Aufgabensammlung
Tutorium zur Einführung in die Programmierung
- Praktischer Teil: Java-Programmierung -

Inhaltsverzeichnis

1 Kontrollstrukturen

Aufgabe 1.1
Schreiben Sie ein Java-Programm, das den Wert einer Variablen vom Typ `int` in Binärdarstellung ausgibt, d.h. als Folge von Nullen und Einsen. Verwenden Sie dazu in geeigneter Weise die beiden Operatoren % und / und nutzen Sie die Tatsache aus, dass ein Wert vom Typ `int` 32 bit hat.
Hinweis:
Eine `while`-Schleife hat in Java folgende Syntax:
`while (` *boolescher Ausdruck* `) {` *Anweisung* `}`
Denken Sie daran, dass der +-Operator, wenn man ihn auf eine int- und eine String-Variable anwendet, die int-Variable in einen String umwandelt und dann verkettet.

Lösung zu 1.1
Das Programm

```
public class BinaerTest {
   public static void main (String args[ ]) {
      int eingabe;
      int rest;
      int i; // Schleifenzaehler
      String s; // fuer die Ausgabe

      // Eingabe mit Wert belegen
      eingabe = 49;

      // Initialisierung der Ausgabe
      s = "";

      // Schleife
      i = 1;
      while ( i <= 32) {
         rest = eingabe%2; // Berechnung einer Binaerstelle
         eingabe = eingabe/2;

         // berechnete Binaerstelle hinten an Ausgabe anhaengen
         s = rest + s;

         i = i+1; // schleifenzaehler erhoehen
      }

      // Ausgabe des Ergebnisses
      System.out.println(s);
   }
}
```

liefert die Ausgabe `00000000000000000000000000111101`.

Aufgabe 1.2
Welche Ausgabe liefert folgendes Java-Programm?

```
public class Var1 {
   public static void main(String[] args) {
      int a = 1, b = 2, c = 3;
      c = c + a + b;
      a = c - a - b;
      b = c - a - b;
      c = c - a - b;
      System.out.println("a = " + a);
      System.out.println("b = " + b);
      System.out.println("c = " + c);
   }
}
```

Lösung zu 1.2
Die folgende Tabelle zeigt die Belegung der Variablen vor bzw./ nach den angegebenen Anweisungen:

	a	b	c
	1	2	3
$c = c + a + b$			
	1	2	6
$a = c - a - b$			
	3	2	6
$b = c - a - b$			
	3	1	6
$c = c - a - b$			
	3	1	2

Das Programm liefert also die Ausgabe

```
a = 3
b = 1
c = 2
```

Aufgabe 1.3
In folgendem Programm werden zwei Integervariablen mit Eingabewerten aus der Kommandozeile belegt und anschließend die eingegebenen Werte wieder ausgegeben.

```
public class IntAdd {
    public static void main (String args[ ]) {
        int i1
        int i2;

        // Belegung der Variablen mit Eingabewerten
        i1 = Integer.parseInt(args[0]);
        i2 = Integer.parseInt(args[1]);

        // Ausgabe der Variablenwerte
        System.out.println ("i1: " + i1);
        System.out.println ("i2: " + i2);
    }
}
```

Beim Programmstart müssen dann entsprechende Werte (Integer-Konstanten) angegeben werden.

Beispiel:
Übersetzen mit `javac IntAdd.java`
Starten mit `java IntAdd 23 65`
liefert die Ausgabe:

```
i1: 23
i2: 65
```

In der Folge können bei jedem Programmaufruf andere Werte angegeben werden, ohne dass ein erneutes Überstzen notwendig wäre.

Ergänzen Sie das angegebene Programm um eine Methode, die die beiden Eingabewerte addiert. Dise Methode soll im Hauptprogramm aufgerufen und das Ergebnis ausgegeben werden.

Lösung zu 1.3
Die Additionsmethode bekommt zwei Integerwerte als Parameter übergeben und liefert einen Interwert als Ergebnis zurück, der Methodenkopf ist also

```
public static int addiere(int x, int y)
```

Der Rumpf soll nun die Summe berechnen. Dazu kann entweder eine lokale (Hilfs-)variabel benutzt werden:

```
{
    int z;
    z = x + y;
    return z;
}
```

oder man verzichtet auf die Hilfsvariable:

```
{
    return x + y;
}
```

Im Hauptprogramm fügen wir den Methodenaufruf `summe = addiere(i1, i2);` ein.

