die berta-Schleife:

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

### 5.3. Die celia-Schleife:

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

### 5.4. Die dora-Schleife:

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

### 5.5. 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 }
```

### 5.6. 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 }
```

## Übung 6: Schleifen B (programmieren)

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. Bei Zahlenfolgen (zum Beispiel: -5 -2 1 4 7 etc.) ist häufig die *Differenzenfolge* (im Beispiel: 3, 3, 3, 3 etc.) hilfreich.

**Achtung:** Alle Ausgaben sollen mit der Methode `p` erfolgen!

Wenn Sie Ihre Lösungen in die Testdatei **Ueb\_SchleifenB.java** eintragen, können Sie sie gleich testen.

1. Programmieren Sie eine Schleife, die die Ganzzahlen von 1 bis 10 (alle auf einer Zeile, mit je einem Blank nach jeder Zahl) zum Bildschirm ausgibt (mit der Methode `p`), 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 ausgibt (mit der Methode `p`), 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.:** In Java gibt es keinen Potenz-Operator und keine Potenz-Funktion für Ganzzahlen.

Für die folgenden Aufgaben sollen Sie annehmen, dass eine Ganzzahl-Variable namens `norbert` vereinbart und mit einer Zahl *größer oder gleich 2* initialisiert wurde, etwa so:

```
int norbert = Math.max(2, EM.liesInt());
```

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 `true` bzw. `false` auszugeben (mit der Methode `p`), 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 (natürlich mit der Methode `p`).

11. Befehlen Sie dem Ausführer die *Dezimalziffern* ('0', '1', ... '9') in `sara` zu zählen und ihre 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 ihre Anzahl auszugeben (wie immer mit der Methode `p`).

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 (Nein, nicht mit `println!`).

## Übung 7: Schleifen C (programmieren)

Die folgenden Übungsaufgaben sollen mit geschachtelten `for`-Schleifen und den Anweisungen `break` und `continue` gelöst werden (auch `break` und `continue` *mit Marken*, siehe dazu im Buch "Java ist eine Sprache" im Abschnitt 4.2.8 Beispiel-03 bis Beispiel-05). Wo möglich sollen `for-each`-Schleifen (statt `for-i`-Schleifen) verwendet werden. Wenn es nicht möglich ist, `for-each`-Schleifen zu verwenden, sollen Sie kurz begründen, warum.

*Nicht* verwendet werden sollen zur Lösung dieser Übungsaufgaben *Sammlungen* (z.B. solche des Typs `ArrayList`) und Methoden aus den Klassen `Arrays` und `Array`. Die Methoden sollen ihre Reihenfolgeparameter grundsätzlich *nicht verändern* (nur lesen) und auch *keine Kopien* davon erzeugen (zu aufwendig).

