# Aufgabensammlung für Arbeitsblätter zur Vorlesung „Einführung in die Informatik mit Java“

Markus Richter, Markus Feist, Florian Keller, Fabian Bülow

7. Mai 2018

# Inhaltsverzeichnis

# Teil I

# Aufgaben für Arbeitsblätter

# Kapitel 1

# Grundlagen

## Aufgabe 1
## Programmentwicklung

| *Bezeichner:* | grundl-java |
|---|---|

### *Aufgabentext*

(a) Wozu dient der Java-Compiler, wozu der Java-Interpreter?

(b) Erläutern Sie die Aussage „Java ist plattformunabhängig.“

### *Lösung*

(a) Der Java-Compiler überprüft einen Java-Quelltext auf syntaktische Korrektheit und meldet ggf. Syntaxfehler. Ist der Quelltext korrekt, so übersetzt der Java-Compiler diesen in den sogenannten Java-Bytecode. Der Java-Interpreter liest diesen Bytecode und führt ihn aus.

(b) Die Programmiersprache Java ist in zweierlei Hinsicht plattformunabhängig: auf Quelltextebene und auf Bytecode-Ebene. Ersteres bedeutet, dass ein Java-Quelltext auf jedem Rechner, unabhängig von dessen Prozessorarchitektur (IBM, Mac, etc.) und dessen Betriebssystem (MS Windows, Linux, Mac OS, etc.), in gleicher Weise übersetzt wird, ohne dass entsprechende Anpassungen vorgenommen werden müssen. Plattformunabhängigkeit auf Bytecode-Ebene bedeutet, dass der Java-Bytecode auf jedem Rechner, unabhängig von dessen Prozessorarchitektur und Betriebssystem, in gleicher Weise ausgeführt wird. Einschränkend muss gesagt werden, dass diese Aussagen nur dann richtig sind, wenn auf den verschiedenen Rechnern dieselbe Version des Java-Compilers und des Java-Interpreters installiert sind.

# Aufgabe 2
# Literale

| *Bezeichner:* | grundl-literale |
|---|---|

### *Aufgabentext*

Nachfolgend finden Sie Beispiele von Java-*Literalen*, d.h. von Zeichenfolgen die in einem Java-Quelltext konstante Werte repräsentieren. Geben Sie jeweils den Datentyp des Literals an. Mögliche Datentypen sind **`int`**, **`long`**, **`double`**, **`float`**, **`bool`**, **`char`** und `String`.

(a) `"Hello!"`

(b) `-156`

(c) `1E6`

(d) `""`

(e) `0.5`

(f) `'0'`

(g) `1.5e-1`

(h) `124f`

(i) `0341`

(j) `"13"`

(k) **`true`**

(l) `'a'`

(m) `0xABC`

(n) `.2`

(o) `0xCEL`

### *Lösung*

(a) `String`
(Hello!)

(b) **`int`**
($-156$)

(c) **`double`**
($1000000$)

(d) `String`
(leere Zeichenkette)

(e) **`double`**
($0{,}5$)

(f) **`char`**
(Das Zeichen „0“)

(g) **`double`**
($0{,}15$)

(h) **`float`**
($124$)

(i) **`int`**
($341_8 = 225$)

(j) `String`
(die Zeichenkette „13“)

(k) **`bool`**
(**`true`**)

(l) **`char`**
(a)

(m) **`int`**
($\mathrm{ABC}_{16} = 2748$)

(n) **`double`**
($0{,}2$)

(o) **`long`**
($\mathrm{CE}_{16} = 206$)

# Aufgabe 3
# Ausdrücke

| *Bezeichner:* | grundl-ausdruecke |
|---|---|

### *Aufgabentext*

Nachfolgend finden Sie Beispiele von Java-Ausdrücken. Geben Sie jeweils das Ergebnis des Ausdrucks sowie den Datentyp des Ergebnisses an. Mögliche Datentypen sind **`int`**, **`double`**, **`bool`**, **`char`** und `String`.

(a) `12.3 + 1`

(b) `0.5 * 4`

(c) `2.0 / 4.0`

(d) `(2-1.0)/5`

(e) `14.0 % 5`

(f) `3 / 6.0`

(g) `(int) 1.23`

(h) `2 / (int) 3.14`

(i) `1.0 / 4`

(j) `15 % 2`

(k) `1>2`

(l) `'a'+ 1`

(m) `1 / 2`

(n) `1 == 2`

(o) `(1+1)>1`

### *Lösung*

(a) $13{,}3$
**`double`**

(b) $2$
**`double`**

(c) $0{,}5$
**`double`**

(d) $0{,}2$
**`double`**

(e) $4$ ($14 = 2 \cdot 5 + \mathbf{4}$)
**`double`**

(f) $0{,}5$
**`double`**

(g) $1$ (ganzzahliger Anteil)
**`int`**

(h) $0$
**`int`**

(i) $0{,}25$
**`double`**

(j) $1$ ($15 = 7 \cdot 2 + \mathbf{1}$)
**`int`**

(k) **`false`**
**`bool`**

(l) $98$ (ANSI: ‘a’ = $97$)
**`int`**

(m) $0$ (Ganzzahldivision)
**`int`**

(n) **`false`**
**`bool`**

(o) **`true`**
**`bool`**

# Aufgabe 4
# Boolsche Algebra

| *Bezeichner:* | grundl-boolalg |
|---|---|

### *Aufgabentext*

Unter einer *Boolschen Algebra* versteht man eine Menge, die aus den beiden Elementen $0$ (oder „falsch“) und $1$ (oder „wahr“) besteht, und auf der die binären Verknüpfungen $\wedge$ („und“) und $\vee$ („oder“) sowie die unäre Verknüpfung $\neg$ („nicht“) definiert ist. Es gilt

$$\begin{array}{lllll} 0 \wedge 0 = 0, & 1 \wedge 0 = 0, & 0 \vee 0 = 0, & 1 \vee 0 = 1, & \neg 0 = 1, \\ 0 \wedge 1 = 0, & 1 \wedge 1 = 1, & 0 \vee 1 = 1, & 1 \vee 1 = 1, & \neg 1 = 0. \end{array}$$

In Java repräsentiert der Basistyp **`bool`** eine Boolsche Algebra. Die Symbole $0, 1, \wedge, \vee$ und $\neg$ sind in Java als **`false`**, **`true`**, `&&`, `||` und `!` definiert. Geben Sie das Ergebnis der nachfolgenden Java-Ausdrücke an

(a) **`true`** `&&` **`false`**

(b) **`true`** `||` **`false`**

(c) `(0 == 1) || (1 < 2)`

(d) `(0 != 1) && !(2 < 1)`

(e) `!!!`**`true`**

(f) `(5 == 1) ||` **`false`**

(g) `(a == 0) && (a != 0)`

(h) `(a == 0) || !(a == 0)`

### *Lösung*

(a) **`false`** ($1 \wedge 0 = 0$)

(b) **`true`** ($0 \vee 1 = 1$)

(c) **`true`** ($0 \vee 1 = 1$)

(d) **`true`** ($1 \wedge \neg 0 = 1 \wedge 1 = 1$)

(e) **`false`** ($\neg\neg\neg 1 = \neg\neg 0 = \neg 1 = 0$)

(f) **`false`** ($0 \vee 0 = 0$)

(g) **`false`**

(h) **`true`**

Das Ergebnis in den Aufgabenteilen (g) und (h) ist unabhängig vom Wert, den die Variable `a` tatsächlich besitzt. Dies macht man sich klar, indem man die folgenden beiden Fälle unterscheidet. Erster Fall: Die Variable `a` besitzt den Wert Null. In diesem Fall ist das Ergebnis des Ausdrucks `(a == 0)` **`true`** und das des Ausdrucks `(a != 0)` **`false`**. Zweiter Fall: Die Variable `a` besitzt einen Wert ungleich Null. In diesem Fall ist das Ergebnis des Ausdrucks `(a == 0)` **`false`** und das des Ausdrucks `(a != 0)` **`true`**. Es werden also in beiden Fällen die Wahrheitswerte **`true`** und **`false`** miteinander Verknüpft. Im Aufgabenteil (g) ist daher $1 \wedge 0 = 0 \wedge 1 = 0$, im Aufgabenteil (b) ist $1 \vee 0 = 0 \vee 1 = 1$.

# Kapitel 2

# Zahldarstellung

## Aufgabe 5
## Stellenwertsysteme

| *Bezeichner:* | zahldarst-stellenwertsystem |
|---|---|

***Aufgabentext***

Bei *Stellenwertsystemen* wird eine Zahl durch eine Folge $z_n z_{n-1} \cdots z_1 z_0$ von *Ziffern* $z_k$ dargestellt. Jedes Stellenwertsystem bezieht sich dabei auf eine bestimmte *Basis* $b$, die man bei Bedarf als Subskript an das Ende der Ziffernfolge schreibt. Für die Ziffern gilt $z_k \in \{0, 1, \ldots, b-1\}$. Eine Ziffernfolge $z_n z_{n-1} \cdots z_1 z_0$ im Stellenwertsystem zur Basis $b$ stellt die Zahl

$$z_n \cdot b^n + z_{n-1} \cdot b^{n-1} + \cdots + z_1 \cdot b^1 + z_0 \cdot b^0$$

dar. Wichtige Stellenwertsysteme sind das *Dezimalsystem* ($b = 10$), das *Binärsystem* ($b = 2$), das *Oktalsystem* ($b = 8$) und das *Hexadezimalsystem* ($b = 16$). Geben Sie die Darstellung der folgenden Binär-, Oktal-, Hexadezimal- und Dezimalzahlen in den jeweils anderen Stellenwertsystemen an. Im Hexadezimalsystem bezeichnen die Buchstaben $\mathrm{A}, \mathrm{B}, \ldots, \mathrm{F}$ die Ziffern $10, 11, \ldots, 15$.

(a) $10_2$
(b) $1100_2$
(c) $10101_2$
(d) $101010_2$
(e) $27_8$
(f) $133_8$
(g) $10_8$
(h) $77_8$
(i) $2\mathrm{A}_{16}$
(j) $10_{16}$
(k) $\mathrm{FF}_{16}$
(l) $\mathrm{D1}_{16}$
(m) $17_{10}$
(n) $33_{10}$
(o) $65_{10}$
(p) $72_{10}$

***Lösung***

(a) $10_2 = 2_8 = 2_{10} = 2_{16}$

(b) $1100_2 = 14_8 = 12_{10} = \mathrm{C}_{16}$

(c) $10101_2 = 25_8 = 21_{10} = 15_{16}$

(d) $101010_2 = 52_8 = 42_{10} = 2\mathrm{A}_{16}$)

(e) $27_8 = 10111_2 = 23_{10} = 17_{16}$

(f) $133_8 = 1011011_2 = 91_{10} = 5\mathrm{B}_{16}$

(g) $10_8 = 1000_2 = 8_{10} = 8_{16}$

(h) $77_8 = 111111_2 = 63_{10} = 3\mathrm{F}_{16}$

(i) $2\mathrm{A}_{16} = 101010_2 = 52_8 = 42_{10}$

(j) $10_{16} = 10000_2 = 20_8 = 16_{10}$

(k) $\mathrm{FF}_{16} = 11111111_2 = 377_8 = 255_{10}$

(l) $\mathrm{D1}_{16} = 11010001_2 = 321_8 = 209_{10}$

(m) $17_{10} = 10001_2 = 21_8 = 11_{16}$

(n) $33_{10} = 100001_2 = 41_8 = 21_{16}$

(o) $65_{10} = 1000001_2 = 101_8 = 41_{16}$

(p) $72_{10} = 1001000_2 = 110_8 = 48_{16}$

# Aufgabe 6
# Stellenwertsysteme

| *Bezeichner:* | zahldarst-stellenwert |
|---|---|

***Aufgabentext***

Geben Sie die folgenden Binär-, Oktal-, Dezimal- und Hexadezimalzahlen in den jeweils anderen Stellenwertsystemen an. Im Hexadezimalsystem bezeichnen die Buchstaben $\mathrm{A}, \mathrm{B}, \ldots, \mathrm{F}$ die Ziffern $10, 11, \ldots, 15$.

(a) $10_2$
(b) $1100_2$
(c) $10101_2$
(d) $101010_2$
(e) $27_8$
(f) $133_8$
(g) $10_8$
(h) $77_8$
(i) $2\mathrm{A}_{16}$
(j) $10_{16}$
(k) $\mathrm{FF}_{16}$
(l) $\mathrm{D1}_{16}$
(m) $17_{10}$
(n) $33_{10}$
(o) $65_{10}$
(p) $72_{10}$

***Lösung***

(a) $10_2 = 2_8 = 2_{10} = 2_{16}$
(b) $1100_2 = 14_8 = 12_{10} = \mathrm{C}_{16}$
(c) $10101_2 = 25_8 = 21_{10} = 15_{16}$
(d) $101010_2 = 52_8 = 42_{10} = 2\mathrm{A}_{16}$
(e) $27_8 = 10111_2 = 23_{10} = 17_{16}$
(f) $133_8 = 1011011_2 = 91_{10} = 5\mathrm{B}_{16}$
(g) $10_8 = 1000_2 = 8_{10} = 8_{16}$
(h) $77_8 = 111111_2 = 63_{10} = 3\mathrm{F}_{16}$
(i) $2\mathrm{A}_{16} = 101010_2 = 52_8 = 42_{10}$
(j) $10_{16} = 10000_2 = 20_8 = 16_{10}$
(k) $\mathrm{FF}_{16} = 11111111_2 = 377_8 = 255_{10}$
(l) $\mathrm{D1}_{16} = 11010001_2 = 321_8 = 209_{10}$
(m) $17_{10} = 10001_2 = 21_8 = 11_{16}$
(n) $33_{10} = 100001_2 = 41_8 = 21_{16}$
(o) $65_{10} = 1000001_2 = 101_8 = 41_{16}$
(p) $72_{10} = 1001000_2 = 110_8 = 48_{16}$

# Aufgabe 7
# Ganzzahl-Datentypen

| *Bezeichner:* | zahldarst-ganzzahl |
|---|---|

### *Aufgabentext*

In Java besitzen die Ganzzahl-Datentypen die folgenden Größen

| Datentyp | Größe |
|---|---|
| `byte` | 8 Bit |
| **`short`** | 16 Bit |
| **`int`** | 32 Bit |
| **`long`** | 64 Bit |

Das führende Bit jedes Datentyps gibt das Vorzeichen der Zahl an: Ein 0-Bit zeigt eine nichtnegative Zahl an, ein 1-Bit eine negative Zahl. Jedem Bitmuster entspricht genau eine ganze Zahl. Ist das führende Bit ein 0-Bit (nichtnegative Zahl), repräsentieren die übrigen Bits die Ziffern der Binärdarstellung dieser Zahl.

