Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Programmierung 1
WiSe 25/26
Moritz Balg
Ausgabe: 17.11.2025
Bearbeiten bis: 23.11.2025

Übungsblatt 7

- Felder -

Aufgabe 1: Kommandozeilenargumente

Schreiben Sie ein Java-Programm, das den Inhalt des Übergabeparameters args (ein Feld von Strings) aus main zeilenweise auf dem Bildschirm ausgibt (pro Feldelement eine Zeile).

Mögliche Lösung:

```
/**
*Ausgabe der Kommandozeilenargumente
*@author Rudolf Berrendorf
*@version 1.0
*/
public class Kommandozeilenargumente{
  public static void main(String[] args){
    for(int i=0; i<args.length; i++){
      System.out.println("Argument " + i + ": " + args[i]) ;
    }
  }
}
```

Aufgabe 2: Feld zu String

Schreiben Sie eine Methode `public static String feldZuString(int[] feld)`. Diese nimmt ein eindimensionales Feld von ganzen Zahlen an, wandelt dieses in ein String um und gibt den String als Ergebnis zurück. Für ein Feld mit Länge n den Einträgen $a_0, a_1, ..., a_{n-1}$ soll der String die folgende Form haben: `[a[0], a[1], ..., a[n-1]]`. Der String beginnt und endet also mit einer „[„ bzw. einer „]“, die Feldelement sind mit Komma getrennt und nach einem Komma steht ein Leerzeichen.

Beispiel

Für das Feld `int[] a = {1,3,4,9}` liefert `feldZuString(a)` den String `[1, 3, 4, 9]`.

Info

Sie können diese Methode entsprechend für Felder anderer Datentypen überladen.

Mögliche Lösung:

```
public class ArrayUtil{
  /**
   * Wandelt ein int-Array in eine String-Darstellung um.
   *
   * @author mbalg2m
   * @param arr das umzuwandelnde int-Array (darf nicht leer sein)
   * @return eine String-Darstellung des Arrays im Format "[x1, x2, ..., xn]"
   *
   * @implNote
```

```
   * Diese Implementierung verwendet String-Konkatenation innerhalb einer Schleife.
   * Das ist funktional korrekt, aber nicht optimal, da Strings in Java immutable sind.
   * Bei jeder Konkatenation wird ein neues String-Objekt erzeugt, was zu erhöhter
   * Speicherbelastung und schlechterer Performance führt.
   *
   */
  public static String feldZuString(int[] arr) {
      String result = "[";
      // Achtung: Nur bis zum vorletzten Element iterieren, das letzte wird separat
angefügt,
      // um das abschließende Komma zu vermeiden.
      for (int i = 0; i < arr.length - 1; i++) {
          result += arr[i] + ", ";
      }
      result += arr[arr.length - 1] + "]";
      return result;
  }
}
```

Aufgabe 3: Felder umkehren

1. Schreiben Sie eine Methode, die ein int-Feld und einen int-Wert annimmt und zurückgibt, wieviele Einträge des Feldes gleich dem Wert sind.

   Beispiel

   Das Ergebnis zum Feld `{1,2,2,4,3,2}` und Wert 2 ist 3.

2. Schreiben Sie eine Methode, die ein int-Feld annimmt und ein neues int-Feld zurückgibt, das die Einträge des übergebenen Feldes in umgekehrter Reihenfolge enthält.

   Beispiel

   Das Ergebnis zum Feld `{1,2,2,4,3,2}` wäre das Feld `{2,3,4,2,2,1}`.

3. Schreiben Sie eine Methode, die ein int-Feld annimmt und es wieder zurückgibt, nachdem die Reihenfolge der Einträge innerhalb des Feldes umgekehrt wurde.

   Verbrauchen Sie hierzu nicht unnötig viel Speicher (legen Sie also kein neues Feld an!), sondern arbeiten Sie nur auf dem Ursprungsfeld.

   Beispiel

   Beispiel: Das Ergebnis zum Feld `{1,2,2,4,3,2}` wäre das gleiche Ursprungsfeld, allerdings mit dem Inhalt `{2,3,4,2,2,1}`.