Wenn Sie die folgenden Methoden in die Datei **Ueb\_SchleifenC.java** eintragen, können Sie sie gleich testen.

```
1 // -----
2 static public boolean enthaeltGleiche(int[] ir) {
3     // Liefert true genau dann wenn mindestens zwei Komponenten
4     // von ir gleich sind. Beispiele:
5     // int[] ir01 = {1, 2, 1, 3};
6     // int[] ir02 = {1, 2, 2, 1, 3, 3, 3};
7     // int[] ir03 = {1, 2, 3, 4};
8     // int[] ir04 = {5};
9     // int[] ir05 = {};
10    //
11    // enthaeltGleiche(ir01) ist gleich true
12    // enthaeltGleiche(ir02) ist gleich true
13    // enthaeltGleiche(ir03) ist gleich false
14    // enthaeltGleiche(ir04) ist gleich false
15    // enthaeltGleiche(ir05) ist gleich false
16
17    return false; // DIESE ZEILE SOLL ERSETZT WERDEN
18 } // enthaeltGleiche
19
20 // Frage: for-each-Schleifen oder for-i-Schleifen?
21 // -----
22 static public int anzahlVerschiedene(int[] ir) {
23     // Liefert die Anzahl der verschiedenen Komponenten von ir. Beispiele:
24     // int[] ir01 = {10, 20, 30, 40, 50};
25     // int[] ir02 = {10, 20, 20, 10, 50};
26     // int[] ir03 = {10, 10, 10};
27     // int[] ir04 = {10};
28     // int[] ir05 = {};
29     //
30     // anzahlVerschiedene(ir01) ist gleich 5
31     // anzahlVerschiedene(ir02) ist gleich 3
32     // anzahlVerschiedene(ir03) ist gleich 1
33     // anzahlVerschiedene(ir04) ist gleich 1
34     // anzahlVerschiedene(ir05) ist gleich 0
35
36    return -999; // DIESE ZEILE SOLL ERSETZT WERDEN
37 } // anzahlVerschiedene
38
39 // Frage: for-each-Schleifen oder for-i-Schleifen?
40 // -----
41
```

```
42 static public boolean enthaelt(int[] irA, int[] irB) {
43     // Liefert true genau dann wenn irB in irA enthalten ist
44     // (d.h. wenn jeder int-Wert, der (mind. einmal) in irB vorkommt
45     // auch (mind. einmal) in irA vorkommt). Beispiele:
46     // int[] ir01 = {10, 20, 30, 40, 50};
47     // int[] ir02 = {50, 10, 50, 10, 50};
48     // int[] ir03 = {30, 20, 10};
49     // int[] ir04 = {10, 60, 10};
50     // int[] ir05 = {40};
51     // int[] ir06 = {60};
52     // int[] ir07 = {};
53     //
54     // enthaelt(ir01, ir02) ist gleich true
55     // enthaelt(ir01, ir03) ist gleich true
56     // enthaelt(ir01, ir04) ist gleich false
57     // enthaelt(ir01, ir05) ist gleich true
58     // enthaelt(ir01, ir06) ist gleich false
59     // enthaelt(ir01, ir07) ist gleich true
60     // enthaelt(ir02, ir01) ist gleich false
61     // enthaelt(ir06, ir07) ist gleich true
62     // enthaelt(ir07, ir07) ist gleich true
63
64     return false; // DIESE ZEILE SOLL ERSETZT WERDEN
65
66 } // enthaelt
67
68 // Frage: for-each-Schleifen oder for-i-Schleifen?
69 // -----
70 static public int[] vereinigung(int[] irA, int[] irB) {
71     // Verlaesst sich darauf, dass irA und irB Mengen repraesentieren
72     // (d.h. keine Doppelgaenger enthalten). Liefert die Vereinigung
73     // von irA und irB. Beispiele:
74     // int[] ir01 = {10,20,30,40};
75     // int[] ir02 = {20,30,40,50};
76     // int[] ir03 = {30,60,70};
77     // int[] ir04 = {10};
78     // int[] ir05 = {50};
79     //
80     // int[] er02 = {10,20,30,40,50};
81     // int[] er03 = {10,20,30,40,60,70};
82     // int[] er04 = {10,20,30,40};
83     // int[] er05 = {10,20,30,40,50};
84     //
85     // vereinigung(ir01, ir02) ist gleich er02
86     // vereinigung(ir01, ir03) ist gleich er03
87     // vereinigung(ir01, ir04) ist gleich er04
88     // vereinigung(ir01, ir05) ist gleich er05
89
90     return new int[]{-999, -999}; // DIESE ZEILE SOLL ERSETZT WERDEN
91
92 } // vereinigung
93 // -----
94 static public int[] durchschnitt(int[] irA, int[] irB) {
95     // Verlaesst sich darauf, dass irA und irB Mengen repraesentieren
96     // (d.h. keine Doppelgaenger enthalten). Liefert den Durchschnitt
97     // von irA und irB. Beispiele:
98     // int[] ir01 = {10, 20, 30, 40};
99     // int[] ir02 = {20, 30, 40, 50};
100    // int[] ir03 = {30, 60, 70};
101    // int[] ir04 = {10};
102    // int[] ir05 = {};
103    //
104    // int[] er02 = {20, 30, 40};
105    // int[] er03 = {30};
106    // int[] er04 = {10};
107    // int[] er05 = {};
108    //
109    // durchschnitt(ir01, ir02) ist gleich er02
110    // durchschnitt(ir01, ir03) ist gleich er03
111    // durchschnitt(ir01, ir04) ist gleich er04
112    // durchschnitt(ir01, ir05) ist gleich er05
113
114    return new int[]{-999, -999}; // DIESE ZEILE SOLL ERSETZT WERDEN
115
116 } // durchschnitt
```

```
117 // -----
118 static public int[] differenz(int[] irA, int[] irB) {
119     // Verlaesst sich darauf, dass irA und irB Mengen repraesentieren
120     // (d.h. keine Doppelgaenger enthalten). Liefert die (Mengen-)
121     // Differenz irA minus irB (die enthaelt alle Elemente von irA,
122     // die nicht in irB enthalten sind). Beispiele:
123     // int[] ir01 = {10, 20, 30, 40};
124     // int[] ir02 = {10, 20, 50};
125     // int[] ir03 = {20, 50};
126     // int[] ir04 = {20};
127     // int[] ir05 = {};
128     // int[] ir06 = {10, 20, 30, 40};
129     //
130     // int[] er02 = {30, 40};
131     // int[] er03 = {10, 30, 40};
132     // int[] er04 = {10, 30, 40};
133     // int[] er05 = {10, 20, 30, 40};
134     // int[] er06 = {};
135     //
136     // differenz(ir01, ir02) ist gleich er02
137     // differenz(ir01, ir03) ist gleich er03
138     // differenz(ir01, ir04) ist gleich er04
139     // differenz(ir01, ir05) ist gleich er05
140     // differenz(ir05, ir06) ist gleich er06
141
142     return new int[]{-999, -999}; // DIESE ZEILE SOLL ERSETZT WERDEN
143
144 } // differenz
145
146 // Frage: for-each-Schleifen oder for-i-Schleifen?
147 // -----
148 static public int[] symmetrischeDifferenz(int[] irA, int[] irB) {
149     // Verlaesst sich darauf, dass irA und irB Mengen repraesentieren
150     // (d.h. keine Doppelgaenger enthalten). Liefert die symmetrische
151     // Differenz von irA und irB (die enthaelt alle Elemente, die nur
152     // in einer der beiden Mengen enthalten ist, aber nicht in beiden).
153     // Beispiele:
154     // int[] ir01 = {10,20,30,40};
155     // int[] ir02 = {10,20,50};
156     // int[] ir03 = {20,50};
157     // int[] ir04 = {50};
158     // int[] ir05 = {10,20,30,40};
159     // int[] ir06 = {};
160     //
161     // int[] er02 = {30,40,50};
162     // int[] er05 = {};
163     // int[] er05 = {10,20,30,40};
164     //
165     // symmetrischeDifferenz(ir01, ir02) ist gleich er02
166     // symmetrischeDifferenz(ir01, ir03) ist gleich er02
167     // symmetrischeDifferenz(ir01, ir04) ist gleich er02
168     // symmetrischeDifferenz(ir01, ir05) ist gleich er05
169     // symmetrischeDifferenz(ir01, ir06) ist gleich er06
170
171     return new int[]{-999, -999}; // DIESE ZEILE SOLL ERSETZT WERDEN
172
173 } // symmetrischeDifferenz
174
175 // Frage: for-each-Schleifen oder for-i-Schleifen?
176 // -----
```