- Wie viele verschiedene ganze Zahlen können mit den Datentypen `byte`, **`short`**, **`int`** und **`long`** dargestellt werden?
- Welches sind jeweils die größten positiven, ganzen Zahlen, die mit den Datentypen `byte`, **`short`**, **`int`** und **`long`** dargestellt werden können?
- Betrachten Sie den folgenden Quelltextausschnitt:

  ```
  byte a = 100;
  a += 100;
  ```

  Stellt der Wert der Variable `a` nach Ausführung dieser Zeilen die Zahl $200$ dar? Begründen Sie Ihre Antwort.

### *Lösung*

- jedes Bitmuster eines Datentyps entspricht genau einer ganzen Zahl. Jedes Bit kann genau zwei Werte annehmen. Daher können mit den Datentypen `byte`, **`short`**, **`int`** und **`long`** jeweils $2^8$, $2^{16}$, $2^{32}$, bzw. $2^{64}$ verschiedene ganze Zahlen dargestellt werden.
- Der Datentyp `byte` mit 8 Bit kann maximal siebenstellige, positive Binärzahlen darstellen, das führende Bit bei solchen Zahlen ein 0-Bit sein muss. Die größte positive, ganze Zahl lautet daher $1111111_2 = 127 = 128 - 1 = 2^7 - 1$. Für die Datentypen **`short`**, **`int`** und **`long`** erhält man in gleicher Weise $2^{15} - 1$, $2^{31} - 1$ bzw. $2^{63} - 1$ als größte darstellbare positive, ganze Zahl.

- Nein, der Wert der Variable a stellt nach Ausführung der Quelltextzeilen die Zahl $200$ nicht dar. Der Datentyp der Variable ist `byte`. Die größte positive, ganze Zahl, die von diesem Datentyp dargestellt werden kann, ist $2^7 - 1 = 127$. Die Zahl $200$ ist somit nicht darstellbar. Man kann die Antwort noch weiter erläutern: Die Binärdarstellung der Zahl $100 = 64+32+4$ lautet $1100100_2$. Der Datentyp `byte` stellt sie als das Bitmuster $01100100$ dar. Das führende 0-Bit zeigt an, dass die Zahl nichtnegativ ist. Addiert man nun $100$ zu der Zahl hinzu, so erhält man $200 = 2 \cdot 100 = 128 + 64 + 8$, in Binärdarstellung $11001000_2$. Dem entspricht das Bitmuster $11001000$. Da das führende Bit ein 1-Bit ist, stellt dieses Bitmuster im Datentyp `byte` eine negative Zahl dar.

# Aufgabe 8
# Binärbrüche

| *Bezeichner:* | zahldarst-binaerbruch |
|---|---|

***Aufgabentext***

Positive Bruchzahlen, deren Beträge kleiner Eins sind, werden im Binärsystem durch eine Folge $0{,}z_{-1}z_{-2}z_{-3}\cdots$ von Ziffern $z_{-k} \in \{0,1\}$ dargestellt. Um zu verdeutlichen, dass es sich hierbei um eine Darstellung im Binärsystem handelt, kann die Basis $2$ als Subskript an die Ziffernfolge angehängt werden. Eine $0{,}z_{-1}z_{-2}z_{-3}\cdots$ stellt im Binärsystem die Bruchzahl

$$z_{-1}\cdot 2^{-1} + z_{-2}\cdot 2^{-2} + z_{-3}\cdot 2^{-3} + \cdots$$

dar. Geben Sie die folgenden Binärbrüche als Dezimalbrüche an.

(a) $0{,}1_2$

(b) $0{,}01_2$

(c) $0{,}001_2$

(d) $0{,}0001_2$

(e) $0{,}11_2$

(f) $0{,}101_2$

(g) $0{,}111_2$

(h) $0{,}1001_2$

(i) $0{,}0011_2$

(j) $0{,}1011_2$

(k) $1{,}1_2$

(l) $10{,}1_2$

(m) $11{,}11_2$

(n) $1{,}01_2$

(o) $100{,}001_2$

***Lösung***

(a) $0{,}1_2 = 0{,}5 = \frac{1}{2}$
$(1\cdot 2^{-1})$

(b) $0{,}01_2 = 0{,}25 = \frac{1}{4}$
$(0\cdot 2^{-1} + 1\cdot 2^{-2})$

(c) $0{,}001_2 = 0{,}125 = \frac{1}{8}$
$(2^{-3})$

(d) $0{,}0001_2 = 0{,}0625 = \frac{1}{16}$
$(2^{-4})$

(e) $0{,}11_2 = 0{,}75$
$(2^{-1} + 2^{-2})$

(f) $0{,}101_2 = 0{,}625$
$(2^{-1} + 2^{-3})$

(g) $0{,}111_2 = 0{,}875$
$(2^{-1} + 2^{-2} + 2^{-3})$

(h) $0{,}1001_2 = 0{,}5625$
$(2^{-1} + 2^{-4})$

(i) $0{,}0011_2 = 0{,}1875$
$(2^{-3} + 2^{-4})$

(j) $0{,}1011_2 = 0{,}6875$
$(2^{-1} + 2^{-3} + 2^{-4})$

(k) $1{,}1_2 = 1{,}5$
$(2^{0} + 2^{-1})$

(l) $10{,}1_2 = 2{,}5$
$(2^{1} + 2^{-1})$

(m) $11{,}11_2 = 3{,}75$
$(2^{1} + 2^{0} + 2^{-1} + 2^{-2})$

(n) $1{,}01_2 = 1{,}25$
$(2^{0} + 2^{-2})$

(o) $100{,}001_2 = 4{,}125$
$(2^{2} + 2^{-3})$

# Aufgabe 9
# Normalisierte Darstellung

| *Bezeichner:* | zahldarst-norm |
|---|---|

***Aufgabentext***

Jeder positive Binärbruch besitzt eine so genannte *normalisierte Darstellung* von der Form

$$1{,}m_{-1}m_{-2}m_{-3}\cdots \cdot 10_2^e,$$

mit *Ziffern* $m_{-k} \in \{0, 1\}$ und einem *Exponenten* $e \in \mathbb{Z}$ zur Basis $10_2 = 2$. Ähnlich wie bei Dezimalbrüchen erhält man die normalisierte Darstellung bei Binärbrüchen durch Verschieben des Kommas und entsprechender Wahl des Exponenten. Als Beispiel betrachten wir den Binärbruch $0{,}0011_2$. Seine normalisierte Darstellung $1{,}1_2 \cdot 10_2^{-3}$ erhält man, indem man das Komma um $3$ Stellen *nach rechts* verschiebt und den Exponenten $-3$ wählt. Die normalisierte Form des Binärbruchs $101{,}1_2$ ist $1{,}011_2 \cdot 10_2^2$. Sie entsteht, indem man das Komma um $2$ Stellen *nach links* verschiebt und den Exponenten $2$ wählt.

Geben Sie die normalisierte Darstellung der nachfolgenden Binärbrüche an.

(a) $0{,}1_2$
(b) $0{,}01_2$
(c) $0{,}001_2$
(d) $0{,}0001_2$
(e) $10{,}0_2$
(f) $100{,}0_2$
(g) $1000{,}0_2$
(h) $0{,}00101_2$
(i) $10{,}01_2$
(j) $111{,}11_2$
(k) $0{,}011_2$
(l) $101{,}0_2$
(m) $1{,}011_2$
(n) $1001_2$
(o) $1_2$

***Lösung***

(a) $0{,}1_2 = 1{,}0_2 \cdot 10_2^{-1}$
(b) $0{,}01_2 = 1{,}0_2 \cdot 10_2^{-2}$
(c) $0{,}001_2 = 1{,}0_2 \cdot 10_2^{-3}$
(d) $0{,}0001_2 = 1{,}0_2 \cdot 10_2^{-4}$
(e) $10{,}0_2 = 1{,}0_2 \cdot 10_2^{1}$
(f) $100{,}0_2 = 1{,}0_2 \cdot 10_2^{2}$
(g) $1000{,}0_2 = 1{,}0_2 \cdot 10_2^{3}$
(h) $0{,}00101_2 = 1{,}01_2 \cdot 10_2^{-3}$
(i) $10{,}01_2 = 1{,}001_2 \cdot 10_2^{1}$
(j) $111{,}11_2 = 1{,}1111_2 \cdot 10_2^{2}$
(k) $0{,}011_2 = 1{,}1_2 \cdot 10_2^{-2}$
(l) $101{,}0_2 = 1{,}01_2 \cdot 10_2^{2}$
(m) $1{,}011_2 = 1{,}011_2 \cdot 10_2^{0}$
(n) $1001_2 = 1{,}001_2 \cdot 10_2^{3}$
(o) $1_2 = 1{,}0_2 \cdot 10_2^{0}$

# Aufgabe 10
# Dezimal- und Binärbrüche

| *Bezeichner:* | zahldarst-dezbinaerbruch |
|---|---|

***Aufgabentext***

Geben Sie die nachfolgenden Dezimalbrüche als Binärbrüche in normalisierter Darstellung an.

(a) $0{,}5$

(b) $0{,}25$

(c) $0{,}125$

(d) $0{,}0625$

(e) $0{,}75$

(f) $1{,}5$

(g) $10{,}0$

(h) $7{,}5$

(i) $0{,}625$

(j) $8{,}0$

(k) $2{,}5$

(l) $1{,}25$

(m) $4{,}75$

(n) $1{,}125$

(o) $0{,}375$

***Lösung***

(a) $0{,}5 = 2^{-1}$
$= 0{,}1_2 = 1{,}0_2 \cdot 10_2^{-1}$

(b) $0{,}25 = 2^{-2}$
$= 0{,}01_2 = 1{,}0_2 \cdot 10_2^{-2}$

(c) $0{,}125 = 2^{-3}$
$= 0{,}001_2 = 1{,}0_2 \cdot 10_2^{-3}$

(d) $0{,}0625 = 2^{-4}$
$= 0{,}0001_2 = 1{,}0_2 \cdot 10_2^{-4}$

(e) $0{,}75 = 2^{-1} + 2^{-2}$
$= 0{,}11_2 = 1{,}1_2 \cdot 10_2^{-1}$

(f) $1{,}5 = 2^0 + 2^{-1}$
$= 1{,}1_2 = 1{,}1_2 \cdot 10_2^0$

(g) $10{,}0 = 2^3 + 2^1$
$= 1010{,}0_2 = 1{,}01_2 \cdot 10_2^3$

(h) $7{,}5 = 2^2 + 2^1 + 2^0 + 2^{-1}$
$= 111{,}1_2 = 1{,}111_2 \cdot 10_2^2$

(i) $0{,}625 = 2^{-1} + 2^{-3}$
$= 0{,}101_2 = 1{,}01_2 \cdot 10_2^{-1}$

(j) $8{,}0 = 2^3$
$= 1000{,}0_2 = 1{,}0 \cdot 10_2^3$

(k) $2{,}5 = 2^1 + 2^{-1}$
$= 10{,}1_2 = 1{,}01 \cdot 10_2^1$

(l) $1{,}25 = 2^0 + 2^{-2}$
$= 1{,}01_2 = 1{,}01 \cdot 10_2^0$

(m) $4{,}75 = 2^2 + 2^{-1} + 2^{-2}$
$= 100{,}11_2 = 1{,}0011_2 \cdot 10_2^2$

(n) $1{,}125 = 2^0 + 2^{-3}$
$= 1{,}001_2 = 1{,}001_2 \cdot 10_2^0$

(o) $0{,}375 = 2^{-2} + 2^{-3}$
$= 0{,}011_2 = 1{,}1_2 \cdot 10_2^{-2}$

# Aufgabe 11
# Gleitkomma-Datentypen

| *Bezeichner:* | zahldarst-gleitkomma |
|---|---|

***Aufgabentext***

In Java stellen die Gleitkomma-Datentypen Bruchzahlen dar, indem sie die Nachkommastellen und den Exponenten der normalisierten Binärbruchdarstellung abspeichern. Für die Speicherung der Nachkommastellen steht folgender Speicherplatz bereit.

| Datentyp | Nachkommastellen |
|---|---|
| `float` | 23 Bit |
| `double` | 52 Bit |

- Welches ist die nächstgrößere Bruchzahl oberhalb der Eins, die vom Datentyp **`float`** bzw. vom Datentyp **`double`** exakt dargestellt werden kann?
- Was versteht man im Zusammenhang mit Gleitkomma-Datentypen unter Rundungsfehlern? Wodurch entstehen sie?
- Welche Maschinengenauigkeit erzielt man mit dem Datentyp **`float`**, welche mit dem Datentyp **`double`**?
- Was versteht man unter Absorption im Zusammenhang mit Gleitkomma-Arithmetik?
- Was versteht man unter Auslöschung im Zusammenhang mit Gleitkomma-Arithmetik?