Insgesamt sieht das Programm dann so aus:

```
public class IntAdd {
    public static void main (String args[ ]) {

        int i1; // erster Eingabewert
        int i2; // zweiter Eingabewert
        int summe; // Ergebnisvariable

        // Belegung mit den Eingabewerten
        i1 = Integer.parseInt(args[0]);
        i2 = Integer.parseInt(args[1]);

        // Additionsmethode aufrufen
        summe = addiere(i1, i2);

        // Ergebnisausgabe
        System.out.println ("i1: " + i1);
        System.out.println ("i2: " + i2);
        System.out.println("Summe: " + summe);
    }

    // Additionsmethode definieren
    public static int addiere(int x, int y) {
        int z;
        z = x + y;
        return z;

        // es geht auch mit nur einer Anweisung und ohne die Hilfsvariable z
        // return (x+y);
    }
}
```

Aufgabe 1.4
Seit Jahrtausenden versuchen die Menschen, die Kreiszahl π, die definiert ist als Kreisumfang dividiert durch Kreisdurchmesser, möglichst exakt zu bestimmen.

So rechneten die Babylonier ca. 2000 v. Chr. mit dem Wert $3\frac{1}{8}$,
in Ägypten ist seit ungefähr der gleichen Zeit der Wert $(\frac{16}{9})^2$ überliefert,
in China wurde seit ca. 250 n. Chr. $\sqrt{10}$ benutzt
und in Europa ist seit Archimedes (ca. 250 v. Chr.) $3\frac{1}{7}$ eine bekannte Näherung.

Bestimmen Sie diese Werte und berechnen Sie die jeweilige Abweichung von dem auf 15 Dezimalstellen exakten Wert, wie er in der Java-Konstanten `Math.PI` zur Verfügung gestellt wird.

Hinweis:
Die Wurzelfunktion können Sie in Java mittels `Math.sqrt(` *einzusetzender Wert*) aufrufen.

Lösung zu 1.4

```
public class piAntik {

    public static void main (String[] args) {

        double piBabylon;
        double piAegypten;
        double piChina;
        double piEuropa;

        piBabylon = 3.0 + (1.0/8.0);
        piAegypten = 16.0*16.0 / (9.0*9.0);
        piChina = Math.sqrt(10.0);
        piEuropa = 3.0 + 1.0/7.0;
```

```
        System.out.println ("Java:      " + Math.PI);
        System.out.print ("Babylon:  " + piBabylon );
        System.out.println ( "\t\t\t  Abweichung: " + (Math.PI - piBabylon));
        System.out.print ("Aegypten: " + piAegypten );
        System.out.println ( "\t\t\t  Abweichung: " + (Math.PI - piAegypten));
        System.out.print ("China:     " + pi );
        System.out.println ( "\t\t\t  Abweichung: " + (Math.PI - piChina));
        System.out.print ("Europa:    " + piEuropa );
        System.out.println ( "\t\t\t  Abweichung: " + (Math.PI - piEuropa));
    }
}
```

Aufgabe 1.5
Ihnen wird folgender Handytarif angeboten:
Der Basispreis beträgt 22,98 Euro/Monat, darin sind 30 freie Gesprächsminuten enthalten. Jede darüber hinausgehende Gesprächsminute kostet 0,248 Euro.
Schreiben Sie ein Java-Programm, das den Rechnungsbetrag berechnet. Die Anzahl der Gesprächsminuten soll beim Programmstart in der Kommandozeile als Parameter übergeben werden.

Lösung zu 1.5

```
public class TarifB{
    public static void main(String[] args) {
        double preisProEinheit = 0.248;
        double grundgeb = 22.98;
        int dauer;
        double betrag;

        dauer = Integer.parseInt(args[0]);
        if (dauer > 30)
            betrag = (dauer - 30)* preisProEinheit + grundgeb;
        else
            betrag = grundgeb;
        System.out.println("Rechnungsbetrag: " + betrag + " Euro");
    }
}
```

Das Programm liefert beim Aufruf mit 30 bzw. 31 Minuten die Beträge:

```
>java TarifB 30
Rechnungsbetrag: 22.98 Euro
>java TarifB 31
Rechnungsbetrag: 23.228 Euro
```

Aufgabe 1.6
Wandeln Sie die Zählschleife des folgenden Beispiels in eine äquivalente fußgesteuerte Schleife um. Testen Sie Ihr Programm mit den Eingabewerten $n = 0, n = 1$ und $n = 5$.

```
int i, n, summe;
summe = 0;
for (i = 1; i <= n; i = i+1)
    summe = summe + i;
```

Lösung zu 1.6

```
public class ForDo {

    public static void main(String[] args) {
        int i;
        int summeFor;
        int summeDoWhile;
        int n;

        n = Integer.parseInt(args[0]);
```

```
        // Berechnung mit for-Schleife
        summeFor = 0;
        for ( i = 1; i <= n; i = i+1)
            summeFor = summeFor + i;

        // Berechnung mit do-while-Schleife
        summeDoWhile = 0;
        i = 1;
        if (i <= n) {
            do {
                summeDoWhile = summeDoWhile + i;
                i = i+1;
            } while (i <= n);
        }
        System.out.println ("Summe von 1 bis " + n);
        System.out.println ("Ergebnis for-Schleife: " + summeFor);
        System.out.println ("Ergebnis do-while-Schleife: " + summeDoWhile);
    }
}
```