## Übung 8: Methoden A

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 als *Funktionen*.

Wenn Sie die folgenden Methoden in die Datei **Ueb\_MethodenA.java** eintragen, können Sie sie gleich testen.

```
1 // -----
2 static public int summe(int n1, int n2) {
3     // Liefert die Summe von n1 und n2
4
5     return -999; // DIESE ZEILE SOLL ERSETZT WERDEN
6 } // summe
7 // -----
8 static public void gibAus(int a, int b, int c) {
9     // Gibt die Namen der Parameter (jeden auf einer neuen Zeile) gefolgt
10    // von einem Doppelpunkt ":" und dem Wert des Parameters aus
11    // (mit dem Befehl pln). Auf dem Bildschirm soll das etwa so aussehen:
12    //
13    // a: 17
14    // b: -22
15    // c: 5
16
17    pln("Noch nicht ersetzt!"); // DIESE ZEILE SOLL ERSETZT WERDEN
18
19 } // gibAus
20 // -----
21 static public int hochZwei(int grundzahl) {
22     // Liefert den Wert grundzahl hoch 2.
23     // Beispiele:
24     // hochZwei( 1) ist gleich 1.
25     // hochZwei( 2) ist gleich 4.
26     // hochZwei( 3) ist gleich 9.
27     // hochZwei(-3) ist gleich 9.
28
29     return -999; // DIESE ZEILE SOLL ERSETZT WERDEN
30
31 } // hochZwei
32 // -----
33 static public int zweiHoch(int exponent) {
34     // Liefert den Wert 2 hoch exponent.
35     // Falls der exponent kleiner oder gleich 0 ist, wird 1 als
36     // Ergebnis geliefert.
37     // Beispiele:
38     // zweiHoch( 0) ist gleich 1
39     // zweiHoch( 1) ist gleich 2
40     // zweiHoch( 2) ist gleich 4
41     // zweiHoch( 3) ist gleich 8
42     // zweiHoch(-3) ist gleich 1
43
44     return -999; // DIESE ZEILE SOLL ERSETZT WERDEN
45
46 } // zweiHoch
47 // -----
48 static public int hoch(int grundzahl, int exponent) {
49     // Liefert den Wert grundzahl hoch exponent.
50     // Falls der exponent kleiner oder gleich 0 ist, wird 1 als
51     // Ergebnis geliefert.
52     // Beispiele:
53     // hoch( 2,  0) ist gleich 1
54     // hoch( 2,  1) ist gleich 2
55     // hoch( 2,  3) ist gleich 8
56     // hoch( 3,  2) ist gleich 9
57     // hoch(-3,  2) ist gleich 9
58     // hoch( 3, -2) ist gleich 1
59
60     return -999; // DIESE ZEILE SOLL ERSETZT WERDEN
61
62 } // hoch
63 // -----
```

```
64     static public boolean istPrim(final int N) {
65         // Liefert true, falls N prim ist, und sonst false.
66         // Liefert false fuer alle N die kleiner/gleich 1 sind.
67         // Beispiele:
68         // istPrim( 1) ist gleich false
69         // istPrim( 2) ist gleich true
70         // istPrim( 3) ist gleich true
71         // istPrim( 4) ist gleich false
72         // istPrim( 5) ist gleich true
73         // istPrim( 5) ist gleich true
74         // istPrim(17) ist gleich true
75         // istPrim( 0) ist gleich false
76         // istPrim(-3) ist gleich false
77
78         return false; // DIESE ZEILE SOLL ERSETZT WERDEN
79
80     } // istPrim
81     // -----
82     static public int primDublette(int minimum) {
83         // Liefert die kleinste Primzahl p fuer die gilt:
84         // p ist groesser oder gleich minimum
85         // p+2 ist kleiner oder gleich Integer.MAX_VALUE
86         // p+2 ist auch eine Primzahl
87         // Gibt es keine solche Primzahl p, wird 0 als Ergebnis geliefert.
88         // Beispiele:
89         // primDublette(2) ist gleich 3 (weil 3 und 5 prim sind)
90         // primDublette(3) ist gleich 3 (weil 3 und 5 prim sind)
91         // primDublette(4) ist gleich 5 (weil 5 und 7 prim sind)
92         // primDublette(6) ist gleich 11 (weil 11 und 13 prim sind)
93         // primDublette(2_147_482_949) ist gleich 2_147_482_949
94         // primDublette(2_147_482_950) ist gleich 0 (weil es innerhalb
95         // des Typs int keine Primdublette oberhalb von 2_147_482_949 gibt)
96
97         return -999; // DIESE ZEILE SOLL ERSETZT WERDEN
98
99     } // primDublette
100    // -----
101    static public void alleRechtwinkligen() {
102        // Gibt alle Ganzzahltripel (a, b, c) aus fuer die gilt:
103        // 1. a, b und c liegen zwischen 1 und 100 (einschliesslich)
104        // 2. a ist groesser/gleich b und b ist groesser/gleich c
105        // 3. a*a ist gleich b*b + c*c (Pythagoras)
106        // Die Ausgabe *eines* solchen Ganzzahltripels soll so aussehen:
107        // Nr. 19: 50, 40, 30.
108        // und endet mit einem Blank '.' und einem Zeilenwechsel '\n'.
109        // Jedes Tripel wird mit einem Aufruf von pln ausgegeben.
110
111        pln("Noch nicht ersetzt!"); // DIESE ZEILE SOLL ERSETZT WERDEN
112
113    } // alleRechtwinkligen
114    // -----
```