***Lösung***

- Die nächstgrößere, exakt darstellbare Bruchzahl oberhalb der Eins ist $1 + 2^{-23}$ für den Datentyp **`float`** und $1 + 2^{-52}$ für den Datentyp **`double`**.
- Unter dem Begriff Rundungsfehler versteht man Fehler, die auftreten, wenn eine Bruchzahl vom jeweiligen Datentyp nicht exakt dargestellt wird. Rundungsfehler entstehen dadurch, dass jeder Datentyp nur eine begrenzte Anzahl von Nachkommastellen der normalisierten Binärbruchdarstellung speichern kann.
- Die Maschinengenauigkeit des Datentyps **`float`** beträgt $2^{-23} \approx 10^{-7}$, die des Datentyps **`double`** beträgt $2^{-52} \approx 2 \cdot 10^{-16}$.
- Unter Absorption versteht man die Entstehung von Rundungsfehlern bei der Addition zweier Gleitkommazahlen, deren Beträge unterschiedliche Größenordnungen haben.

  *Beipiel*: Mit dem Datentyp **`float`** können ca. $8$ Dezimalstellen dargestellt werden. Die Addition von $1{,}0000001 \cdot 10^7$ ($8$ Stellen) und $1 \cdot 10^{-1}$ ($1$ Stelle) ergibt $1{,}00000011 \cdot 10^7$ ($9$ Stellen). Dieses Ergebnis kann mit dem Datentyp **`float`** nicht exakt dargestellt werden und wird auf $1{,}0000001 \cdot 10^7$ gerundet.

- Unter Auslöschung versteht man die Entstehung von Rundungsfehlern bei der Subtraktion zweier fast gleicher Gleitkommazahlen.

  *Beispiel*: Subtrahiert man von der Zahl $1{,}0000011$ die Zahl $1{,}0000010$, so erwartet man das Ergebnis $1 \cdot 10^{-7}$. Führt man diese Subtraktion in Gleitkomma-Arithmetik mit dem Datentyp **`float`** durch, so erhält man das Ergebnis $1{,}1920929 \cdot 10^{-7}$. Nur eine einzige Stelle dieses Wertes ist exakt.

# Aufgabe 12
# Rundungsregel nach IEEE*

| *Bezeichner:* | zahldarst-rundung |
|---|---|

### *Aufgabentext*

Jeder Gleitkomma-Datentyp kann nur eine bestimmte Anzahl $N$ von Nachkommastellen eines Binärbruchs (in normalisierter Darstellung) darstellen. Kann ein Binärbruch nicht mit $N$ Nachkommastellen dargestellt werden, wird dieser auf $N$ Nachkommastellen gerundet. Dies geschieht nach der *IEEE-Rundungsregel* (IEEE: Institute of Electrical and Electronics Engineers). Diese ist folgendermaßen definiert:

- Jeder Binärbruch wird auf den nächstgelegenen darstellbaren Binärbruch gerundet („round to nearest“).
- Liegt der Binärbruch genau in der Mitte zwischen zwei darstellbaren Binärbrüchen, so wird er auf denjenigen darstellbaren Binärbruch gerundet, dessen letzte Stelle eine Null ist („round to nearest even“).

Betrachten wir dazu das folgende Beispiel: Der Bruch $1{,}101_2 = 1{,}625$ soll nach der IEEE-Rundungsregel auf zwei darstellbare Nachkommastellen gerundet werden. Der Bruch liegt genau in der Mitte zwischen den darstellbaren Brüchen $1{,}10_2 = 1{,}5$ und $1{,}11_2 = 1{,}75$. Laut IEEE-Rundungsregel wird er auf den darstellbaren Bruch gerundet, dessen letzte Stelle eine Null ist. Dies ist der Bruch $1{,}10_2 = 1{,}5$.

Runden Sie die nachfolgenden Binärbrüche nach der IEEE-Rundungsregel auf vier Nachkommastellen. Ein Querstrich bezeichnet sich periodisch wiederholende Nachkommastellen.

(a) $1{,}00001_2$
(b) $1{,}00011_2$
(c) $1{,}100011_2$
(d) $1{,}01111_2$
(e) $1{,}0100101_2$
(f) $1{,}01101_2$
(g) $1{,}\overline{01}_2$
(h) $1{,}\overline{1100}_2$
(i) $1{,}0\overline{1}_2$

### *Lösung*

(a) $1{,}0000_2$
(b) $1{,}0010_2$
(c) $1{,}1001_2$
(d) $1{,}1000_2$
(e) $1{,}0101_2$
(f) $1{,}0110_2$
(g) $1{,}0101_2$
(h) $1{,}1101_2$
(i) $1{,}1000_2$

# Aufgabe 13
# Bias-Darstellung

| *Bezeichner:* | zahldarst-bias |
|---|---|

***Aufgabentext***

Die Bias-Darstellung ist eine Möglichkeit, ganze Zahlen in einem bestimmten Bereich durch vorzeichenlose Zahlen darzustellen. Dazu definiert man zunächst eine nichtnegative, ganze Zahl $B$ genannt *Bias* (engl. Neigung, Verzerrung). Die *Bias-$B$-Darstellung* einer ganzen Zahl $z \in \mathbb{Z}$ ist dann durch $z + B$ gegeben. Auf diese Weise können ganze Zahlen größer oder gleich $-B$ durch vorzeichenlose Zahlen dargestellt werden.

Geben Sie die Bias-$7$-Darstellung der folgenden Dezimalzahlen in Binärdarstellung an

(a) $3$
(b) $5$
(c) $-1$
(d) $-3$
(e) $8$
(f) $4$
(g) $-5$
(h) $-7$
(i) $0$

***Lösung***

(a) $3 + 7 = 10 = 1010_2$
(b) $5 + 7 = 12 = 1100_2$
(c) $-1 + 7 = 6 = 110_2$
(d) $-3 + 7 = 4 = 100_2$
(e) $8 + 7 = 15 = 1111_2$
(f) $4 + 7 = 11 = 1011_2$
(g) $-5 + 7 = 2 = 10_2$
(h) $-7 + 7 = 0 = 0_2$
(i) $0 + 7 = 7 = 111_2$

# Kapitel 3

# Anweisungen

## Aufgabe 14
## Anweisungen

| *Bezeichner:* | anweis-1 |
|---|---|

***Aufgabentext***

Gegeben sei der folgende Ausschnitt eines Java-Programms. Dabei sei $n$ eine Variable vom Typ **`int`**.

```
if ( n < 3 ) {
    if ( n > 1 ) {
        System.out.println("A");
    } else {
        System.out.println("B");
    }
} else {
    if ( n <= 3 ) {
        System.out.println("C");
    } else {
        System.out.println("D");
    }
}
```

(a) Welcher Buchstabe wird auf dem Bildschirm ausgegeben, falls $n$ den Wert $1$, $2$, $3$ bzw. $4$ besitzt?

(b) Welcher Buchstabe wird auf dem Bildschirm ausgegeben, falls $n$ den Wert $1$, $2$, $3$ bzw. $4$ besitzt und **jeder** Boolsche Ausdruck in dem Programmabschnitt durch seine Negation ersetzt wurde?

***Lösung***

(a) Für den vorliegenden Programmausschnitt erhält man

| n | **Ausgabe** |
|---|---|
| 1 | B |
| 2 | A |
| 3 | C |
| 4 | D |

(b) Ersetzt man die Ausdrücke `(n < 3)`, `(n > 1)` und `(n <= 3)` durch ihre Negationen `(n >= 3)`, `(n <= 1)` und `(n > 3)`, so erhält man

| n | **Ausgabe** |
|---|---|
| 1 | D |
| 2 | D |
| 3 | B |
| 4 | B |

# Aufgabe 15
# Anweisungen

| *Bezeichner:* | anweis-2 |
|---|---|

### *Aufgabentext*

Gegeben sei der folgende Ausschnitt eines Java-Programms. Dabei sei `i` eine Variable vom Typ **`int`**.

```
if ( i < 1 || i > 3 ) {
    System.out.println("A");
} else if ( i == 1 || i == 2  ) {
    System.out.println("B");
} else {
    System.out.println("C");
}
```

(a) Welcher Buchstabe wird auf dem Bildschirm ausgegeben, falls `i` den Wert $1$, $2$, $3$ bzw. $4$ besitzt?

(b) Realisieren Sie diesen Programmausschnitt mit einer **`switch`**-Anweisung.

### *Lösung*

(a) Für den vorliegenden Programmausschnitt erhält man

| n | **Ausgabe** |
|---|---|
| 1 | B |
| 2 | B |
| 3 | C |
| 4 | A |

(b) Realisation mit einer **`switch`**-Anweisung:

```
switch ( i ) {
    case 1:
    case 2:
        System.out.println("B");
        break;
    case 3:
        System.out.println("C");
        break;
```

```
        default:
            System.out.println("A");
    }
```

# Aufgabe 16
# Kompilerfehler

| | |
|---|---|
| *Bezeichner:* | anweis-fehler |
| *Beschreibung:* | Programmfehler anhand des Compilers finden |
| *Schwierigkeit:* | einfach |
| *Quelle:* | unbekannt |
| *Verwendung:* | SS17 |

### *Aufgabentext*

Das Java-Program `Fehler.java` wurde kompiliert, wobei einige Fehler und Warnungen auftraten. Finden Sie mit Hilfe der Kompilerausgaben die Fehler und notieren Sie die eine Möglichkeit den Fehler zu verbessern. Jeder entdeckte Fehler soll in allen nachfolgenden Programmzeilen als korrigiert gelten, sodass keine Folgefehler auftreten können.

Fehler.java

```
1  public class Fehler {
2    public static void main (String[] args) {
3      int j;
4      final int c = 0;
5      c++;
6      int[] a;
7      a = {1};
8      System.out.println(j)
9      j = a+4;
10     fo (int i = 0; i < j; i++) {
11       System.out.println(a[i]);
12     }
13     int a = 0;
14   }
15 }
```

Der Kompiler hat folgende Fehler ausgegeben:

```
Fehler.java:7: error: variable j might not have been initialized
System.out.println(j);
^
Fehler.java:5: error: cannot assign a value to final variable c
c++;
^
Fehler.java:8: error: ';' expected
System.out.println(j)
```

```
^

Fehler.java:9: error: bad operand types for binary operator '+'
j = a+1;
^
first type:  int[]
second type: int

Fehler.java:10: error: '.class' expected
fo (int i = 0; i < j; i++) {
^
Fehler.java:10: error: > expected
fo (int i = 0; i < j; i++) {
^
Fehler.java:10: error: not a statement
fo (int i = 0; i < j; i++) {
^
Fehler.java:10: error: ';' expected
fo (int i = 0; i < j; i++) {
^
required: double,double
found: int
reason: actual and formal argument lists differ in length
Fehler.java:14: error: variable a is already defined in method main(String[])
int a = 0;
```

### *Lösung*

fehler-korr.java

```
public class Corr {
  public static void main (String[] args) {
    int j = 1;
    final int c;
    c = 0;
    int a[] = new int[] {1};
    System.out.println(c);
    for (int i = 0; i < j; i++) {
      System.out.println(a[i]);
    }
    int b = 0;
  }
}
```

# Kapitel 4

# Schleifen

## Aufgabe 17
## Schleifen

| *Bezeichner:* | schleifen-1 |
|---|---|

***Aufgabentext***

Gegeben sei der folgende Ausschnitt eines Java-Programms:

```
int n = 0;
for (int i=0; i < 7; i++) {
    if (i == 4) {
        i = i + 1;
        n = n + i;
    } else {
        n += i;
    }
}
System.out.println(n);
```

(a) Welche Zahl wird auf dem Bildschirm ausgegeben?

(b) Realisieren Sie diesen Programmausschnitt mit einer **`while`**-Schleife.

***Lösung***

(a) Für die einzelnen Schleifendurchläufe ergeben sich

| i | n | i < 7 | i == 4 | i = i + 1 | n = n + i | n += i | i++ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | *wahr* | *falsch* | – | – | 0 | 1 |
| 1 | 0 | *wahr* | *falsch* | – | – | 1 | 2 |
| 2 | 1 | *wahr* | *falsch* | – | – | 3 | 3 |
| 3 | 3 | *wahr* | *falsch* | – | – | 6 | 4 |
| 4 | 6 | *wahr* | *wahr* | 5 | 11 | – | 6 |
| 6 | 11 | *wahr* | *falsch* | – | – | 17 | 7 |
| 7 | 17 | *falsch* | – | – | – | – | – |

Es wird also die Zahl 17 auf der Konsole ausgegeben.

(b) Realisation mit einer **`while`**-Schleife:

```
int n;
int i = 0;
while (i < 7) {
    if (i == 4) {
        i = i + 1;
        n = n + i;
    } else {
        n += i;
    }
    i++;
}
System.out.println(n);
```

# Aufgabe 18
# Schleifen

| *Bezeichner:* | schleifen-2 |
|---|---|

### *Aufgabentext*

Gegeben sei der folgende Ausschnitt eines Java-Programms:

```
int k = 0;
while (k < 8) {
    if (k % 4 == 0) {
        System.out.print("+");
    } else {
        System.out.print("//");
    }
    k += 2;
}
```

(a) Was wird auf dem Bildschirm ausgegeben?

(b) Realisieren Sie diesen Programmausschnitt mit einer **`for`**-Schleife.

### *Lösung*

(a) Für die einzelnen Schleifendurchläufe ergeben sich

| `k` | `k < 8` | `k % 4 == 0` | **Ausgabe** | `k += 2` |
|---|---|---|---|---|
| 0 | *wahr* | *wahr* | „+“ | 2 |
| 2 | *wahr* | *falsch* | „//“ | 4 |
| 4 | *wahr* | *wahr* | „+“ | 6 |
| 6 | *wahr* | *wahr* | „//“ | 8 |
| 8 | *falsch* | – | – | – |

Es wird also die Zeichenkette „+//+//“ auf der Konsole ausgegeben.