Das Programm liefert die korrekte Ausgabe

```
> java ForDo 5
Summe von 1 bis 5
Ergebnis for-Schleife: 15
Ergebnis do-while-Schleife: 15
> java ForDo 1
Summe von 1 bis 1
Ergebnis for-Schleife: 1
Ergebnis do-while-Schleife: 1
> java ForDo 0
Summe von 1 bis 0
Ergebnis for-Schleife: 0
Ergebnis do-while-Schleife: 0
```

Aufgabe 1.7 (EulerE)
Die Euler'sche Zahl e hat folgende Reihendarstellung:

$$e = \sum_{k=0}^{\infty} \frac{1}{k!}$$

Schreiben Sie ein Java-Programm, das den Wert dieser Summe berechnet

a) bis zum 100. Reihenglied
b) bis der neue Summand < 0.00000001 ist.

Lösung zu 1.7

```
public class EulerE {

    public static void main (String[] args) {
        double e, summand;
        int k;

        // Loesung fuer Teil a)
        e = 1.0;
        summand = 1.0;
        for (k = 1; k < 100; k = k+1) {
            summand = summand/k;
            e = e + summand;
        }
        System.out.println("Naeherung fuer e nach " + k + " Iterationen: " + e);

        // Loesung fuer Teil b)
        e = 1.0;
        summand = 1.0;
```

```
        k = 1;
        do {
            summand = summand/k;
            e = e + summand;
            k = k+1;
        } while (summand >= 0.00000001);
        System.out.println("Naeherung fuer e nach " + k + " Iterationen: " + e);
    }
}
```

2 Methoden

Aufgabe 2.1
Die *harmonische Reihe* ist definiert als unendliche Summe

$$\sum_{k=1}^{\infty} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots$$

Implementieren Sie eine Methode, die die harmonische Reihe bis zu einer vorgegebenen Anzahl von Reihengliedern berechnet.

Sie können Ihre Methode in diese Testumgebung einbetten:

```
public class HarmonischeReihe {

    public static void main (String[] args) {

        int n;
        double sum;

        n = Integer.parseInt(args[0]);

        // Methodenaufruf
        // sum = ....;

        System.out.println(sum);

    }

    // Methodendefinition

}
```

Lösung zu 2.1

```
public class HarmonischeReihe {

    public static void main (String[] args) {

        int n;
        double sum;

        n = Integer.parseInt(args[0]);

        sum = berechneReihe(n);

        System.out.println(sum);

    }


    public static double berechneReihe(int n) {
        double s = 0.0;
        for (int k = 1; k <= n; k++)
            s = s + 1.0/k;
        return s;
```

```
    }
}
```

Beispielaufrufe und Ergebnisse:

```
> java HarmonischeReihe 2
1.5

> java HarmonischeReihe 10
2.9289682539682538
```

Aufgabe 2.2
Für ein Seminar muß jeder Teilnehmer eine schriftliche Hausarbeit anfertigen und einen mündlichen Vortrag halten. Dafür sind jeweils maximal 10 Punkte erhältlich. Zur Ermittlung der Gesamtpunktzahl wird die Punktzahl der Hausarbeit mit 3, die des Vortrags mit 2 multipliziert. Das folgende Java-Programm soll die Gesamtpunktzahl eines Studenten berechnen, der für die Hausarbeit 5 Punkte, für den Vortrag 8 Punkte erhalten hat. Was geschieht? Wie kann man den Fehler beheben?

```
public class Seminarunkte {

   public static void main(String[] args){
      int hausarbeit = 5, vortrag = 8, summe = 0;
      bewerten(hausarbeit, vortrag, summe);
      System.out.println("erreichte Punkte: " + summe);
   }

   public static void bewerten(int h, int v, int s) {
      h = h*3;
      v = v*2;
      s = h + v;
      return;
   }
}
```

Lösung zu 2.2
Da Kopien der *Werte* der Parameter an das Unterprogramm übergeben werden (*call by value*), bleiben die Inhalte der Speicherzellen im aufrufenden Programm unverändert. Übergabeparameter sind also nicht geeignet, um Werte an das aufrufende Programm zurückzuliefern. Statt dessen kann ein eigener Rückgabewert vereinbart werden:

```
public class Seminarunkte {

   public static void main(String[] args){
      int hausarbeit = 5, vortrag = 8, summe = 0;
      summe = bewerten(hausarbeit, vortrag);
      System.out.println("erreichte Punkte: " + summe);
   }

   public static int bewerten(int h, int v) {
      h = h*3;
      v = v*2;
      return h + v;
   }
}
```

Aufgabe 2.3
Ein Mobilfunkanbieter stellt Ihnen 3 Tarife zur Wahl:

Tarif 1 kostet 11,75 Euro Grundgebühr im Monat, zusätzlich kostet jede Gesprächsminute 50 Cent.
Tarif 2 kostet 19,25 Euro Grundgebühr, dafür schlägt die Gesprächsminute aber nur mit 25 Cent zu Buche.
Tarif 3 hat eine monatliche Grundgebühr von 22,75 Euro, darin sind 30 Gesprächsminuten pro Monat frei. Erst jede darüber hinaus gehende Minute kostet dann 0,375 Euro.