## Übung 9: Methoden B

Wenn Sie die folgenden Methoden in die Datei **Ueb\_MethodenB.java** eintragen, können Sie sie gleich testen.

```
1 // -----
2 static char[] a2b(char[] r) {
3     // Liefert eine Kopie von r, in der alle Vorkommen von 'a' durch
4     // 'b' ersetzt wurden.
5
6     return new char[]{'?'}; // DIESE ZEILE SOLL ERSETZT WERDEN
7
8 } // a2b
9 // -----
10 static char[] a2bb(char[] r) {
11     // Liefert eine Kopie von r, in der jedes Vorkommen von 'a' durch
12     // je zwei 'b' ersetzt wurden.
13
14     return new char[]{'?'}; // DIESE ZEILE SOLL ERSETZT WERDEN
15
16 } // a2bb
17
```

**Hinweis (zu a2bb):** 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.

```
18 // -----
19 static char[] aa2bbb(char[] r) {
20     // Liefert eine Kopie von r, in der jedes Vorkommen von zwei unmittel-
21     // bar nacheinander liegenden 'a' durch drei 'b' ersetzt wurden.
22     // Nur nicht-ueberlappende Vorkommen von "Doppel-a" werden ersetzt.
23
24     return new char[]{'?'}; // DIESE ZEILE SOLL ERSETZT WERDEN
25
26 } // aa2bbb
```

**Hinweis (zu aa2bbb):** Auch hier soll die Ergebnis-Reihung "genau die richtige Länge" haben. Außerdem soll gelten: Eine Reihung {'a', 'a', 'a'} soll in die folgende Reihung der Länge 4 übersetzt werden: {'b', 'b', 'b', 'a'} (und nicht die Reihung der Länge 6: {'b', 'b', 'b', 'b', 'b', 'b'}).

**Hinweis (zu c2d, c2dd und cc2ddd):** Zur Übung sollen Sie keine replace-Methode der Klasse String verwenden, sondern nur die replace-Methode der Klasse StringBuilder!

```
27 // -----
28 public static String c2d(String s) {
29     // Liefert eine Kopie von s, in der alle Vorkommen von 'a' durch
30     // 'b' ersetzt wurden.
31
32     return "Noch nicht ersetzt!"; // DIESE ZEILE SOLL ERSETZT WERDEN
33
34 } // c2d
35 // -----
36 public static String c2dd(String s) {
37     // Liefert eine Kopie von s, in der jedes Vorkommen von 'a' durch
38     // je zwei 'b' ersetzt wurde.
39
40     return "Noch nicht ersetzt!"; // DIESE ZEILE SOLL ERSETZT WERDEN
41
42 } // c2dd
43 // -----
44 public static String cc2ddd(String s) {
45     // Liefert eine Kopie von s, in der jedes Vorkommen von zwei unmit-
46     // telbar nacheinander liegenden 'c' durch drei 'd' ersetzt wurden.
47     // Nur nicht-ueberlappende Vorkommen von "Doppel-as" werden ersetzt.
48
49     return "Noch nicht ersetzt!"; // DIESE ZEILE SOLL ERSETZT WERDEN
50
51 } // cc2ddd
```