(b) Realisation mit einer **`for`**-Schleife:

```
for (int k = 0; k < 8; k += 2) {
    if (k % 4 == 0) {
        System.out.print("+");
    } else {
        System.out.print("//");
    }
}
```

# Kapitel 5

# Felder

## Aufgabe 19
## Felder

| *Bezeichner:* | felder-felder |
|---|---|

***Aufgabentext***

Betrachten Sie die folgenden Ausschnitte eines Java-Programms:

(a)
```
int[] feld = {1, 2, 3, 4, 5};
for (int i = 1; i < feld.length; i++) {
     feld[i] += feld[i-1];
}
```

(b)
```
int[] feld = {1, 4, 9, 16, 25};
for (int i = feld.length-1; i > 0; i--) {
    feld[i] -= feld[i-1];
}
```

(c)
```
int[] feld = {5, 4, 3, 2, 1};
for (int i = 0; i < feld.length; i++) {
    feld[i] = feld[feld.length-i-1];
}
```

(d)
```
int[] feld = {1, 0, 4, 2, 3};
for (int i = 0; i < feld.length; i++) {
    feld[i] = feld[feld[i]];
}
```

(e)
```
int[] feld = {1, 2, 3, 4, 5};
for (int i = 0; i < feld.length; i++) {
    feld[i] *= (i+1);
}
```

(f)
```
int[] feld = {5, 4, 3, 2, 1};
for (int i = feld.length-1; i >= 0; i--) {
    int k = feld[i];
    feld[i] = feld[feld.length-i-1];
    feld[feld.length-i-1] = k;
}
```

Was wird jeweils auf der Konsole ausgegeben, wenn unmittelbar nach jedem Programmausschnitt die folgenden Zeilen ausgeführt werden?

```
for (int i = 0; i < feld.length; i++) {
    System.out.print(feld[i] + ", ");
}
```

***Lösung***

(a) Es wird „1, 3, 6, 10, 15, “ auf der Konsole ausgegeben.

(b) Es wird „1, 3, 5, 7, 9, “ auf der Konsole ausgegeben.

(c) Es wird „1, 2, 3, 2, 1, “ auf der Konsole ausgegeben.

(d) Es wird „0, 0, 3, 3, 3, “ auf der Konsole ausgegeben.

(e) Es wird „1, 4, 9, 16, 25, “ auf der Konsole ausgegeben.

(f) Es wird „5, 4, 3, 2, 1, “ auf der Konsole ausgegeben.

# Aufgabe 20
# Referenzen

| *Bezeichner:* | felder-referenzen |
|---|---|

### *Aufgabentext*

Betrachten Sie die folgende `main`-Methode eines Java-Programms.

```
public static void main(String[] args) {
    int[] feld1 = {1};
    int[] feld2 = methode(feld1);
    System.out.print(feld1[0] + ", " + feld2[0]);
}
```

Die Klassenmethode `methode` wird im folgenden auf drei verschiedene Weisen definiert:

(a)
```
static int[] methode(int[] feld) {
    feld[0] = feld[0] + 1;
    return feld;
}
```

(b)
```
static int[] methode(int[] feld) {
    int[] hilfsfeld = feld;
    hilfsfeld[0] = feld[0] + 1;
    return hilfsfeld;
}
```

(c)
```
static int[] methode(int[] feld) {
    int[] hilfsfeld = new int [1];
    hilfsfeld[0] = feld[0] + 1;
    return hilfsfeld;
}
```

Geben Sie für jede Methodendefinition an, was beim Ausführen des Programms auf der Konsole ausgegeben wird.

### *Lösung*

(a) Es wird „2, 2“ auf der Konsole ausgegeben.

(b) Es wird „2, 2“ auf der Konsole ausgegeben.

(c) Es wird „1, 2“ auf der Konsole ausgegeben.

# Aufgabe 21
# Seiteneffekte

| *Bezeichner:* | felder-sideEff |
|---|---|

***Aufgabentext***

Betrachten Sie das folgende Java-Programm mit den Klassenmethoden `computeNorm` die sich sowohl im Datentype des Übergabeparameters und des Rückgabeparameters unterscheiden.

FelderFunktionen.java

```
1  // Author: A. Mink, Nov.2017, side effects
2  public class FelderFunktionen{
3    static int computeNorm(int n) {
4      n = n*n;
5      return n;
6    }
7    static double computeNorm(int[] x) {
8      x[0] = x[0]*x[0] + x[1]*x[1];
9      return Math.sqrt(x[0]);
10   }
11   public static void main (String [] args){
12     int a = 2;
13     System.out.println(computeNorm(a));
14     System.out.println(a);
15
16     int[] d = {-1, 1};
17     System.out.println(computeNorm(d));
18     System.out.println(d[0]);    // What happened?
19   }
20 }
```

(a) Die Klassenmethode `computeNorm` mit Rückgabewert **`int`** berechnet das Quadrat des Übergabeparameters vom Datentyp **`int`**. In der Klassenmethode wird eine Kopie des Übergabeparameters angelegt, damit wird der Wert von `a` nicht verändert. Besser ist schlicht **`return`** `n*n;` auszuführen.

(b) Nun wird ein Feld an die Klassenmethode `computeNorm` übergeben. Nach dem Aufruf der Klassenmethode ist auch der Übergabeparameter verändert. Erklären Sie dieses Verhalten, Stichwort *Seiteneffekt*.

(c) Implementieren Sie eine sichere `computeNorm` die den Übergabeparameter nicht verändert.

***Lösung***

(a) Nichts zu tun.

(b) In Klassenmethode `computeNorm` mit Übergabeparameter **`int`**`[]` wird keine Kopie angelegt. Das erlaubt es wiederrum direkt in das übergebene Feld `d` zuschrieben und den ursprünglichen Wert zu verändern.

(c) Sichere Implementierung ohne Seiteneffekte.

FelderFunktionen.java

```
1  // Author: A. Mink, Nov.2017, side effects
2  public class FelderFunktionenSol{
3    static int computeNorm(int n) {
4      return n*n;
5    }
6    static double computeNorm(int[] x) {
7      return x[0]*x[0] + x[1]*x[1];   // fix side effect
8    }
9    public static void main (String [] args){
10     int a = 2;
11     System.out.println(computeNorm(a));
12     System.out.println(a);
13
14     int[] d = {-1, 1};
15     System.out.println(computeNorm(d));
16     System.out.println(d[0]);
17   }
18 }
```

# Aufgabe 22
# Felder: Grundlagen

| *Bezeichner:* | felder-grundl |
|---|---|

### *Aufgabentext*

Felder können aus den Datentypen **`double`**, **`int`**, **`char`**, `String`, ... bestehen. Der Index aller Felder beginnt mit '0', d.h. ein Feld mit 10 Elementen hat die Indizes von 0 bis 9.

Gegeben sei der folgende Ausschnitt eines `Java`–Programms.

```
int[] feld = new int[10];
for(int i=0; i<feld.length; i++){
  feld[i]=1;
  for(int j=0; j<i; j++){
    feld[i]=feld[i]+feld[j];
  }
}
for(int i=0; i<feld.length; i++){
  System.out.println("feld("+i+") = "+feld[i]);
}
```

Was wird auf dem Bildschirm ausgegeben?

### *Lösung*

Es wird

```
feld(0) = 1
feld(1) = 2
feld(2) = 4
feld(3) = 8
feld(4) = 16
feld(5) = 32
feld(6) = 64
feld(7) = 128
feld(8) = 256
feld(9) = 512
```

ausgegeben.

# Aufgabe 23
# Felder: Binäre Suche

| *Bezeichner:* | felder-binsearch |
|---|---|

***Aufgabentext***

Gegeben sei untenstehende Implementierung der Binären Suche. Das Feld `feld` sei dabei ein aufsteigend geordnetes Feld aus ganzen Zahlen (z.B. 1, 3, 5, 8, 9, 100).

```
import java.util.*;

class binsearch {
  public static void main(String[] args){
    Locale.setDefault(Locale.US);
    Scanner sc = new Scanner(System.in);

    int[] feld = {2, 4, 6, 8, 10, 12, 14};

    System.out.println("Bitte Wert eingeben : ");
    int wert=sc.nextInt();

    int min = 0;
    int max = feld.length-1;

    while(min!=max){
      int mid = (min+max)/2;
      System.out.println(" feld("+mid+") = " + feld[mid]);

      if (wert < feld[mid])
        max = mid-1;
      if (wert > feld[mid])
        min = mid+1;
      if (wert == feld[mid] || min > max)
        min = max = mid;
    }

    if (wert != feld[min])
      System.out.println("Der Wert "+wert+" ist nicht in feld gespeichert ");
    else
      System.out.println("Der Wert "+wert+" ist in feld("+min+") gespeichert ");
  }
}
```

Welche Ausgabe liefert das Programm für die Eingaben 9 bzw. 10 zurück?

***Lösung***

Eingabe 9:
feld(3) = 8
feld(5) = 12
Der Wert 9 ist nicht in feld gespeichert

Eingabe 10:
feld(3) = 8
feld(5) = 12
Der Wert 10 ist in feld(4) gespeichert

# Kapitel 6

# Funktionen

## Aufgabe 24
## Funktionen

| *Bezeichner:* | funktionen-grundl |
|---|---|

***Aufgabentext***

Funktionen stellen gekapselte Programmabschnitte dar in denen seperat Befehle ausgeführt werden. Eine Funktion kann Übergabeparameter und auch Rückgabewerte besitzen, womit sie sich sehr gut für sich wiederholende Programmabläufe eignet.

(a) Erstellen Sie eine Funktion vom Type **`void`**, die `Hello World!` auf der Konsole ausgibt.

(b) Erstellen Sie eine Funktion vom Type **`int`**, die auf eine gegebene Zahl eins addiert und das Ergebnis zurück gibt.

(c) Erstellen Sie eine Funktion vom type **`double`**, die die Sume zweier Gleitkommazahlen zurück gibt.

***Lösung***

Funktionen.java

```
/* Minimal example for functions*/
import java.util.*;
public class Funktionen {
  public static void foo() {
    System.out.println("Hello World!");
  }
  public static int addOne(int a) {
```

```
    return a + 1;
  }
  public static double add(double lhs, double rhs) {
    return lhs + rhs;
  }
  public static void main(String[] args) {
    foo();
    System.out.println("addOne(3): " + addOne(3));
    System.out.println("add(1.1,7): "+add(1.1,7));
  }
}
```

# Aufgabe 25
# Funktionen-Trapezregel

| *Bezeichner:* | funktionen-trapez |
|---|---|

### *Aufgabentext*

Berechnen Sie numerisch das Integral der Funktion $f$ definiert als $f(x) = -x^2 + 4$ in den Grenzen $0$ und $3$.

(a) Verwenden Sie die Trapezregel mit zwei Stützstellen

$$I_1 = \int_a^b f(x)dx \simeq (b-a)\frac{f(a)+f(b)}{2}.$$

Hier soll $a = 0$ und $b = 3$ sein.

(b) Verwenden Sie die zusammengesetzte Trapezregel für $N$ Stützstellen

$$I_2 = \int_a^b f(x)dx \simeq h\left[\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum_{n=1}^{N-1} f\left(a+nh\right)\right],$$

mit Gewicht $h = \frac{b-a}{N}$ und Integrationsgrenzen $a = 0$ und $b = 3$.

**Hinweis:** Implementieren Sie eine Java-Funktion

```
public static function(double x) {
  return -x*x + 4;
}
```

### *Lösung*

Analytische Berechnung von Integral liefert $\int_0^3 -x^2 + 4\ dx = 3$. Beispiel Implementierung.

Trapez.java

```
// compute integral via trapezoidal rule
import java.util.*;
public class Trapez{
  public static double f(double x) {
    return -x*x + 4;
  }
  public static void main(String[] args) {
    double a = 0;
    double b = 3;
```

```
    int N = 6;
    // (a)
    double I1 = (b-a) * (f(a) + f(b)) / 2;
    // (b)
    double sum = 0;
    for (int n = 1; n < N; n++) {
      sum += f(a + n * (b-a)/N);
    }
    double I2 = (b-a)/N * (0.5*f(a) + 0.5*f(b) + sum);
    System.out.println("a: " + I1);
    System.out.println("b: " + I2);
  }
}
```

# Aufgabe 26
# Exponentialreihe

| | |
|---|---|
| *Bezeichner:* | funktionen-ss2017 |
| *Beschreibung:* | Berechnung der Exponentialreihe |

### *Aufgabentext*

Schreiben Sie ein kompilierbares Java-Programm zur Bestimmung der Funktionswerte der Exponentialfunktion $e^x$. Hierzu soll die Reihendarstellung

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \dots$$

verwendet werden. Die Funktionswerte sollen durch endliche Summen

$$e^x \approx S(N) := \sum_{i=0}^{N} \frac{x^i}{i!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \dots + \frac{x^N}{N!}$$

angenähert werden. Die einzelnen Summanden $y_i := x^i/i!$ lassen sich dabei wie folgt berechnen:

$$\begin{aligned} y_0 &:= 1 \\ y_i &:= \frac{x}{i}\, y_{i-1}, \quad i = 1, 2, \dots \ . \end{aligned}$$

Gehen Sie dabei wie folgt vor:

- Schreiben Sie das Hauptprogramm. Lesen Sie dabei erst den Wert $x$ von der Konsole ein und speichern Sie diesen in einem geeigneten Datentyp ab.
- Verwenden Sie zur Summation eine **`do`**–**`while`**-Schleife. Brechen Sie die Summation ab, wenn sich zwei aufeinanderfolgende Summen $S(N)$ und $S(N+1)$ um weniger als $\varepsilon = 10^{-12}$ voneinander unterscheiden.
- Geben Sie den berechneten Wert und die Anzahl der zur Berechnung benötigten Summanden auf dem Bildschirm aus. Vergleichen Sie den genäherten Wert mit dem Wert der Standardfunktion, vgl. `Math.exp()`.