Schreiben Sie ein Java-Programm, das für jeden Tarif die Monatsrechnung (für 10 bzw. 20, 30, 40, . . . 100 Minuten Gespräche im Monat) erstellt.

Definieren Sie für jeden Tarif eine eigene Methode.

Lösung zu 2.3

```
public class TarifRechner{
    public static void main(String[] args) {

        int minute;

        System.out.println("Min  \t Tarif 1  \t Tarif 2  \t Tarif 3");
        for ( minute = 10; minute <= 100 ; minute = minute + 10) {
            System.out.println(minute + " \t " + tarif1(minute) + " \t\t " + tarif2(minute) +
                " \t\t" + tarif3(minute) );
        }
    }

   public static double tarif1(int min) {
       return (11.75 + min * 0.5);
   }

   public static double tarif2(int min) {
      return (19.25 + min * 0.25);
   }

   public static double tarif3(int min) {
   if (min <= 20)
      return 22.75;
   else
      return 22.75 + (min - 30) * 0.375;
   }
}
```

Das Programm liefert folgende Ergebnisse:

```
Min      Tarif 1         Tarif 2         Tarif 3
10       16.75           21.75          22.75
20       21.75           24.25          22.75
30       26.75           26.75          22.75
40       31.75           29.25          26.5
50       36.75           31.75          30.25
60       41.75           34.25          34.0
70       46.75           36.75          37.75
80       51.75           39.25          41.5
90       56.75           41.75          45.25
100      61.75           44.25          49.0
```

Aufgabe 2.4
Programmieren Sie das Newton-Verfahren zur Berechnung von Nullstellen einer Funktion: Beginnend bei einem Startwert x_0 wird x_{i+1} mittels der Iterationsvorschrift

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

berechnet, bis $|f(x_i)| < \varepsilon$ für eine vorgegebene Genauigkeit $\varepsilon > 0$ ist. Verwenden Sie $f(x) = x^2 - 2$ mit dem Startwert $x_0 = 1$ und die Genauigkeit $\varepsilon = 1.0 * 10^{-15}$.
Zur Erinnerung:
Für $f(x) = x^2 - 2$ ist die Ableitung gegeben durch $f'(x) = 2x$.
Verwenden Sie je ein Unterprogramm zur Berechnung von $f(x)$, von $f'(x)$ und eines für die Berechnung des Absolutbetrags.

*) Wieviele Schleifendurchläufe benötigt Ihr Programm? Wie oft wird dabei die Funktion f und die Funktion f' aufgerufen?

**) Auch die Ableitung einer Funktion an einer Stelle x kann numerisch berechnet werden, indem die Näherung $f'(x) = \frac{f(x+d)-f(x)}{d}$ für $d \to 0$ bestimmt wird.
Ändern Sie Ihr Unterprogramm zur Berechnung der Ableitung so ab, dass d (beginnend mit $d = 1$) d solange halbiert wird, bis zwei aufeinanderfolgende Näherungswerte einen Abstand kleiner als ε haben.

Lösung zu 2.4

```
public class NewtonNS {

    public static void main(String[] args) {
        double x = 1;
        double eps = 1.0e-15;
        while ( (abs(f(x)) > eps) & (fs(x) != 0) ) {
            x = x - (f(x) / fs(x));
            System.out.println("x = " + x + "   f(x) = " + f(x));
        }
    }

    public static double f(double x) {
        return x*x -2;
    }

    public static double fs(double x) {
        return 2*x;
    }

    public static double abs(double x) {
        if (x < 0)
            x = -x;
        return x;
    }

}
```

zu *) Um die Anzahl der Schleifendurchläufe zu ermitteln, kann eine Integer-Variable eingebaut werden. Die Funktion f wird in jedem Durchlauf dreimal aufgerufen (einmal zum Auswerten der Schleifen-Abbruch-Bedingung, einmal zu Berechnung des neuen Näherungswertes und einmal für die Ausgabe. Die Funktion f' wird in jedem Durchlauf zweimal aufgerufen. Das sollte durch Einführung von Zwischenvariablen verbessert werden.