## Übung 10: Reihungen A

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

Wenn Sie die folgenden Methoden in die Datei **Ueb\_ReihungenA.java** eintragen, können Sie sie gleich testen.

```
1 // -----
2 public static String printR(int[] ir) {
3     // Liefert "null", wenn ir gleich null ist.
4     // Liefert "[]" wenn ir leer ist.
5     // Liefert "[123]" wenn ir nur eine Komponente 123 enthaelt.
6     // Liefert einen String wie z.B. "[123, -17, 55]", wenn ir
7     // die 3 Komponenten 123, -17 und 55 Komponenten enthaelt.
8
9     return "Noch nicht ersetzt!"; // DIESE ZEILE SOLL ERSETZT WERDEN
10
11 } // printR
12 // -----
13 public static int min(int[] ir) {
14     // Liefert die kleinste Komponente von ir.
15     // Liefert Integer.MAX_VALUE, wenn ir leer ist.
16
17     return -999; // DIESE ZEILE SOLL ERSETZT WERDEN
18
19 } // min
20 // -----
21 public static boolean mindEineGerade(int[] ir) {
22     // Liefert true genau dann wenn
23     // ir mindestens eine gerade Zahl enthaelt.
24
25     return false; // DIESE ZEILE SOLL ERSETZT WERDEN
26
27 } // mindEineGerade
28 // -----
29 public static boolean alleGerade(int[] ir) {
30     // Liefert true genau wenn
31     // alle Komponenten von ir gerade Zahlen sind.
32
33     return false; // DIESE ZEILE SOLL ERSETZT WERDEN
34
35 } // alle Gerade
36 // -----
37 public static boolean kommtVor(int n, int[] ir) {
38     // Liefert true genau wenn
39     // n (mindestens einmal) in ir vorkommt.
40
41     return false; // DIESE ZEILE SOLL ERSETZT WERDEN
42
43 } // kommtVor
44 // -----
45 public static int index(int n, int[] ir) {
46     // Liefert den kleinsten Index i fuer den gilt n == ir[i]
47     // bzw. -1, falls es keinen solchen Index gibt
48     // (d.h. falls n in ir nicht vorkommt).
49
50     return -999; // DIESE ZEILE SOLL ERSETZT WERDEN
51
52 } // index
53 // -----
54 public static void procAlleDurch2(int[] ir) {
55     // Teilt jede Komponente von ir durch 2 (d.h. ir wird veraendert)
56
57     return; // DIESE ZEILE SOLL ERSETZT WERDEN
58
59 } // pAlleDurch2
60 // -----
61 public static int[] funkAlleDurch2(int[] ir) {
62     // Liefert eine Kopie der Reihung ir, in der alle Komponenten durch 2
63     // geteilt wurden (d.h. ir wird nicht veraendert).
64
65     return new int[]{-999, -999, -999}; // DIESE ZEILE SOLL ERSETZT WERDEN
66
67 } // fAlleDurch2
```

```
68 // -----
69 public static boolean sindGleich(int[] ira, int[] irb) {
70     // Liefert true genau dann wenn
71     // 1. ira und irb gleich lang sind und
72     // 2. jede Komponente ira[i] gleich der entsprechenden
73     // Komponente irb[i] ist.
74
75     return false; // DIESE ZEILE SOLL ERSETZT WERDEN
76
77 } // sindGleich
78 // -----
79 public static boolean sindDisjunkt(int[] ira, int[] irb) {
80     // Liefert true genau dann wenn
81     // jede Zahl in ira verschieden ist von jeder Zahl in irb
82     // (d.h. wenn keine Zahl in ira und in irb vorkommt)
83
84     return false; // DIESE ZEILE SOLL ERSETZT WERDEN
85
86 } // sindDisjunkt
87 // -----
88 public static String printR(String[] sr) {
89     // Liefert "null"      , wenn sr gleich null ist.
90     // Liefert "[]"       , wenn sr leer ist.
91     // Liefert "[\"ABC\"]" , wenn sr nur eine Komponente "ABC" enthaelt.
92     // Liefert einen String wie z.B. "[\"ABC\", \"12\", \"\", null]" aus,
93     // wenn sr die 4 Komponenten "ABC", "12", "" und null enthaelt.
94
95     return "Noch nicht ersetzt!"; // DIESE ZEILE SOLL ERSETZT WERDEN
96
97 } // printR
98 // -----
```

## Übung 11: Reihungen B

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

Wenn Sie die folgenden Methoden in die Datei **Ueb\_ReihungenB.java** eintragen, können Sie sie gleich testen.

```
1 // -----
2 static public int summe(int[][] irr) {
3     // Liefert die Summe aller int-Komponenten von irr.
4
5     return -999; // DIESE ZEILE SOLL ERSETZT WERDEN
6
7 } // summe
8 // -----
9 static public boolean alleGerade(int[][] irr) {
10    // Liefert true genau dann wenn
11    // alle int-Komponenten von irr gerade sind.
12
13    return false; // DIESE ZEILE SOLL ERSETZT WERDEN
14
15 } // alleGerade
16 // -----
17 static public int anzahlUngerade(int[][] irr) {
18    // Liefert die Anzahl der ungeraden int-Komponenten von irr.
19
20    return -999; // DIESE ZEILE SOLL ERSETZT WERDEN
21
22 } // anzahlUngerade
23 // -----
24 static public boolean kommtVor(int n, int[][] irr) {
25    // Liefert true genau dann wenn n
26    // (als int Komponente) in irr vorkommt.
27
28    return false; // DIESE ZEILE SOLL ERSETZT WERDEN
29
30 } // kommtVor
31 // -----
32 static public int anzahlIntKomponenten(int[][] irr) {
33    // Liefert die Anzahl der int-Komponenten von irr.
34
35    return -999; // DIESE ZEILE SOLL ERSETZT WERDEN
36
37 } // anzahlIntKomponenten
38 // -----
39 static public int anzahlBuchstaben(String[] sr) {
40    // Liefert die Anzahl der Buchstaben, die in den String-Komponenten
41    // von sr vorkommen. Als Buchstaben sollen hier die 52 Zeichen
42    // 'A' bis 'Z' und 'a' bis 'z' gelten.
43
44    return -999; // DIESE ZEILE SOLL ERSETZT WERDEN
45
46 } // anzahlBuchstaben
47 // -----
```

**Übung 12: Reihungen C (mehrstufige, als Bojen)**

*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*.