### *Lösung*

```
/* Compute exponential series, author: Albert Mink */
import java.util.Scanner;
public class Expo{
```

```
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.println("Berechne Exponentialfunktion exp(x) fuer x = ");
    double x = in.nextDouble();
    double exp = 1;
    double exp_alt = 0;
    double yi = 1;
    int i = 1;
    do {
      System.out.println(exp);
      exp_alt = exp;
      yi = x/i *yi;
      exp = exp_alt + yi;
      i++;
    } while ( Math.abs(exp-exp_alt) >= 10e-12 );
    System.out.println("exp(x) = " + exp);
    System.out.println("Abbruch nach " +i+ " Schleifendurchlaeufen");
    System.out.println("Differenz " + (Math.exp(x)-exp));
  }
}
```

# Kapitel 7

# Rekursion

## Aufgabe 27
## Rekursion

| *Bezeichner:* | rekursion-1 |
|---|---|

***Aufgabentext***

Nachfolgend ist die Definition einer Klassenmethode gegeben, die zu einer Zahl $n \in \mathbb{N}$ die Fakultät $n! := n(n-1)(n-2)\cdots 1$ rekursiv berechnet.

```
static int fakultaet(int n) {
    if (n <= 1) {
        return 1;
    } else {
        return n * (fakultaet(n-1));
    }
}
```

(a) Wie oft wird die Methode durch den Aufruf `fakultaet(4);` aufgerufen?

(b) Geben Sie eine äquivalente Definition der Methode ohne rekursiven Aufruf an.

***Lösung***

(a) Die Methode wird viermal aufgerufen.

(b) Eine mögliche Klassendefinition ohne rekursiven Aufruf lautet

```
static int fakultaet(int n) {
    if (n <= 1) {
        return 1;
    } else {
        int fak = 1;
        for (int i = 2; i <= n; i++) {
            fak *= i;
        }
        return fak;
    }
}
```

# Aufgabe 28
# Rekursion

| *Bezeichner:* | rekursion-2 |
|---|---|

### *Aufgabentext*

Betrachten Sie die nachfolgende Definition einer Klassenmethode in der Programmiersprache Java.

```
static void schachtelung(double x, double a, double b) {
    if ( b - a <= 0.25 ) {
        System.out.println(a + "; " + b);
    } else {
        double c = 0.5 * (a + b);
        if ( x < c ) {
            schachtelung(x,a,c);
        } else {
            schachtelung(x,c,b);
        }
    }
}
```

(a) Wie oft wird die Methode durch den Aufruf `schachtelung(0.6,0.0,1.0);` aufgerufen? Was wird in diesem Fall auf der Konsole ausgegeben?

(b) Geben Sie eine äquivalente Definition der Methode ohne rekursiven Aufruf an.

### *Lösung*

(a) Die Methode wird dreimal aufgerufen. Es wird „$0{,}5$; $0{,}75$“ auf der Konsole ausgegeben.

(b) Eine mögliche Klassendefinition ohne rekursiven Aufruf lautet

```
static void schachtelung(double x, double a, double b) {
    double c;
    while ( b - a > 0.25  ) {
        c = 0.5 * (a + b);
        if ( x > c ) {
            a = c;
        } else {
            b = c;
        }
```

```
        }
        System.out.println(a + "; " + b);
    }
```

# Aufgabe 29
# Rekursion

| *Bezeichner:* | rekursion-3 |
|---|---|

### *Aufgabentext*

Die Folge der *Fibonacci–Zahlen* $a_0, a_1, a_2, \ldots$ ist folgendermaßen definiert:

$$a_0 := 1, \qquad a_1 := 1, \qquad a_n := a_{n-1} + a_{n-2} \quad \text{für } n = 2, 3, \ldots$$

Die Klassenmethode `fibonacci` berechnet die Fibonacci–Zahlen rekursiv. Sie ist folgendermaßen definiert:

```
static int fibonacci(int n) {
    if ( n <= 1 ) {
        return 1;
    } else {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
```

(a) Geben Sie die ersten zehn Folgenglieder der Folge der Fibonacci–Zahlen an.

(b) Wie oft wird die Methode durch den Aufruf `fibonacci(4)` aufgerufen?

### *Lösung*

(a) Die ersten zehn Folgenglieder lauten: $1$, $1$, $2$, $3$, $5$, $8$, $13$, $21$, $34$, $55$.

(b) Die Methode wird insgesamt neunmal aufgerufen.

# Aufgabe 30
# Funktionen

| *Bezeichner:* | rekursion-4 |
|---|---|

***Aufgabentext***

Es seien die folgenden Java-Funktionen f1 und f2 gegeben:

```
static int f1(int x, int y)
{
   if (y <= 0) return 1;
   if (y % 2 == 0)
      return f1(x*x, y/2);
   else
      return x*f1(x, y-1);
}

static int f2(int x, int y)
{
   if (y <= 0) return 1;
   int result = 1;
   while(...){
     if(y % 2 == 0){
        x *= x;
        y /= 2;
     }
     else{
        result *= x;
        y--;
     }
   }
   return result;
}
```

a) Welche Funktion wird in f1 berechnet?

b) Wieviele Multiplikationen werden für den Aufruf f1(3, 4) benötigt?

c) Die (noch unvollständige) Funktion f2 soll für alle Eingaben exakt das gleiche Ergebnis zurückgeben wie f1. Welche Bedingung ist in der while–Schleife einzusetzen?

***Lösung***

(a) $f(x) = x^y, \quad y \geq 0.$

(b) 3

(c) $while(y >= 1)$

# Kapitel 8

# Komplexität

## Aufgabe 31
## Komplexität

| *Bezeichner:* | komplexitaet-1 |
|---|---|

***Aufgabentext***

Betrachten Sie die nachfolgenden Methodendefinitionen. Jede Methode besitzt einen formalen Parameter vom Typ **`int`**, über den eine natürliche Zahl $n \in \mathbb{N}$ an die Methode übergeben werden kann. Geben Sie jeweils den *asymptotischen Rechenaufwand* (hier: Anzahl der Multiplikationen) der Methoden in Abhängigkeit von $n$ als *Landau-Symbol* an. Mögliche Landau-Symbole sind $\mathcal{O}(1)$, $\mathcal{O}(\log n)$, $\mathcal{O}(n)$, $\mathcal{O}(n \log n)$, $\mathcal{O}(n^2)$, $\mathcal{O}(n^3)$, $\mathcal{O}(b^n)$, $\mathcal{O}(n!)$.

(a)
```
static int methode(int n) {
    int s = 1;
    for (int i=1; i <= n; i++) {
        s *= i;
    }
    return s;
}
```

(b)
```
static int methode(int n) {
    if ( n <= 0 ) {
        return 1;
    } else {
        return methode(n-1) * methode(n-1);
    }
}
```

(c)
```
static int methode(int n) {
    int s = 1;
    for (int i = 0; i < n; i++) {
        for (int j = 1; j <= n; j++) {
            s *= j;
        }
    }
    return s;
}
```

(d)
```
static int methode(int n) {
    if ( n <= 0 ) {
        return 1;
    } else {
        return n * methode(n/2);
    }
}
```

(e)
```
static int methode(int n) {
    if (n <= 0) {
        return 1;
    } else {
        int s = 1;
        for (int i = 0; i < n; i++) {
            s *= methode(n-1);
        }
        return s;
    }
}
```

(f)
```
static int methode(int n) {
    if ( n <= 0 ) {
        return 1;
    } else {
        return n * methode(n-1);
    }
}
```

***Lösung***

(a) $O(n)$ (lineare Komplexität). Die Multiplikationen werden in einer Schleife ausgeführt, die $n$-mal durchlaufen wird. Pro Schleifendurchlauf wird genau eine Multiplikation ausgeführt. Also ergeben sich für $n$ Schleifendurchläufe genau $n$ Multiplikationen.

(b) $O(2^n)$ (exponentielle Komplexität). Die Multiplikationen werden in einer rekursiv definierten Methode ausgeführt. Ein Methodenaufruf für einen Parameter $n > 0$ führt zu einer Multiplikation sowie zu zwei Methodenaufrufen für den Parameter $n - 1$. Für $n \leq 0$ wird keine Multiplikation ausgeführt und keine weitere Methode aufgerufen. Daher ergeben sich insgesamt $2^n$ Methodenaufrufe mit jeweils einer Multiplikation.

(c) $O(n^2)$ (quadratische Komplexität). Die Multiplikationen werden in einer geschachtelten Schleife ausgeführt. Die äußere Schleife wird $n$-mal durchlaufen. Bei jedem Durchlauf der äußeren Schleife wird die innere Schleife $n$-mal Durchlaufen. Bei jedem Durchlauf der inneren Schleife wird genau eine Multiplikation ausgeführt. Also ergeben sich insgesamt $n \cdot n = n^2$ Multiplikationen.

(d) $O(\log n)$ (logarithmische Komplexität). Die Multiplikationen werden in einer rekursiv definierten Methode ausgeführt. Ein Methodenaufruf für einen Parameter $n > 0$ führt zu einer Multiplikation sowie zu einem Methodenaufruf für den Parameter $\lfloor n/2 \rfloor$. Für $n \leq 0$ wird keine Multiplikation ausgeführt und keine weitere Methode aufgerufen. Daher ergeben sich insgesamt $\lfloor \log_2(n) \rfloor + 1$ Methodenaufrufe mit jeweils einer Multiplikation.

(e) $O(n!)$ (faktorielle Komplexität). Die Multiplikationen werden in einer Schleife innerhalb einer rekursiv definierten Methode ausgeführt. Ein Methodenaufruf für einen Parameter $n > 0$ führt zu $n$ Schleifendurchläufen mit jeweils einer Multiplikation sowie zu $n$ Methodenaufrufen für den Parameter $n - 1$. Für $n \leq 0$ wird keine Multiplikation ausgeführt und keine weitere Methode aufgerufen. Daher ergeben sich insgesamt $n \cdot (n-1) \cdots 1 = n!$ Multiplikationen.

(f) $O(n)$ (lineare Komplexität). Die Multiplikationen werden in einer rekursiv definierten Methode ausgeführt. Ein Methodenaufruf mit für einen Parameter $n > 0$ führt zu einer Multiplikation sowie zu einem Methodenaufruf für den Parameter $n - 1$. Für $n = 0$ wird keine Multiplikation ausgeführt und keine weitere Methode aufgerufen. Daher ergeben sich insgesamt $n$ Methodenaufrufe mit jeweils einer Multiplikation.

# Aufgabe 32
# Effizienz

| *Bezeichner:* | komplexitaet-eff |
|---|---|

### *Aufgabentext*

(a) Gegeben ist folgender Programmausschnitt.

1) Beschreiben Sie das Ergebnis der Rechnung.
2) Zählen Sie die zur Berechnung nötige Anzahl an Rechenoperationen. Im weiteren sei `n` eine beliebige natürliche Zahl. Charakterisieren Sie den Rechenaufwand des Ausschnittes mit Hilfe eines Ausdrucks in der Landau-Notation.

Effizienz.java

```
1  // Vergangene Klausuraufgabe
2  import java.util.Arrays;
3  public class Effizienz{
4    public static void main (String[] args) {
5      int n = 2;
6      double[][] a = {{1,2},{3,4}};
7      double[] b = {5,6};
8      double[] c = {7,8};
9      double[] res = new double[n];
10     for (int mu = 0; mu < n; mu++) {
11       double skp = 0.0;
12       for (int nu = 0; nu < n; nu++) {
13         skp += a[mu][nu] * b[nu];
14       }
15       res[mu] = skp + c[mu];
16     }
17     System.out.println(Arrays.toString(res));
18   }
19 }
```

(b) Vereinfachen Sie die folgenden Ausdrücke zur Komplexität durch geeignetes Runden zu jeweils einen Ausdruck in der Landau-Notation, z.B. $\mathcal{O}(n^3 + n) = \mathcal{O}(n^3)$:

1) $\mathcal{O}(10n^3/(2n^2) + n^2)$
2) $\mathcal{O}(n(n^3 + n) - n^2)$
3) $\mathcal{O}(e^{1000} + ln(n))$

4) $\mathcal{O}(-1 + e + (ln(n)) * n + 1/n)$

### *Lösung*

(a) (1)
- Mit Hilfe von einer **`for`**-Schleife werden die Elemente des neuen Vektors `res` berechnet.
- Dabei wird mit der Hilfsvariablen `skp` und einer weiteren **`for`**-Schleife das Skalarprodukt der `mu`-ten Spalte der Matrix `a` und des Vektors `b` berechnet.
- Zum Schluss wird noch das jeweilige Element des Vektors `c` addiert.
- Das enstpricht der Berechnung `d = a * b + c`,

(2)
- Die äußere Schleife wird `n`-mal durchgelaufen, das entspicht $n$-mal die Anzahl der Operationen in der inneren Schleife.
- Die innere Schleife wird auch `n`-mal durchgelaufen. Dort wird eine Addition und eine Multiplikation durchgeführt. Dazu kommt noch eine Addition. Das ergibt $2n + 1$ Operationen.
- Insgesamt sind das wegen der Schachtelung der Schleifen $n(2n + 1) = 2n^2 + n$ Operationen. Das Entspricht einer Komplexität von $\mathcal{O}(n^2)$.

(b)
1) $\mathcal{O}(10n^3/(2n^2) + n^2) = \mathcal{O}(5n + n^2) = \mathcal{O}(n^2)$
2) $\mathcal{O}(n(n^3 + n) - n^2) = \mathcal{O}(n^4 + n^2 - n^2) = \mathcal{O}(n^4)$
3) $\mathcal{O}(e^{1000} + ln(n)) = \mathcal{O}(ln(n))$
4) $\mathcal{O}(-1 + e + (ln(n)) * n + 1/n) = \mathcal{O}((ln(n)) * n + 1/n) = \mathcal{O}((ln(n)) * n)$

# Kapitel 9

# Klassen

## Aufgabe 33
## Artikelverwaltung

| *Bezeichner:* | strukt-artikel |
|---|---|

***Aufgabentext***

Schreiben Sie ein Java-Programm, welches die Verkaufsartikel eines Fachbuchhandlung verwaltet. Gehen Sie dabei folgendermaßen vor:

(a) Definieren Sie eine öffentlichen Klasse namens `Artikel`, und definieren Sie für die Klasse die folgenden Elemente: Eine Instanzvariable `anzahl` vom Typ **`int`** für die Anzahl der vorhandenen Einheiten des Artikels, eine Instanzvariable `preis` vom Typ **`double`** für den Preis einer Einheit und eine Instanzvariable `bezeichnung` vom Typ `String` für die Bezeichnung des Artikels.