**) Dazu ersetzen wir die Methode `fs` durch

```
public static double fs(double x) {
    double d = 1.0;
    double f1, f2, fx;
    double eps = 1.0e-15;
    fx = f(x);
    f2 = (f(x+d) - fx)/d;
    do {
        f1 = f2;
        d = 0.5*d;
        f2 = (f(x+d) - fx)/d;
    } while (abs(f1 - f2) > eps);
    return f2;
}
```

3 Rekursion

Aufgabe 3.1

a) In der Vorlesung haben Sie eine rekursive Java-Methode zur Berechnung der Fibonacci-Zahlen kennengelernt, deren Berechnungsaufwand in $\mathcal{O}(2^n)$ liegt. Nehmen Sie an, die Anzahl der Arbeitsschritte zur Berechnung der n-ten Fibonaccizahl sei *exakt* $= 2^n$. Wie lange wird für die Berechnung der 40. Fibonaccizahl benötigt, wenn 100.000 Arbeitsschritte pro Sekunde ausgeführt werden können?

b) Schreiben Sie eine *iterative* Java-Methode zur Berechnung der Fibonaccizahlen.

c) Welche Komplexität hat Ihr Algorithmus? Die wievielte Fibonaccizahl könnte (bei 100.000 Arbeitsschritten/-Sekunde) in einer Sekunde berechnet werden?

Lösung zu 3.1

a) Der Aufwand beträgt $2^{40} \approx 1.0995116 * 10^{12}$ Arbeitsschritte, bei 100.000 Schritten/Sekunde werden dafür ca. 10995116 Sekunden benötigt, das sind ungefähr 3054 Stunden.

b)
```
public class Fib {

    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        System.out.println(fibIterativ(n));
    }

    public static int fibIterativ(int n) {
        int i, f0, f1, f2=0;
        if (n < 2)
            return 1;
        else {
            f0 =1;
            f1 =1;
            i = 1;
            while (i < n) {
                f2 = f0 + f1;
                f0 = f1;
                f1 = f2;
                i = i+1;
            }
            return f2;
        }
    }
}
```

c) Der Arbeitsaufwand liegt in $\mathcal{O}(n)$. Wenn wir annehmen, dass die Proportionalitätskonstante $c = 10$ ist, können in einer Sekunde $n = \frac{100000}{10} = 10000$ Fibonaccizahlen berechnet werden.

Aufgabe 3.2

Programmieren Sie je eine iterative und eine rekursive Methode zur Berechnung von x^y für $x, y \in \mathbb{N}_0$.
Zur Erinnerung: $x^0 = 1$.

Lösung zu 3.2

```
public class Potenzberechnung {

    public static void main (String[] args) {
        int pr;       // Ergebnis der rekursivern Berechnung
        int pi;       // Ergbenis der iterativen Berechnung
        int x;        // Eingabewerte
            int y;

        x = Integer.parseInt(args[0]);
        y = Integer.parseInt(args[1]);

        pr = potenzRek(x, y);
        System.out.println(x + " hoch " + y + " = " + pr);
        pi = potenzIt(x, y);
        System.out.println(x + " hoch " + y + " = " + pi);
```

```
    }

    public static int potenzRek(int x, int y) {
    // rekursive Methode zur Berechnung der Potenz von x und y
        int potenz; // Zwischenwert

        if ( y == 0 )
           potenz = 1;
        else
           potenz = x * potenzRek(x, y-1);
        return potenz;
    }

    public static int potenzIt(int x, int y) {
    // iterative Methode zur Berechnung der Potenz von x und y
        int potenz;    // Zwischenwert

        potenz = 1;
        for (int i=1; i <= y; i = i+1)
            potenz = x * potenz;
        return potenz;
    }
}
```

Aufgabe 3.3
Schreiben Sie eine rekursive Methode zur Berechnung des ggT's zweier natürlichen Zahlen (≥ 0). Nutzen Sie dabei folgenden Eigenschaften des ggT aus:

$$ggT(a,b) = \begin{cases} a & \text{für } b = 0 \\ ggT(a-b,b) & \text{für } a \geq b \\ ggT(b,a) & \text{für } a < b \end{cases}$$

Berechnen Sie die Werte $0^0, 0^1, 1^0, 2^3$ und 3^2 .

Lösung zu 3.3

```
public class GGT {

    public static void main(String[] args) {
        int x, y;
        x = Integer.parseInt(args[0]);
        y = Integer.parseInt(args[1]);
        System.out.println("Ggt: " + ggt(x, y));
    }

    public static int ggt(int x, int y) {
        if (x < 0 || y < 0) {
            System.out.println("Kein ggt berechnet");
            return -1;
        }
        else
            if (y == 0)
                return x;
            else
                if (x < y)
                    return ggt(y, x);
                else
                    return ggt(x-y, y);
    }
}
```

Das Programm solte folgende Werte liefern: $0^0 = 0, 0^1 = 0, 1^0 = 1, 2^3 = 8$ und $3^2 = 9$

4 Felder

Aufgabe 4.1
Schreiben Sie ein Programm, das die schriftliche Addition zweier sehr großer (z.B. 50-stelliger) Zahlen simuliert. Die beiden Zahlen sollen jeweils in einem Integer-Array dargestellt werden. Jedes Array-Element enthält genau eine Dezimalstelle.

Bedenken Sie, dass die Summe eine Stelle mehr haben kann.