## Übung 13: Sammlungen

In dieser Übung geht es um Sammlungen (engl. collections) der Typen `ArrayList` und `TreeSet`.

1. Als Grundlage benötigen Sie die Datei **Ueb\_Sammlungen.java**. Sie sollen ein paar Befehle in diese Datei einfügen (genaue Instruktionen stehen als Kommentare in der Datei). Danach kann man die Datei dem Ausfühler übergeben (z.B. compilieren) und ausführen lassen.

Lesen Sie die Datei `Ueb_Sammlungen.java` durch und finden Sie heraus, was jeder einzelne Befehl darin bewirkt. Das wird Ihnen das Lösen der folgenden Teilübungen erleichtern.

2. Angenommen, die folgenden Befehle stehen in einem Java-Programm (z.B. in der `main`-Methode):

```

1      TreeSet<String> tina = new TreeSet<String>();
2
3      tina.add("CCC");
4      tina.add("AAA");
5      tina.add("EEE");
6
7      pln(tina);
8      pln(tina.first());
9      pln(tina.lower ("BBB"));
10     pln(tina.lower ("CCC"));
11     pln(tina.floor ("BBB"));
12     pln(tina.floor ("CCC"));
13     pln(tina.ceiling("CCC"));
14     pln(tina.ceiling("DDD"));
15     pln(tina.higher ("CCC"));
16     pln(tina.higher ("DDD"));
17     pln(tina.last());

```

Geben Sie von jedem der `pln`-Befehle in den Zeilen 7 bis 17 an, was er zum Bildschirm ausgibt.

Wie immer ist auch hier `pln` eine Abkürzung für `System.out.println`.

3. Sei `toni` eine beliebige, aber nicht-leere `TreeSet`-Sammlung

(`toni` enthält also mindestens *eine* Komponente).

Beschreiben Sie (auf Deutsch), welche Komponenten die Ausdrücke in der folgenden Tabelle bezeichnen:

Ausdruck	Welche Komponente der Sammlung <code>toni</code> bezeichnet (oder: liefert) der Ausdruck?
<code>toni.first()</code>	
<code>toni.lower(k)</code>	
<code>toni.floor(k)</code>	
<code>toni.ceiling()</code>	
<code>toni.higher(k)</code>	
<code>toni.last(k)</code>	

4. Was bezeichnen (oder: liefern) die obigen Ausdrücke (die in der Tabell), wenn `toni` eine *leere* `TreeSet`-Sammlung ist?

## Übung 14: Bojen A

*Bojen* werden in den **Abschnitten 5.7, 7.1 und 10.2 des Buches** behandelt. Im **Abschnitt 12.1** werden Klassen und Objekte beschrieben und auf **S. 279** ist eine `Person01`-Variable als Boje dargestellt.

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, und zwar mit den den folgenden Befehlen:

```
51 ...
52 Otto ob1 = new Otto(17, "Hallo ");
53 Otto ob2 = new Otto(25, "Sonja!" );
54 Otto ob3 = new Otto();
55 ...
```

Später wurden die Objekte durch folgende Befehle verändert:

```
56 ...
57 ob1.zahl = ob2.zahl;
58 ob1.text = ob2.text;
59 ob3.text = ob1.text;
60 ...
```

1. Wie sahen die Variablen `ob1` bis `ob3` unmittelbar nach ihrer Erzeugung aus?

Stellen Sie die drei Variablen als Bojen dar.

2. Wie sahen die Variablen `ob1` bis `ob3` nach Ausführung der drei Zuweisungen aus?

Stellen Sie die drei veränderten Variablen als Bojen dar.

## Übung 15: Bojen B (von Reihungen)

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).

### 15.1. Die Reihung ali:

```
1 int[] ali = new int[] {25, 50, 34, 87};
2 for (int i=ali.length-1; i>=0; i--) {
3     ali[i]++;
4 }
```

### 15.2. Die Reihung bernd:

```
5 int[] bernd = new int[] {12, 19, 23, 10, 15};
6 for (int i=1; i<bernd.length-1; i++) {
7     bernd[i] = (bernd[i-1] + bernd[i] + bernd[i+1]) / 3;
8 }
```

### 15.3. Die Reihung christian:

```
9 int[] christian = new int[] {6, 7, 8, 9};
10 for (int i=0; i<christian.length; i++) {
11     if (christian[i] % 2 == 0) {
12         christian[i] /= 2;
13     } else if (christian[i] % 3 == 0) {
14         christian[i] /= 3;
15     } else {
16         christian[i] = 0;
17     }
18 }
```

### 15.4. Die Reihung dirk:

```
19 StringBuilder[] dirk = {
20     new StringBuilder("Auf-"),
21     new StringBuilder("der-"),
22     new StringBuilder("Mauer.")
23 };
24 for (int i=1; i<dirk.length; i++) {
25     dirk[i].insert(0, dirk[i-1]);
26 }
```