(b) Definieren Sie eine öffentliche Klasse namens `Artikelverwaltung`. Definieren für diese Klasse eine Klassenmethode namens `liesArtikel` ohne Parameter, in der die Bezeichnung, die Anzahl der vorhandenen Einheiten und der Preis eines Artikels mit begleitendem Text von der Konsole eingelesen werden und diese in Form einer Instanz der Klasse `Artikel` zurück gegeben werden.

(c) Definieren Sie ferner eine Klassenmethode namens `liesListe` mit einem Parameter vom Typ **`int`** für die Anzahl $n$ der einzulesenden Artikel. Die Methode soll $n$ Artikel mit Hilfe der Methode `liesArtikel` von der Konsole eingelesen. Die eingelesenen Artikel sollen als Feld vom Typ `Artikel` zurück gegeben werden.

(d) Definieren Sie eine Klassenmethode namens `zeigeArtikel` mit einem Parameter vom Typ `Artikel`. Die Methode soll die Bezeichnung des übergebene Artikels, die Anzahl der vorhandenen Einheiten sowie den Preis einer Einheit und den Gesamtwert des Artikels auf der Konsole ausgeben.

(e) Definieren Sie eine Klassenmethode namens `zeigeListe` mit einem Parameter vom Typ `Artikel[]`. Die Methode soll alle Artikel, die im übergebenen Feld gespeichert sind mittels `zeigeArtikel` auf der Konsole ausgeben. Ferner soll der Gesamtwert aller Artikel berechnet und auf der Konsole ausgegeben werden.

(f) Definieren Sie eine `main`-Methode, in der die Anzahl der einzulesenden Artikel und die Artikel selber von der Konsole eingelesen und anschließend als Liste auf der Konsole ausgegeben werden. Testen Sie die Klasse mit folgenden Daten: Es sollen

- 25 Einheiten vom Artikel „Mathematik-Lehrbuch“ (Preis 29,95 Euro),
- 30 Einheiten vom Artikel „Java-Lehrbuch“ (Preis 24,95 Euro),
- sowie 15 Einheiten vom Artikel „Physik-Lehrbuch“ (Preis 34,95 Euro),

vorhanden sein. Der Gesamtwert aller Artikel beträgt für dieses Beispiel 2021,50 Euro.

### *Lösung*

Artikel.java

```
public class Artikel {
  int anzahl;
  double preis;
  String bezeichnung;
}
```

Artikelverwaltung.java

```
import java.util.*;
public class Artikelverwaltung {
  static Scanner sc;

  static Artikel liesArtikel () {
    Artikel Ding = new Artikel();
    System.out.print( "Bitte Artikelname eingeben: " );
    Ding.bezeichnung = sc.next();
    System.out.print( "Bitte vorhandene Einheiten eingeben: " );
    Ding.anzahl = sc.nextInt();
    System.out.print( "Bitte Preis pro Einheit eingeben: " );
    Ding.preis = sc.nextDouble();
    return Ding;
  }

  /*Lese alle Artikel von der Konsole ein.*/
  static Artikel[] liesListe ( int AnzArtikel ) {
    Artikel[] Artikelliste = new Artikel[AnzArtikel];
    for ( int i = 0; i < AnzArtikel; ++i ) {
```

```
      Artikelliste[i] = liesArtikel();
    }
    return Artikelliste;
  }

  /*Gebe einen Artikel auf der Konsole aus.*/
  static void zeigeArtikel ( Artikel Ding ) {
    double Gesamtwert = Ding.anzahl * Ding.preis;
    System.out.printf( "%10.2f %10.2f %6s %20s \n",
                       Gesamtwert, Ding.preis, Ding.anzahl, Ding.bezeichnung );
  }

  /*Gebe eine Artikelliste auf der Konsole aus.*/
  static void zeigeListe ( Artikel[] Artikelliste ) {
    System.out.printf( "%10s %10s %6s %20s \n",
                       "Gesamtwert", "Preis", "Anzahl", "Bezeichnung" );
    double Gesamtwert = 0.0;
    for ( int i = 0; i < Artikelliste.length; ++i ) {
      zeigeArtikel( Artikelliste[i] );
      Gesamtwert += Artikelliste[i].anzahl
                    * Artikelliste[i].preis;
    }
    System.out.printf( "----------\n%10.2f\n", Gesamtwert );
  }

  /*Hauptprogramm.*/
  public static void main ( String[] args ) {
    Locale.setDefault( Locale.US );
    sc = new Scanner( System.in );

    System.out.print( "Bitte Anzahl der verschiedenen Artikel eingeben: " );
    int AnzArtikel = sc.nextInt();

    /*Lese alle Artikel von der Konsole ein.*/
    Artikel[] Artikelliste = liesListe( AnzArtikel );

    /*Gebe alle Artikel auf der Konsole aus.*/
    zeigeListe( Artikelliste );
  }
}
```

# Aufgabe 34
# Klassen- und Instanzelemente

| *Bezeichner:* | klassen-elemente-1 |
|---|---|

### *Aufgabentext*

Die Klasse `Klasse` sei folgendermaßen definiert

```
public class Klasse {
    public static int variable1 = 1;
    public int variable2 = 2;
}
```

In der `main`-Methode eines Java-Programms findet man die Zeile

```
Klasse instanz = new Klasse();
```

Entscheiden Sie, welche der nachfolgenden Ausdrücke gültig sind. Begründen Sie Ihre Entscheidung.

(a) `Klasse.variable1 += 1;`

(b) `instanz.variable2 += 1;`

(c) `Klasse.variable2 += 1;`

(d) `instanz.variable1 += 1;`

### *Lösung*

Klassenelemente (d.h. Klassenvariablen und -methoden) existieren unabhängig von möglichen Instanzen einer Klasse. Instanzelemente (d.h. Instanzvariablen und -methoden) existieren nur für einzelne Instanzen einer Klasse. Daher ergeben sich folgende Lösungen:

(a) Der Ausdruck ist gültig.

(b) Der Ausdruck ist gültig.

(c) Der Ausdruck ist ungültig.

(d) Der Ausdruck ist gültig.

# Aufgabe 35
# Klassen- und Instanzelemente

| *Bezeichner:* | klassen-elemente-2 |
|---|---|

### *Aufgabentext*

Die Klasse `Klasse` sei folgendermaßen definiert

```
public class Klasse {
    private static int variable1 = 1;
    private int variable2 = 2;

    /* Methodendefinitionen */
}
```

Entscheiden Sie, welche der folgenden Methodendefinitionen für die Klasse gültig sind. Begründen Sie Ihre Entscheidung.

(a)
```
public static int methode1() {
    return variable1;
}
```

(b)
```
public int methode2() {
    return variable1;
}
```

(c)
```
public int methode3() {
    return variable2;
}
```

(d)
```
public static int methode4() {
    return variable2;
}
```

(e)
```
public int methode5() {
    return this.variable1;
}
```

(f)
```
public static int methode6() {
    return this.variable1;
}
```

***Lösung***

Klassenelemente (Klassenvariablen und -methoden) existieren unabhängig von Instanzen einer Klasse. Daher kann innerhalb einer Klassenmethode und innerhalb einer Instanzmethode auf andere Klassenelemente zugegriffen werden. Instanzelemente (d.h. Instanzvariablen und -methoden) existieren nur für Instanzen einer Klasse. Daher kann innerhalb einer Klassenmethode, die unabhängig von den Instanzen einer Klasse existiert, nicht auf Instanzelement zugegriffen werden. Innerhalb einer Instanzmethode ist dies möglich, da die Instanzmethode selbst nur für Instanzen der Klasse existiert. Auch die Selbstreferenz **`this`** existiert nur für die Instanzen einer Klasse. Daher ergeben sich folgende Lösungen:

(a) Die Methodendefinition ist gültig.

(b) Die Methodendefinition ist gültig.

(c) Die Methodendefinition ist gültig.

(d) Die Methodendefinition ist ungültig.

(e) Die Methodendefinition ist gültig.

(f) Die Methodendefinition ist ungültig.

# Aufgabe 36
# Klassenhierarchien

| *Bezeichner:* | klassen-hierarchie |
|---|---|

### *Aufgabentext*

Betrachten Sie die nachfolgenden Klassendefinitionen:

```
public class Wal { }

public class Zahnwal extends Wal { }

public class Pottwal extends Zahnwal { }

public class Delphin extends Zahnwal { }

public class Bartenwal extends Wal { }

public class Grauwal extends Bartenwal { }
```

In der `main`-Methode eines Java-Programms findet man die Zeilen

```
Wal w;
Zahnwahl z;
```

Entscheiden Sie, welche der folgenden Ausdrücke gültig sind. Begründen Sie Ihre Entscheidung.

(a) `z = new Zahnwal();`

(b) `z = new Delphin();`

(c) `w = new Bartenwal();`

(d) `z = new Bartenwal();`

(e) `w = new Grauwal();`

(f) `z = new Wal();`

(g) `z = (Zahnwal) new Wal();`

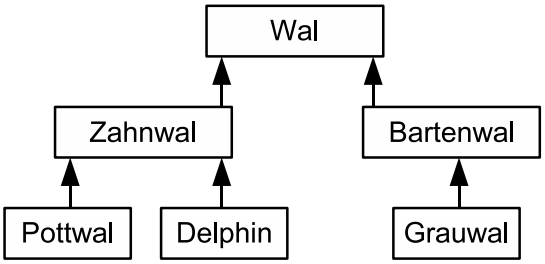
(h) `z = (Zahnwal) new Pottwal();`

(i) `z = (Zahnwal) new Grauwal();`

(j) `z = (Zahnwal) null;`

### *Lösung*

Für die Lösung dieser Aufgabe ist das folgende Klassendiagramm hilfreich. Jeder Pfeil zeigt von einer Klasse auf deren Basisklasse.




(a) Der Ausdruck ist gültig. Der Variable `z` vom Typ `Zahnwal` wird eine Instanz der Klasse `Zahnwal` zugewiesen.

(b) Der Ausdruck ist gültig. Die Klasse `Delphin` ist von der Klasse `Zahnwal` abgeleitet. Daher ist jede Instanz der Klasse `Delphin` auch eine Instanz der Klasse `Zahnwal` („Delphine sind Zahnwale.“).

(c) Der Ausdruck ist gültig. Die Klasse `Bartenwal` ist von der Klasse `Wal` abgeleitet. Daher ist jede Instanz der Klasse `Bartenwal` auch eine Instanz der Klasse `Wal` („Bartenwale sind Wale.“).

(d) Der Ausdruck ist ungültig. Die Klasse `Bartenwal` und die Klasse `Zahnwal` sind zwar beide von der Klasse `Wal` abgeleitet. Die Klasse `Bartenwal` ist jedoch nicht von der Klasse `Zahnwal` abgeleitet. Aus diesem Grund sind Instanzen der Klasse `Bartenwal` keine Instanzen der Klasse `Zahnwal` („Bartenwale sind keine Zahnwale.“).

(e) Der Ausdruck ist gültig. Die Klasse `Grauwal` ist über die Klasse `Bartenwal` von der Klasse `Wal` abgeleitet. Jede Instanz der Klasse `Grauwal` ist daher auch eine Instanz der Klasse `Wal` („Grauwale sind Wale.“).

(f) Der Ausdruck ist ungültig. Die Klasse `Wal` ist Basisklasse der Klasse `Zahnwal`. Daher ist ein Instanz der Klasse `Wal` keine Instanz der Klasse `Zahnwal` („Nicht jeder Wal ist ein Zahnwal.“).

(g) Der Ausdruck ist ungültig. Die Klasse `Wal` ist Basisklasse der Klasse `Zahnwal`. Die Typumwandlung von einer Basisklasse zu einer abgeleiteten Klasse ist nicht möglich.

(h) Der Ausdruck ist gültig. Die Klasse `Pottwal` ist von der Klasse `Zahnwal` abgeleitet. Die Typumwandlung von einer abgeleiteten Klasse zur entsprechenden Basisklasse ist möglich. Man beachte, dass die Typumwandlung auf die Instanz selbst keinen Einfluss hat, d.h. nach der Initialisierung ist in der Variable `z` eine Instanz der Klasse `Pottwal` gespeichert.

(i) Der Ausdruck ist ungültig. Die Klasse `Grauwal` ist nicht von der Klasse `Zahnwal` abgeleitet. Die Typumwandlung ist daher nicht möglich.

(j) Der Ausdruck ist gültig. Die leere Referenz `null` verweist auf keine Instanz. Die Typumwandlung wird ohne jeden Effekt durchgeführt.

# Aufgabe 37
# Zugriffsrechte

| *Bezeichner:* | klassen-zugriff |
|---|---|

***Aufgabentext***

Betrachten Sie die nachfolgenden Definitionen der Klassen `Basis` und `Ableitung`:

```
public class Basis {
    private void methode1() { }
    protected void methode2() { }
    public void methode3() { }
}


public class Ableitung extends Basis {
    private void methode4() { }
    protected void methode5() { }
    public void methode6() { }
}
```

In der `main`-Methode eines Java-Programms findet man die Zeilen

```
    Basis instanz1 = new Basis();
    Ableitung instanz2 = new Ableitung();
```

Die Klasse, in der die `main`-Methode definiert ist, gehöre dabei zu einem anderen Paket als die Klassen `Basis` und `Ableitung`. Sie sei weiterhin weder von der Klasse `Basis` noch von der Klasse `Ableitung` abgeleitet.

(a) Geben Sie die Bezeichner aller Methoden an, auf die innerhalb der Klasse `Basis` mittels **`this`** zugegriffen werden kann.