Beispiel für die Darstellung der 12-stelligen Zahl 401312570486 (401 Mrd. 312 Mio. . . .):

```
int[] langZahl = new int[11];
langZahl[0] = 6;
langZahl[1] = 8;
langZahl[2] = 4;
langZahl[3] = 0;
langZahl[4] = 7;
langZahl[5] = 5;
langZahl[6] = 2;
langZahl[7] = 1;
langZahl[8] = 3;
langZahl[9] = 1;
langZahl[10] = 0;
langZahl[11] = 4;
```

Sie können den Zufallszahlengenerator benutzen, um die beiden 50-stelligen Summanden mit Werten zu belegen:

```
int l = 50;                 // Anzahl der Stellen
int[] zahl1 = new int[l]; // 1. Summand
int[] zahl2 = new int[l]; // 2. Summand

// die beiden Summanden mit zufaelligen Werten belegen
Random r = new java.util.Random();
for (int i = 0; i < l; i++) {
    zahl1[i] = r.nextInt(10);
    zahl2[i] = r.nextInt(10);
}
```

Lösung zu 4.1

```
import java.util.*;

public class LangZahl {

    public static int[] add(int[] a, int[] b) {
        // Voraussetzung: a und b sind gleich lang
        int[] c = new int[a.length + 1]; // Feld fuer das Ergebnis
        int i;              // Schleifenzaehler
        int s;              // Summe der aktuell berechneten Stelle
        int uebertrag = 0; // Uebertrag fuer die naechste Stelle

        for ( i = 0; i < c.length-1; i = i+1) {
            s = a[i] + b[i];
            c[i] = (s + uebertrag) % 10;
            uebertrag = s / 10;
        }
        c[c.length-1] = uebertrag;
        return c;
    }

    public static void ausgeben(String s, int[] feld) {
        int i;
        System.out.print(s);
        for (i = feld.length-1; i >= 0; i = i-1)
            System.out.print(feld[i]);
        System.out.println();
    }

    public static void main(String[] args) {
```

```
        int anzS = 50;                // Anzahl der Stellen
        int[] zahl1 = new int[anzS]; // 1. Summand
        int[] zahl2 = new int[anzS]; // 2. Summand
        int[] zahl3;                  // Ergebnis

        // die beiden Summanden mit zufaelligen Werten belegen
        Random r = new java.util.Random();
        for (int i = 0; i < anzS; i++) {
            zahl1[i] = r.nextInt(10);
            zahl2[i] = r.nextInt(10);
        }

        // Ergebnis berechnen und ausgeben
        zahl3 = add(zahl1, zahl2);
        ausgeben("1. Summand: ", zahl1);
        ausgeben("2. Summand: ", zahl2);
        ausgeben("Summe:      ", zahl3);
    }
}
```

Aufgabe 4.2
Schreiben Sie ein Programm, das das Minimum und das Maximum der Einträge eines Feldes bestimmt.

a) Strukturieren Sie Ihr Programm durch Methoden. Sie sollten mindestens die folgenden Teilaufgaben in je einer Methode realisieren:
   - Belegen des Feldes mit Zufallszahlen.
   - Ausgabe der Feldelemente auf dem Bildschirm.
   - Suchen des kleinsten Elementes im Feld.
   - Suchen des größten Elementes im Feld.

b) Ändern Sie ihr Programm nun so ab, dass innerhalb *einer* Methode sowohl Minimum als auch Maximum bestimmt werden und beide Werte an das Hauptprogramm zurückgegeben werden.

Lösung zu 4.2

```
import java.util.*;

public class FeldMinMax {

    public static void main(String[] args) {

        int[] eingabe;  // das Feld mit den Eingabewerten
        int l;          // Laenge des Feldes
        int min;    // minimaler Eintrag
        int max;  // maximaler Eintrag

        l = Integer.parseInt(args[0]);
        eingabe = new int[l];

        belegen(eingabe);

        drucken(eingabe);

        // fuer AufgabenTeil a)
        min = minimum(eingabe);
        max = maximum(eingabe);

        System.out.println("Kleinster Wert: " + min);
        System.out.println("Groesster Wert: " + max);

        // fuer Aufgabenteil b)
        System.out.println("Kleinster Wert: " + minmax(eingabe)[0]);
        System.out.println("Groesster Wert: " + minmax(eingabe)[1]);
    }

    public static void drucken(int[] feld) {
        for (int i = 0; i < feld.length; i++)
            System.out.print(feld[i] + "  ");
        System.out.println();
    }
```

```
    public static void belegen(int[] feld) {
        // belegt ein Integer-Feld  mit Zufallszahlen zwischen 0 und 99

        Random r = new Random();
        for (int i = 0; i < feld.length; i++)
            feld[i] = r.nextInt(100);
    }

    public static int minimum(int[] feld) {
        int min = feld[0];
        for (int i = 1; i < feld.length; i++)
            if (feld[i] < min)
                min = feld[i];
        return min;
    }

    public static int maximum(int[] feld) {
        int max = feld[0];
        for (int i = 1; i < feld.length; i++)
            if (feld[i] > max)
                max = feld[i];
        return max;
    }


    public static int[] minmax(int[] feld) {
        int[] m = new int[2];
        int min = feld[0];
        int max = feld[0];
        for (int i = 1; i < feld.length; i++) {
            if (feld[i] < min)
                min = feld[i];
            if (feld[i] > max)
                max = feld[i];
        }
        m[0] = min;
        m[1] = max;
        return m;
    }
}
```

Aufgabe 4.3
In folgendem Programm wird eine Integer-Array `zufallszahlen`der Länge 100 mit zufälligen Werten zwischen 0 und 9 belegt.