### 15.5. Die Reihung ertan:

```
27 StringBuilder[] ertan = new StringBuilder[3];
28 ertan[0] = new StringBuilder("eins");
29 ertan[1] = new StringBuilder("zwei");
30 ertan[2] = new StringBuilder("drei");
31 ertan[1].append("mal");
32 ertan[0].setCharAt(0, 'E');
33 ertan[2] = new StringBuilder("vier");
```

### 15.6. Die Reihung frank:

```
34 int[] frank = new int[] {22, 78, 78, 35, 22};
35 for (int i1=0; i1<frank.length-1; i1++) {
36     for (int i2=i1+1; i2<frank.length; i2++) {
37         if (frank[i1] == frank[i2]) {
38             ++frank[i1];
39             --frank[i2];
40         }
41     }
42 }
```

## Übung 16: Klassen A

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:** Wie viele *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:** Wie viele *Module* existieren in diesem Moment und wie heißen diese Module?

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

**Frage 11:** Wie viele *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:** Wie viele *Module* existieren in diesem Moment und wie heißen diese Module?

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

**Frage 14:** Wie viele *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:** Wie viele *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 17: Klassen B

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). Dieser eine Befehl enthält mehrere Unterbefehle. Den gesamten Befehl (zusammen mit seinen Unterbefehlen) kann man etwa so ins Deutsche übersetzen:

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

1. Vereinbarungen von Klasselementen:

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 weise ihn der Variablen *anna* zu.

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 Standard-Anfangswert 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:

```

1 class Yehyahan {
2     private             long    wisam;
3     public              double  erdogan = 99.9;
4     private static String selemon = "Hallo! ";
5     public              void    ertan(String s) {
6         wisam = erdogan / 3.0 + 17;
7         pln(selemon + s);
8     }
9     public              void    selcuk() {
10        pln("Ihr Name?");
11        String name = EM.liesString();
12        ertan(name);
13    }
14    public              long    raed() {
15        return wisam;
16    }
17 } // class Yehyahan

```

## Übung 18: Klassen C

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
```

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, wie viele *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 IntRech02

```

## 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 bil = EM.liesBigInteger();
16            if (bil.compareTo(BigInteger.ZERO) == 0) break;
17            p ("D BigInteger BI2? ");
18            BigInteger bi2 = EM.liesBigInteger();
19
20            pln("E BI1:          " + bil);
21            pln("F BI2:          " + bi2);
22
23            pln("G BI1 + BI2:       " + bil.add(bi2));
24            pln("H BI1 - BI2:       " + bil.subtract(bi2));
25            pln("I BI1 * BI2:       " + bil.multiply(bi2));
26            pln("J BI1 / BI2:       " + bil.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 = bil.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:   " + bil.mod(bi2));
39        } // while
40        pln("N BigInteger01: Das war's erstmal!");
41    } // end main
```

## Übung 19: Klassen D

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.

```
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 20: Strings

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

1. Wie viele Konstruktoren hat die Klasse `StringBuilder`? Schauen Sie in ihrer Lieblingsdokumentation der Java-Standardklassen nach.
2. Wie viele Konstruktoren hat die Klasse `String`?
3. In der Klasse `String` gibt es eine Methode `String valueOf(char[] data)`. Geben Sie eine (kurze) 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 }
```

## Übung 21: Ausnahmen A

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   println();
```

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("ABCDEFGF") 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 Ausnahmen20 { // 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("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
```

**Hinweis:** Die Meldung in einem `NumberFormatException`-Objekt sieht z.B. so aus:

For input string: "xyz"

wobei "xyz" der String ist, der nicht umgewandelt werden konnte.

## Übung 22: Ausnahmen B

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-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 23: E01Punkt etc.**

Diese Übung soll Sie dabei unterstützen, sich mit den drei Klassen `E01Punkt`, `E01Rechteck` und `E01Quadrat` genau vertraut zu machen. Die Vereinbarungen dieser Klassen finden Sie im Buch auf den Seiten 287, 288 und 290.

Die Klasse `E01Punkt` erbt von der Klasse `Object` ein paar Elemente. Diese Elemente sollen in dieser Übung *nicht* berücksichtigt und *nicht* mitgezählt werden.

1. Wie viele Elemente werden in der Klasse `E01Punkt` vereinbart und wie heißen diese Elemente?
2. Wie viele Elemente werden in der Klasse `E01Rechteck` vereinbart und wie heißen diese Elemente?
3. Wie viele Elemente werden in jedes `E01Rechteck`-Objekt eingebaut und wie heißen diese Elemente?
4. Wie viele Elemente werden in der Klasse `E01Quadrat` vereinbart und wie heißen diese Elemente?
5. Wie viele Elemente werden in jedes `E01Quadrat`-Object eingebaut und wie heißen diese Elemente?

Betrachten Sie die folgenden Vereinbarungen:

```
1 E01Punkt    p01 = new E01Punkt    ( 3.0, -4.0);
2 E01Rechteck r01 = new E01Rechteck(-3.0,  4.0, 2.0, 5.0);
3 E01Quadrat  q01 = new E01Quadrat (-4.0, -3.0, 2.0);
```

Wenn die folgenden Befehle in der angegebenen Reihenfolge ausgeführt werden, was geben dann die `println`-Befehle (zum Bildschirm) aus?