(b) Geben Sie die Bezeichner aller Methoden an, auf die innerhalb der Klasse `Ableitung` mittels **`this`** zugegriffen werden kann.

(c) Geben Sie die Bezeichner aller Methoden an, auf die innerhalb der Klasse `Ableitung` mittels `super` zugegriffen werden kann.

(d) Geben Sie die Bezeichner aller Methoden der Instanz `instanz1` an, auf die innerhalb der `main`-Methode zugegriffen werden kann.

(e) Geben Sie die Bezeichner aller Methoden der Instanz `instanz2` an, auf die innerhalb der `main`-Methode zugegriffen werden kann.

***Lösung***

Private Elemente (Variablen und Methoden) sind nur innerhalb einer Klasse sichtbar. Das bedeutet, man kann nur innerhalb solcher Methoden auf das Element zugreifen, die innerhalb derselben Klasse wie das private Element definiert sind. Geschützte Elemente sind nur innerhalb einer Klasse und innerhalb abgeleiteter Klassen sichtbar. Öffentliche Element sind global sichtbar. Daher ergeben sich die folgenden Lösungen:

(a) Innerhalb der Klasse `Basis` kann auf folgende Methoden mittels **`this`** zugegriffen werden: `methode1`, `methode2`, `methode3`.

(b) Innerhalb der Klasse `Ableitung` kann auf folgende Methoden mittels **`this`** zugegriffen werden: `methode2`, `methode3`, `methode4`, `methode5`, `methode6`.

(c) Innerhalb der Klasse `Ableitung` kann auf folgende Methoden mittels `super` zugegriffen werden: `methode2`, `methode3`.

(d) Innerhalb der `main`-Methode kann auf folgende Methoden der Instanz `instanz1` zugegriffen werden: `methode3`.

(e) Innerhalb der `main`-Methode kann auf folgende Methoden der Instanz `instanz2` zugegriffen werden: `methode3`, `methode6`.

# Aufgabe 38
# Polymorphie

| *Bezeichner:* | klassen-polymorphie |
|---|---|

***Aufgabentext***

Betrachten Sie die nachfolgenden Definition der Klassen `Basis` und `Ableitung`:

```
public class Basis {
    public static void methode1() {
        System.out.println("A");
    }
    public void methode2() {
        System.out.println("B");
    }
}


public class Ableitung extends Basis {
    public static void methode1() {
        System.out.println("C");
    }
    public void methode2() {
        System.out.println("D");
    }
}
```

In der `main`-Methode eines Java-Programms findet man die Zeilen

```
Basis instanz1 = new Basis();
Basis instanz2 = new Ableitung();
Ableitung instanz3 = new Ableitung();
Basis instanz4 = (Basis) new Ableitung();
```

Was wird bei den nachfolgenden Methodenaufrufen jeweils auf der Konsole ausgegeben?

(a) `instanz1.methode1();`

(b) `instanz2.methode1();`

(c) `instanz3.methode1();`

(d) `instanz4.methode1();`

(e) `instanz1.methode2();`

(f) `instanz2.methode2();`

(g) `instanz3.methode2();`

(h) `instanz4.methode2();`

***Lösung***

Klassenelemente (d.h. Klassenvariablen und -methoden) existieren unabhängig von etwaigen Instanzen einer Klasse. Daher ist bei der Definition einer Variable die Typendeklaration maßgebend dafür, welche Klassenelemente der Variable zugeordnet sind. Klassenmethoden werden bei Vererbung nicht überschrieben. Instanzelemente (d.h. Instanzvariablen und -methoden) existieren nur für einzelne Instanzen einer Klasse. Bei der Definition einer Variable wird daher durch die Initialisierung, d.h. durch den Aufruf des entsprechenden Konstruktors, festgelegt, welche Instanzelemente der Variable zugeordnet sind. Instanzvariablen können bei Vererbung überschrieben werden. Eine Typenkonvertierung hat keine Auswirkung auf die Instanz. Es ergeben sich demnach die folgenden Antworten:

(a) Es wird „A“ ausgegeben.

(b) Es wird „A“ ausgegeben.

(c) Es wird „C“ ausgegeben.

(d) Es wird „A“ ausgegeben.

(e) Es wird „B“ ausgegeben.

(f) Es wird „D“ ausgegeben.

(g) Es wird „D“ ausgegeben.

(h) Es wird „D“ ausgegeben.

# Aufgabe 39
# Komplexe Zahlen

| | |
|---|---|
| *Bezeichner:* | klassen-komplex |
| *Beschreibung:* | Darstellung komplexer Zahlen durch eine Klasse. |

***Aufgabentext***

Unter einer *komplexen Zahl* $z$ versteht man eine Zahl der Form $z = a + b\mathrm{i}$, wobei $a$ und $b$ zwei reelle Zahlen und $\mathrm{i}$ die sogenannte *imaginäre Einheit* ist. Die imaginäre Einheit ist durch $\mathrm{i}^2 = -1$ definiert. Die Zahl $a$ heißt *Realteil*, die Zahl $b$ *Imaginärteil* von $z$. Die Menge der komplexen Zahlen wird mit $\mathbb{C}$ bezeichnet und kann als algebraische Erweiterung der reellen Zahlen $\mathbb{R}$ verstanden werden. Der *Betrag* $|z|$ einer komplexen Zahl $z = a + b\mathrm{i}$ ist wie folgt definiert:

$$|z| = \sqrt{a^2 + b^2}.$$

Für die *Addition* zweier komplexer Zahlen $z = a + b\mathrm{i}$ und $w = c + d\mathrm{i}$ gilt:

$$z + w = (a + c) + (b + d)\mathrm{i}.$$

Für die *Multiplikation* gilt entsprechend

$$zw = (ac - bd) + (ad + bc)\mathrm{i}.$$

Diese Formel folgt nach Ausmultiplizieren direkt aus der Definition $\mathrm{i}^2 = -1$. Schreiben Sie ein Java–Programm, welches das Rechnen mit komplexen Zahlen ermöglicht.

1. Erstellen Sie eine öffentliche Klasse namens `Komplex` die eine komplexe Zahl repräsentiert. Definieren Sie für diese Klasse zwei private Instanzvariablen namens `real` und `imag` vom Typ **`double`**, die den Realteil bzw. den Imaginärteil einer komplexen Zahl speichern.

2. Definieren Sie einen öffentlichen Konstruktor mit zwei formalen Parametern namens `real` und `imag` vom Typ **`double`**. Weisen Sie in diesem Konstruktor die Parameterwerte den gleichnamigen Instanzvariablen zu. Verwenden Sie dazu das Schlüsselwort **`this`**.

3. Definieren Sie für die Klasse `Komplex` eine öffentliche Instanzmethode namens `toString`, ohne formale Parameter. Die Methode soll die Zeichenkette „$a$ + i$b$“ als Wert vom Typ `String` zurück geben, wobei $a$ und $b$ durch die Werte der Instanzvariablen `real` und `imag` zu ersetzen sind. Wenn Sie nun den Befehl `System.out.println(z);` für eine Instanz `z` der Klasse `Komplex` aufrufen, wird die von der Methode `toString` zurückgegebene Zeichenkette auf der Konsole ausgegeben.

4. Definieren Sie für die Klasse `Komplex` eine öffentliche Instanzmethode namens `betrag` ohne formale Parameter, welche den Betrag der komplexen Zahl berechnet und als Wert vom Typ **`double`** zurückgibt.

5. Definieren Sie für die Klasse `Komplex` zwei öffentliche Klassenmethoden mit den Namen `addieren` und `multiplizieren`. Versehen Sie beide Methoden mit zwei formalen Parametern vom Typ `Komplex`. Berechnen Sie in den Methoden die Summe bzw. das Produkt der übergebenen komplexen Zahlen und geben Sie das Ergebnis jeweils als Wert vom Typ `Komplex` zurück.

6. Erstellen Sie eine öffentliche Klasse namens `KomplexeArithm` mit der `main`-Methode des Programms. Lesen Sie in der `main`-Methode den Real- und Imaginärteil zweier komplexer Zahlen von der Konsole ein und erzeugen Sie zwei entsprechende Instanzen der Klasse `Komplex`. Geben Sie die Beträge beider komplexer Zahlen, sowie deren Summe und Produkt auf der Konsole aus.

7. Testen Sie Ihr Programm mit den folgenden Daten:

$$z = 1 + 1\mathrm{i}, \quad w = 2 + 2\mathrm{i}, \quad |z| = \sqrt{2}, \quad |w| = \sqrt{8}, \quad z + w = 3 + 3\mathrm{i}, \quad zw = 4\mathrm{i};$$
$$u = 2\mathrm{i}, \quad v = 4\mathrm{i}, \quad |u| = 2, \quad |v| = 4, \quad u + v = 6\mathrm{i}, \quad uv = -8.$$

### *Lösung*

Komplex.java

```
/** Diese Klasse repraesentiert eine komplexe Zahl. */
public class Komplex {

  /** real und imaginaer Teil. */
  private double real;
  private double imag;

  /** Konstruktor. */
  public Komplex( double real_, double imag_ ) {
    real = real_;
    imag = imag_;
  }

  /** toString-Methode. */
  public String toString() {
    return "" + real + " + i" + imag;
  }

  /** Betrag der komplexen Zahl. */
  public double betrag() {
    return Math.sqrt( real*real + imag*imag );
  }

  /** Gibt die Summe zweier komplexer Zahlen zurueck. */
  public static Komplex addieren( Komplex z1, Komplex z2 ) {
```

```
    return new Komplex( z1.real + z2.real, z1.imag + z2.imag );
  }

  /** Gibt das Produkt zweier komplexer Zahlen zurueck. */
  public static Komplex multiplizieren( Komplex z1, Komplex z2 ) {
    double x1 = z1.real;
    double x2 = z2.real;
    double y1 = z1.imag;
    double y2 = z2.imag;
    return new Komplex( x1 * x2 - y1 * y2, x1 * y2 - x2 * y1 );
  }
}
```

KomplexeArithm.java

```
import java.util.*;

public class KomplexeArithm {

  public static void main( String[] args ) {
    Locale.setDefault( Locale.US );
    Scanner sc = new Scanner( System.in );

    System.out.println( "Geben Sie den Real- und Imaginaerteile einer " +
                        "komplexen Zahl z ein:" );
    double x = sc.nextDouble();
    double y = sc.nextDouble();
    Komplex z = new Komplex( x, y );

    System.out.println( "Geben Sie den Real- und Imaginaerteile einer " +
                        "komplexen Zahl w ein:" );
    x = sc.nextDouble();
    y = sc.nextDouble();
    Komplex w = new Komplex( x, y );

    System.out.println( "|z| = " + z.betrag() );
    System.out.println( "|w| = " + w.betrag() );
    System.out.println( "z + w = " + Komplex.addieren( z, w ) );
    System.out.println( "z * w = " + Komplex.multiplizieren( z, w ) );
  }
}
```

# Teil II

# Anhang

# Anhang A

# Das Paket `exercises`

## A.1 Einführung

Das Paket `exercises` definiert Makros und Umgebungen, mit denen Aufgaben erstellt und anschließend in Aufgabenblätter oder -verzeichnisse eingefügt werden können. Das Paket ermöglicht es, Aufgaben als separaten Dateneinheiten zu verwalten, und sie bei Bedarf in andere LaTeX–Dokumente einzubinden.

## A.2 Aufgaben erstellen

In diesem Abschnitt wird anhand eines konkreten Beispiels gezeigt, wie man eine Aufgabe erstellt, die dann in Aufgabenblätter oder -verzeichnisse eingefügt werden kann. In der Aufgabe soll es darum gehen, die quadratische Gleichung

$$x^2 - 7x + 12 = 0$$

mit der Mitternachts–Formel zu lösen. Dazu müssen ein Aufgabentext und eine Musterlösung formuliert werden. Ferner sollen zusätzliche Informationen über die Aufgabe zur Verfügung gestellt werden.

Um eine Aufgabe zu erstellen muss zunächst ein *Aufgabenordner* angelegt werden. Der Name des Aufgabenordners dient als Bezeichner für die Aufgabe und darf **keine Umlaute oder Sonderzeichen außer dem Minuszeichen** (-) enthalten. Da im vorliegenden Beispiel eine quadratische Gleichung gelöst werden soll, erstellen wir einen Ordner mit dem Namen `quadr-gleichung` im aktuellen Verzeichnis, so dass sich für den Aufgabenordner der relative Pfad

```
./quadr-gleichung
```

ergibt. Als nächstes wird dem angelegten Aufgabenorder die *Aufgabendatei* erstellt. Die Aufgabendatei ist eine LaTeX–Quelltextdatei, welche **denselben Namen wie der Aufgabenorder** trägt. Wir erstellen daher im Aufgabenorder eine Datei mit dem Namen `quadr-gleichung.tex`. Der relative Pfad der Aufgabendatei ist dementsprechend durch

```
./quadr-gleichung/quadr-gleichung.tex
```

gegeben. Die grundlegenden Syntax einer Aufgabendatei sieh folgendermaßen aus:

```
\begin{exercise}{ Überschrift }
  \description{ Kurze Beschreibung }
  \difficulty{ Schwierigkeit }
  \source{ Herkunft }
  \utilization{ Verwendung }

  \begin{comments}
    Kommentare
  \end{comments}

  \begin{body}
    Aufgabentext
  \end{body}

  \begin{solution}
    Musterlösung
  \end{solution}
\end{exercise}
```

Die Umgebung `exercise` fasst eine Aufgabe als Datenstruktur zusammen. Sie besitzt ein Argument, über welches die Überschrift der Aufgabe definiert wird. Ähnlich zu den LaTeX–Makros `\section`, `\subsection` oder `\subsubsection` kann der Umgebung `exercise` auch ein optionales Argument übergeben werden, welches eine alternative Überschrift definiert. Die alternative Überschrift wird dann beispielsweise bei der Erstellung von Inhaltsverzeichnissen verwendet.