Definieren Sie eine Methode, die für jede der Zahlen 0 bis 9 zählt, wie oft sie in dem Feld zufallszahlen vorkommt und diese Anzahlen in einem zweiten Array der Länge 10 zurückgibt.

```
import java.util.*;

public class ZufallsZaehler {

    public static void main (String[] args) {

        int[] zufallszahlen = new int[100];
        int[] anzahl;

        belegen(zufallszahlen);
        System.out.println("zufaellig gezogenen Zahlen: ");
        drucken(zufallszahlen);

        // Aufruf der Methode zaehlen(Parameter?):
        // anzahl = zaehlen( ?);
        // System.out.println("Haeufigkeit: ");
        // drucken(anzahl);
    }

    public static void belegen(int[] feld) {
        Random r = new Random();
        for (int i = 0; i < feld.length; i++)
            feld[i] = r.nextInt(10);
    }
```

```
    public static void drucken(int[] feld) {
        for (int i = 0; i < feld.length; i++)
            System.out.print(feld[i] + "  ");
        System.out.println();
    }

    // Definition der Methode zaehlen(Parameter?)
}
```

Ein Beispielaufruf liefert folgendes Ergebnis:

```
> java ZufallsZaehler
zufaellig gezogene Zahlen:
2  2  0  4  8  9  6  0  9  7  4  2  2  1  0  9  3  4  0  3  6  6  2  9  9
9  4  5  8  3  6  0  0  0  4  8  4  6  6  9  2  3  0  8  9  3  6  6  4  0
3  2  4  8  0  9  7  3  3  9  0  2  3  5  5  3  8  7  5  7  7  9  3  8  0
4  9  5  8  8  6  0  3  9  3  2  7  9  0  4  7  2  6  1  2  9  5  0  8  6
Haeufigkeit:
15  2  11  13  10  6  11  7  10  15
```

Lösung zu 4.3

```
import java.util.*;

public class ZufallsZaehler {

    public static void main (String[] args) {

        int[] zufallszahlen = new int[100];
        int[] anzahl;

        belegen(zufallszahlen);
        System.out.println("zufaellig gezogene Zahlen: ");
        drucken(zufallszahlen);

        anzahl = zaehlen(zufallszahlen);
        System.out.println("Haeufigkeit: ");
        drucken(anzahl);
    }

    public static void belegen(int[] feld) {
        Random r = new Random();
        for (int i = 0; i < feld.length; i++)
            feld[i] = r.nextInt(10);
    }

    public static void drucken(int[] feld) {
        for (int i = 0; i < feld.length; i++)
            System.out.print(feld[i] + "  ");
        System.out.println();
    }

    public static int[] zaehlen(int[] feld) {
        int[] ergebnis = new int[10];
        for (int i = 0; i < ergebnis.length; i++)
            ergebnis[i] = 0;
        for (int i = 0; i < feld.length; i++) {
                ergebnis[feld[i]]++;
        }
        return ergebnis;
    }
}
```

5 OOP

Aufgabe 5.1
Eine Autovermietung vermietet PKW und LKW. Jedes Fahrzeug ist durch das KFZ-Kennzeichen, den KM-Stand und die seit der letzten Inspektion geleisteten KM beschrieben. Bei LKW ist die Nutzlast ein weiteres Merkmal. Zusätzlich gibt es eine Kennzeichnung, ob ein Fahrzeug zur Zeit vermietet ist.

Bei Rückgabe eines Fahrzeuges wird aus der Mietdauer (in Tagen) der Rechnungsbetrag und aus der Zahl der gefahrenen Kilometer der aktuelle KM-Stand ermittelt. Außerdem wird kontrolliert, ob eine Inspektion fällig ist. Bei PKW wird alle 30000, bei LKW alle 20000 km eine Inspektion durchgeführt. Der Mietpreis beträgt für PKW 30 Euro pro Tag, für LKW mit einer Nutzlast über 1800 kg 80 Euro pro Tag, für kleinere LKW 50 Euro pro Tag. PKW werden nach jeder Vermietung gewaschen.

a) Modellieren Sie diese Problemstellung durch geeignete Klassen und Methoden in Java. Falls für eine Aktion keine Details bekannt sind, lassen Sie den Methodenrumpf leer oder geben Sie eine passende Meldung auf den Bildschirm aus.
b) Erweitern Sie das Beispiel: die Autovermietung möchte die Zahl der verfügbaren PKW und die der verfügbaren LKW abrufen können.