```
4                                     // Ausgabe
5
6 println("PA " + p01.text());        //
7
8 println("PB " + p01.urAbstand());    //
9
10 println("PC " + p01.toString());    //
11
12 p01.urSpiegeln();
13
14 println("PD " + p01.toString());    //
15
16 println("RA " + r01.text());        //
17
18 println("RB " + r01.urAbstand());    //
19
20 println("RC " + r01.toString());    //
21
22 r01.urSpiegeln();
23
24 println("RD " + r01.toString());    //
25
26 println("QA " + q01.text());        //
27
28 println("QB " + q01.urAbstand());    //
29
30 println("QC " + q01.toString());    //
31
32 q01.urSpiegeln();
33
34 println("QD " + q01.toString());    //
```

### Übung 24: 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	

**Übung 25: 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, ...).

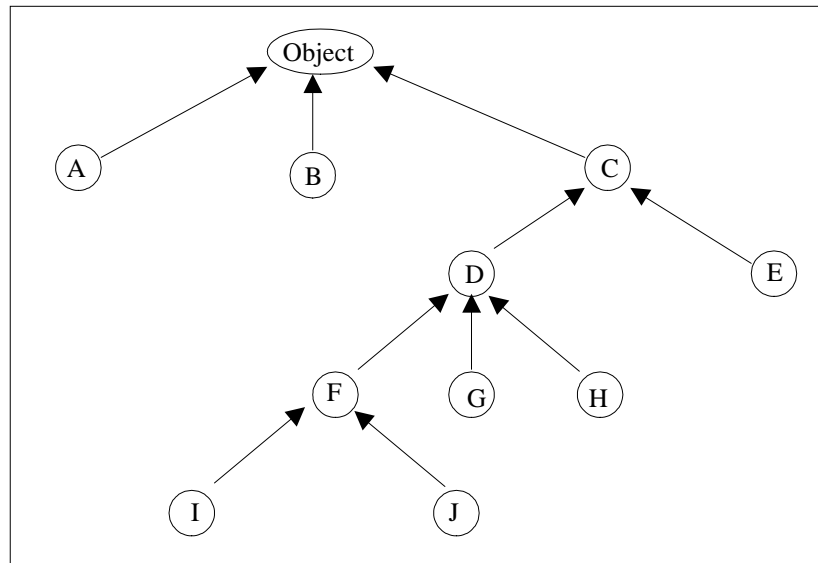
Ein Pfeil von K2 nach K1 bedeutet (verschiedene alternative Formulierungen):

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 26: 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 jede 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`-Literale 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
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					

## Übung 27: 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, 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:

```
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 }
```

## Übung 28: Histogramme

Ein Histogramm ist eine Liste von Häufigkeiten oder Anzahlen. Beispiel: Die Häufigkeiten von Sonnentagen in Berlin in den Monaten des Jahres 2005 (die Zahlen sind frei erfunden):

Monat	Jan	Feb	März	Apr	Mai	Juni	Juli	Aug	Sept	Okt	Nov	Dez
Anz. Sonnentage	12	8	10	11	15	17	16	20	16	17	9	8

1. Schreiben Sie eine Methode, die der folgenden Spezifikation entspricht:

```

1  static int[] histoBuchstaben(String s) {
2      // Liefert eine Reihung der Laenge 26 in der steht, wie oft die
3      // Buchstaben 'A' bis 'Z' in s vorkommen. Ist erg die Ergebnisreihung,
4      // so gilt:
5      // erg[ 0] enthaelt die Anzahl der 'A'
6      // erg[ 1] enthaelt die Anzahl der 'B'
7      // ...
8      // erg[25] enthaelt die Anzahl der 'Z'
9      //
10     // Beispiel:
11     // int[] er01 = histoBuchstaben("ABC123ABC456AAB");
12     // Die ersten drei Komponenten von er01 haben die Werte 4, 3, 2,
13     // die uebrigen Komponenten haben den Wert 0.
14     // ...
15 } // histoBuchstaben

```

**Tipp:** Wenn eine char-Variable c garantiert einen Buchstaben zwischen 'A' und 'Z' enthält, dann hat der Ausdruck `c - 'A'` einen Wert zwischen 0 und 25.

1. Schreiben Sie eine Methode, die der folgenden Spezifikation entspricht:

```

1  static int[] histo(String z, String s) {
2      // Liefert eine Reihung der Laenge z.length() in der steht,
3      // wie oft die Zeichen z[i] in s vorkommen. Ist erg die Ergebnis-
4      // reihung, so gilt:
5      //
6      // Wie oft das Zeichen z[0] in s vorkommt steht in erg[0]
7      // Wie oft das Zeichen z[1] in s vorkommt steht in erg[1]
8      // Wie oft das Zeichen z[2] in s vorkommt steht in erg[2]
9      // ...
10     // Wie oft das Zeichen z[z.length-1] in s vorkommt
11     // steht in          erg[z.length-1]
12     //
13     // Beispiel:
14     // int[] er01 = histo("?A9", "ABC9??!!78999");
15     // Dann ist er01 gleich {2, 1, 4}.
16     // ...
17 } // histo

```

**Tipp:** Die Klasse String enthält eine Objektmethode mit dem Profil `int indexOf(char c)`.