Das Makro `\description` definiert eine kurze Beschreibung der Aufgabe. Aus einer solchen Beschreibung sollte das Thema und/oder das Lernziel der Aufgabe hervorgehen. Über das Makro `\difficulty` kann eine Angabe zur Schwierigkeit der Aufgabe gemacht werden. Über das Makro `\source` kann angegeben werden, aus welcher Quelle die Aufgabe stammt. Mit dem Makro `\utilization` können Angaben darüber gemacht werden, in welchen Semestern und in welchen Veranstaltungen die Aufgabe bereits gestellt wurden. Insbesondere ehemalige Klausuraufgaben können so kenntlich gemacht werden. Alle genannten Makros können optional angegeben werden, d.h. ihre Verwendung ist nicht zwingend erforderlich.

Die Umgebung `comments` dient dazu, Hinweise für den Dozenten oder Übungsleiter zu hinterlassen. So können beispielsweise eventuelle Fehler in der Musterlösung vermerkt werden, die bei einer späteren Überarbeitung der Aufgabe zu korrigieren sind. Die Umgebung `comments` kann optional angegeben werden. Die wichtigste Umgebung ist die Umgebung `body`. Sie muss zwingend verwendet werden, und dient dazu, den Aufgabentext zu definieren. Schließlich kann in der Umgebung `solution` optional eine Musterlösung zur Aufgabe angegeben werden.

Wenden wir uns nun unserer Beispielaufgabe zu. Für sie könnte der Quelltext der Aufgabendatei folgendermaßen aussehen:

```
\begin{exercise}{Eine quadratische Gleichung}
  \description{Lösen einer quadratischen Gleichung
  mit der Mitternachts--Formel}
  \difficulty{leicht}
  \source{unbekannt}
  \utilization{WS06 (Arithmetik I)}

  \begin{comments}
    Der Umfang dieser Aufgabe ist zu gering und
    sollte erweitert werden.
  \end{comments}

  \begin{body}
    Für $x \in \mathbb{R}$ sei die folgende,
    quadratische Gleichung gegeben:
      \[ x^2 - 7x + 12 = 0. \]
    Lösen Sie diese Gleichung mit der
    Mitternachts--Formel.
  \end{body}

  \begin{solution}
    Laut Mitternachts--Formel sind die Lösungen
    $x_1$ und $x_2$ der Gleichung durch
      \[ x_{1,2} = \frac{7 \pm \sqrt{49 - 48}}{2} \]
    gegeben. Man rechnet leicht nach dass
      \[ x_1 = 4,\quad x_2 = 3. \]
  \end{solution}
\end{exercise}
```

Die Beispielaufgabe wurde offenbar mit der Überschrift „Eine quadratische Gleichung“versehen. Die Beschreibung der Aufgabe lautet „Lösen einer quadratischen Gleichung mit der Mitternachts–Formel“. Die Aufgabe von der Schwierigkeit her als „leicht“ eingestuft. Ihre Herkunkft ist nicht bekannt. Ferner wurde angegeben, dass die Aufgabe bereits im Wintersemester des Jahres 2006 im Rahmen der Vorlesung „Arithmetik I“ gestellt wurde. Der Kommentar bezieht sich auf den Umfang der Aufgabe, der als zu gering eingestuft wird. Es folgen der Aufgabentext und die Musterlösung.

## A.3 Aufgabenblätter erstellen

Um ein Aufgabenblatt zu erstellen, legt man eine separates LaTeX–Dokument an, in dessen Präambel man das Paket `exercises` durch die Quelltextzeile

```
\usepackage{exercises}
```

einbindet. Im Dokument (d.h. innerhalb der Umgebung `document`) können einzelne Aufgaben dann mit dem Makro `\inputexercise` eingefügt werden. Dieses Makro besitzt ein Argument über das der **Bezeichner des Aufgabenordners** der jeweiligen Aufgabe übergeben wird. Zuvor muss jedoch angegeben werden, in welchem Verzeichnis sich die einzelnen Aufgabenordner befinden. Dies geschieht mit dem Makro `\exercisespath`. Wir erinnern uns, dass der Pfad des Aufgabenordners der Beispielaufgabe `./quadr-gleichung` lautete. Der Aufgabenorder befindet sich also im lokalen Verzeichnis. Diese wird durch die Quelltextteile

```
\exercisespath{./}
```

angegeben. Der Schrägstrich (`/`) muss in diesem Fall zwingend angegeben werden. Um nun die Beispielaufgabe in das Dokument einzufügen, verwendet man die Quelltextzeile

```
\inputexercise{quadr-gleichung}
```

Dadurch wird die folgende Ausgabe erzeugt:

---

**Aufgabe 1** *Eine quadratische Gleichung*

Für $x \in \mathbb{R}$ sei die folgende, quadratische Gleichung gegeben:

$$x^2 - 7x + 12 = 0.$$

Lösen Sie diese Gleichung mit der Mitternachts–Formel.

---

Wie man sieht, wurden lediglich die Überschrift und der Aufgabentext gedruckt. Die Musterlösung sowie alle übrigen Information bleiben verborgen. Jede eingefügte Aufgabe wird mit einer fortlaufenden Numerierung versehen. Die Numerierung wird über einen Zähler namens `exercise` realisiert. Dieser kann wie alle anderen Zähler auch mittels `\setcounter` manipuliert werden. Um das Format der Numerierung zu ändern, kann man das Makro `\theexercise` neu definieren.

Man kann eine Aufgabe auch direkt in einem LaTeX–Dokument definieren. Dazu fügt man an der Stelle, an der die Aufgabe erscheinen soll, eine entsprechende `exercise`–Umgebung ein.

## A.4 Aufgabenblätter mit Musterlösung erstellen

Wird eine Aufgabe in ein Aufgabenblatt eingefügt, so werden zunächst nur die Überschrift und der Aufgabentext gedruckt. Um die Musterlösung zu drucken, verwendet man das Makro

```
\printsolutions
```

Im Fall der Beispielaufgabe führt dieser Befehl zur nachfolgenden Ausgabe:

---

**Lösung 1** *Eine quadratische Gleichung*

Laut Mitternachts–Formel sind die Lösungen $x_1$ und $x_2$ der Gleichung durch

$$x_{1,2} = \frac{7 \pm \sqrt{49 - 48}}{2}$$

gegeben. Man rechnet leicht nach dass

$$x_1 = 4, \quad x_2 = 3.$$

---

Das Makro `\printsolutions` veranlasst, dass die Musterlösungen **aller** zuvor eingefügten Aufgaben nacheinander gedruckt werden.

## A.5 Aufgaben anpassen

### A.5.1 Verrechnungspunkte festlegen

Verrechnungspunkte können mit dem Makro `\scorepoints` für eine Aufgabe festgelegt werden. Das Makro besitzt ein Argument über das die Anzahl der Verrechnungspunkte übergeben wird. Das Makro `\scorepoints` wirkt dabei nur auf die unmittelbar nachfolgende Aufgabe. Die Quelltextzeile

```
\scorepoints{4}\inputexercise{quadr-gleichung}
```

führt bespielsweise zu der folgenden Ausgabe:

---

**Aufgabe 1** *Eine quadratische Gleichung* (4 Punkte)

Für $x \in \mathbb{R}$ sei die folgende, [...]

---

### A.5.2 Freiwillige Aufgabe und Pflichtaufgaben festlegen

Mit den Makros `\mandatory` und `\voluntary` können Aufgaben als Pflichtaufgabe bzw. als freiwillige Aufgabe gekennzeichnet werden. Beide Makros wirken nur auf die unmittelbar nachfolgende Aufgabe. Die Quelltextzeilen

```
\mandatory\inputexercise{quadr-gleichung}
```

und

```
\voluntary\inputexercise{quadr-gleichung}
```

erzeugen beispielsweise die Ausgaben

---

**Aufgabe 1 (Pflichtaufgabe)** *Eine quadratische Gleichung*

Für $x \in \mathbb{R}$ sei die folgende, [...]

---

bzw.

---

**Aufgabe 1 (freiwillig)** *Eine quadratische Gleichung*

Für $x \in \mathbb{R}$ sei die folgende, [...]

---

### A.5.3 Aufgabentypen spezifizieren

Mit dem `\exercisetype` kann der Typ einer Aufgabe näher spezifiziert werden. Das Makro besitzt ein Argument, über den die Spezifizierung übergeben wird. Das Makro wirkt nur auf die unmittelbar nachfolgende Aufgabe. Die Quelltextzeile

```
\exercisetype{Rechenaufgabe}\inputexercise{quadr-gleichung}
```

erzeugt beispielsweise die Ausgabe

---

**Aufgabe 1 (Rechenaufgabe)** *Eine quadratische Gleichung*

Für $x \in \mathbb{R}$ sei die folgende, [...]

---

### A.5.4 Musterlösungen verbergen

Wird das Makro `\nosolution` einer Aufgabe vorangestellt, so wird deren Musterlösung nicht gedruckt, wenn das Makro `\printsolutions` aufgerufen wird. Das Makro `\nosolution` wirkt nur auf die unmittelbar nachfolgende Aufgabe.

### A.5.5 Abstände anpassen

Mit den Makros `\atendbody`, `\atendsolution`, `\atendexercise` kann festgelegt werden, was am Ende eines Aufgabentextes, eines Musterlösung oder einer Aufgabe in den LaTeX–Quelltext eingefügt werden soll. Diese Makros können beispielsweise dazu verwendet werden, am Ende jeder Aufgabe einen Seitenumbruch zu erzwingen.

## A.6 Weitere Makros und Umgebungen

### A.6.1 Aufgabenteile festlegen

Innerhalb einer `exercise`–Umgebung ist die Aufzählungsumgebung `parts` definiert. Diese dient dazu, Aufgabenteile aufzuzählen. Der Quelltext

```
\begin{parts}
  \item Zeigen Sie: Die Funktion $f$ ist stetig.
  \item Zeigen Sie: Die Funktion $f$ besitzt einen Fixpunkt.
  \item Berechnen Sie einen Näherungswert für den Fixpunkt.
\end{parts}
```

erzeugt beispielsweise die Ausgabe

---

(a) Zeigen Sie: Die Funktion $f$ ist stetig.

(b) Zeigen Sie: Die Funktion $f$ besitzt einen Fixpunkt.

(c) Berechnen Sie einen Näherungswert für den Fixpunkt.

---

Anstelle von `\item` kann innerhalb der `parts`–Umgebung auch `\part` verwendet werden. Dies sollte jedoch vermieden werden, da das Makro `\part` bereits in LaTeX definiert ist.

### A.6.2 Hinweise einfügen

Das Makro `\hint` kann innerhalb einer `exercise`–Umgebung dazu verwendet werden, Aufgabenhinweise kenntlich zu machen. Der Quelltext

```
Zeigen Sie: Alle Eigenwerte der Matrix $A$
besitzen einen positiven Realteil.

\hint Verwenden Sie den Satz von Gerschgorin.
```

erzeugt beispielsweise die Ausgabe

---

Zeigen Sie: Alle Eigenwerte der Matrix $A$ besitzen einen positiven Realteil.

**Hinweis:** Verwenden Sie den Satz von Gerschgorin.

---

### A.6.3 Dateien und Grafiken in Aufgaben einbinden

Möchte man den Inhalt anderer Dateien oder Grafiken in den den Aufgabentext oder die Musterlösung einer Aufgabe einfügen, so sollten die entsprechenden Dateien im Aufgabenordner gespeichert werden. Mit den Quelltextzeilen

```
\input{\filename{meineDatei.tex}}
```

und

```
\includegraphics{\filename{meineGrafik.eps}}
```

können dann Dateien bzw. Graphiken eingefügt werden. Das Makro `\filename` setzt dabei einen relativen Pfadnamen **bezüglich der Aufgabendatei** in einen relativen Pfadnamen **bezüglich der Hauptdatei** um. Diese Makro ist nur innerhalb der `exercise`–Umgebung definiert.

## A.7 Aufgabenverzeichnisse erstellen

Aufgabenverzeichnisse werden ähnlich wie Aufgabenblätter erstellt. Man erstellt dazu ein separates LaTeX–Dokument, in dessen Präambel man die Quelltextzeile

```
\usepackage[catalogue]{exercises}
```

einfügt. Einzelne Aufgaben können danach über das Makro `\inputexercise` in das Verzeichnis eingefügt werden. Der Unterschied besteht in der Darstellung der Aufgaben. Fügt man beispielsweise die Beispielaufgabe in das Verzeichnis ein, so führt dies zu der folgenden Ausgabe:

---

**Aufgabe 1**
**Eine quadratische Gleichung**

| | |
|---|---|
| *Bezeichner:* | quadr-gleichung |
| *Beschreibung:* | Lösen einer quadratischen Gleichung mit der Mitternachts–Formel |
| *Schwierigkeit:* | leicht |
| *Quelle:* | unbekannt |
| *Verwendung:* | WS06 (Arithmetik I) |

***Kommentare***
Der Umfang dieser Aufgabe ist zu gering und sollte erweitert werden.

***Aufgabentext***
Für $x \in \mathbb{R}$ sei die folgende, quadratische Gleichung gegeben:

$$x^2 - 7x + 12 = 0.$$

Lösen Sie diese Gleichung mit der Mitternachts–Formel.

***Lösung***
Laut Mitternachts–Formel sind die Lösungen $x_1$ und $x_2$ der Gleichung durch

$$x_{1,2} = \frac{7 \pm \sqrt{49 - 48}}{2}$$

gegeben. Man rechnet leicht nach dass

$$x_1 = 4, \quad x_2 = 3.$$

---

Man erkennt, dass in einem Verzeichnis alle Informationen zu einer Aufgabe gedruckt werden. Das Layout unterscheidet sich außerdem grundlegend von dem Layout, welches für Aufgabenblätter verwendet wird. In Verzeichnissen bleiben die Makros `\scorepoints`, `\mandatory`, `\voluntary` und `\exercisetype` wirkungslos.