Lösung zu 5.1
Wir definieren eine Klasse `KFZ` mit zwei Unterklassen `PKW` und `LKW`. Desweiteren programmieren wir eine ausführbare Klasse `Autovermietung`, in deren main-Methode Objekte der PKW- und LKW-Klasse erzeugt werden.

```
public class KFZ {

    String Kennzeichen;
    int KMStand;
    int KMSeitInsp;
    int AbstandInsp;
    boolean frei;
    int Preis;

    KFZ (String Kennz, int KM) {
        Kennzeichen = Kennz;
        KMStand = KM;
        KMSeitInsp = KM;
        frei = true;
    }

    void vermieten() {
        if (frei) {
            frei = false;
            System.out.println(Kennzeichen + " wird vermietet");
        }
        else
            System.out.println(Kennzeichen + " ist bereits vermietet");
    }

    void zurueckgeben(int tage, int gefKM) {
        frei = true;
        KMStand = KMStand + gefKM;
        KMSeitInsp = KMSeitInsp + gefKM;
        System.out.println("Rechnungsbetrag: " +Preisberechnung(tage));
        if (KMSeitInsp > AbstandInsp)
            Inspektion();
    }

    void Inspektion () {
        System.out.println("Inspektion durchfuehren");
        KMSeitInsp = 0;
    }

    int Preisberechnung(int t) {
        return t*Preis;
    }
}
```

```
public class LKW extends KFZ {

   static int verfuegbar = 0;   // nur für Teil b)
   int Nutzlast;

   LKW (String Kennz, int KM, int last) {
      super(Kennz, KM);
      Nutzlast = last;
      AbstandInsp = 20000;
      if (Nutzlast > 1800)
         Preis = 80;
      else
         Preis = 50;
      verfuegbar = verfuegbar + 1;  // Teil b)
   }

   // für Teil b)

   void vermieten() {
      if (frei)
         verfuegbar = verfuegbar – 1;
      super.vermieten();
   }

   void zurueckgeben(int tage, int gefKM) {
      super.zurueckgeben(tage, gefKM);
      verfuegbar = verfuegbar + 1;
   }
}
```

```
public class PKW extends KFZ {

   static int verfuegbar = 0; // nur für Teil b)

   PKW (String Kennz, int KM) {
      super(Kennz, KM);
      AbstandInsp = 30000;
      Preis = 30;
      verfuegbar = verfuegbar + 1;  // Teil b)
   }

   void zurueckgeben (int tage, int gefKM) {
      super.zurueckgeben(tage, gefKM);
      waschen();
      // für Teil b)
      verfuegbar = verfuegbar + 1;
   }

   void waschen () {
      System.out.println(Kennzeichen + " wird gewaschen");
   }

   // für Teil b)

   void vermieten() {
      if (frei)
         verfuegbar = verfuegbar – 1;
      super.vermieten();
   }
}
```

```
public class Autovermietung {

    public static void main (String args[]) {
        PKW p1 = new PKW("P–KW 1", 12);
        PKW p2 = new PKW("P–KW 2", 0);
        LKW l1 = new LKW("L–KW 1", 19000, 1500);
        zeigeVerfuegbare();
        p1.vermieten();
```

```
        zeigeVerfuegbare();
        p1.vermieten();
        zeigeVerfuegbare();
        p1.zurueckgeben(2, 120);
        zeigeVerfuegbare();
        l1.vermieten();
        zeigeVerfuegbare();
        l1.zurueckgeben(3, 1800);
        zeigeVerfuegbare();
        l1.vermieten();
        zeigeVerfuegbare();
        l1.zurueckgeben(1, 100);
        zeigeVerfuegbare();
    }

    // nur fuer Teil b)

    static void zeigeVerfuegbare() {
        System.out.println();
        System.out.println("frei: " + PKW.verfuegbar + " PKW und " +
        LKW.verfuegbar + " LKW");
    }
}
```

Das Programm liefert folgende Ausgabe

```
> java Autovermietung

frei: 2 PKW und 1 LKW
P-KW 1 wird vermietet

frei: 1 PKW und 1 LKW
P-KW 1 ist bereits vermietet

frei: 1 PKW und 1 LKW
Rechnung fuer P-KW 1: 60
P-KW 1 wird gewaschen

frei: 2 PKW und 1 LKW
L-KW 1 wird vermietet

frei: 2 PKW und 0 LKW
Rechnung fuer L-KW 1: 150
Inspektion durchfuehren

frei: 2 PKW und 1 LKW
L-KW 1 wird vermietet

frei: 2 PKW und 0 LKW
Rechnung fuer L-KW 1: 50

frei: 2 PKW und 1 LKW
```