Mögliche Lösung:

```
public class Felder {
  public static int zaehlen(int[] a, int wert) {
    int summe = 0;
    for (int i = 0; i < a.length; ++i) {
      if (a[i] == wert) {
        ++summe;
      }
    }
    return summe;
  }
```

```
  public static int[] umgekehrt(int[] a) {
    int[] ergebnis = new int[a.length];
    for (int i = 0; i < ergebnis.length; ++i) {
      ergebnis[i] = a[a.length - 1 - i];
    }
    return ergebnis;
  }

  public static int[] umkehrung (int[] a){
    for(int i=0; i<a.length/2 ; ++ i ){
      int tmp = a[i] ;
      a[i] = a[a.length - 1 - i];
      a[a.length - 1 - i] = tmp ;
    }
    return a ;
  }

  public static void main(String[] args) {
    int[] a = {3, 5, 6, 3, 7, 3, 6, 9, 1, 2, 3, 5, 1, 6};
    System.out.println(zaehlen(a, 3) + " mal ist " + 3 + " im Feld enhalten.");

    int[] b = umgekehrt(a);

    umkehrung(a);

    for(int i=0;i<a.length;++i){
      System.out.print(a[i]+" ");
      if(a[i]!=b[i]){
        System.out.println("\nFehler: Die Felder sind verschieden!");
        return;
      }
    }

  }
}
```

Aufgabe 4: Felder konkatenieren

Geben Sie eine Methode an, die für zwei int-Felder als Methodenparameter ein Ergebnisfeld als Methodenresultat liefert. Der Inhalt des Ergebnisfeldes entspricht dem Hintereinanderfügen der Inhalte der beiden Argumentfelder. Im Resultatfeld steht also zu Beginn der Inhalt des ersten Argumentfeldes und danach folgend der Inhalt des zweiten Argumentfeldes.

Beispiel

Für die beiden Felder `{1,2,3}` und `{4,5,6}` würde die Methode als Ergebnis das Feld `{1,2,3,4,5,6}` liefern.

Mögliche Lösung:

```
/**
 * Fügt zwei int-Arrays zusammen.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class FelderAnhaengen {
    public static int[] zusammenfuegen(int[] x, int[] y) {
        int[] z = new int[x.length + y.length];
```

```
        // x- Teil kopieren
        for (int i = 0; i < x.length; i++) {
            z[i] = x[i];
        }

        // y- Teil kopieren
        for (int i = 0; i < y.length; i++) {
            z[x.length + i] = y[i];
        }

        return z;
    }

    public static void main(String[] args) {
        int[] x = {1, 2, 3};
        int[] y = {4, 5, 6};

        int[] z = zusammenfuegen(x, y);

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

        System.out.println();
    }
}
```

Aufgabe 5: Felder vergleichen

1. Schreiben Sie eine Methode, die zwei gleichgroße eindimensionale Felder $a, b \in \mathbb{R}^n$ vom Java Basistyp `double` auf gleichen Inhalt untersucht und true liefert, wenn beide Felder inhaltsgleich sind bzgl. dieses Kriteriums. Die beiden Felder gelten als gleich, wenn für alle möglichen Indexwerte i für die Elemente a_i und b_i gilt:
   $|a_i - b_i| < \varepsilon$. Der Wert für ε wird der Methode neben den beiden Feldern mit übergeben.

2. Schreiben Sie eine zweite Methode, die dies analog für zweidimensionale Felder untersucht.

Mögliche Lösung:

```
/**
 * Zwei 1D Felder vergleicht.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class FelderVergleichen1D {
  public static boolean vergleichen(double[] a, double[] b, double epsilon) {
    for (int i = 0; i < a.length; i++) {
      if (Math.abs(a[i] - b[i]) > epsilon) {
        return false;
      }
    }
    return true;
  }

  public static void main(String[] args) {
    double[] a = {1, 2, 3};
    double[] b = a;
    double epsilon = 1e-6;
```

```
    System.out.println(vergleichen(a, b, epsilon));
  }
}

/**
 * Zwei 2D Felder vergleicht.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class FelderVergleichen2D {
  public static boolean vergleichen(double[][] a, double[][] b, double epsilon) {
    for (int i = 0; i < a.length; i++) {
      for (int j = 0; j < a[0].length; j++) {
        if (Math.abs(a[i][j] - b[i][j]) > epsilon) {
          return false;
        }
      }
    }
    return true;
  }

  public static void main(String[] args) {
    double[][] a = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    double[][] b = a;
    double epsilon = 1e-6;

    System.out.println(vergleichen(a, b, epsilon));
  }
}
```

Aufgabe 6: Matrix transponieren

Schreiben Sie eine Methode, die eine zweidimensionale quadratische Matrix transponiert, d.h. der Wert `a[i][j]` steht anschließend in `a[j][i]`. Diese Methode hat den Resultattyp void, da die existierende Matrix, die an die Methode übergeben wird, direkt verändert wird. Sie dürfen in der Methode keinen nennenswerten weiteren Speicherplatz anlegen, also zum Beispiel keine weitere Matrix anlegen.

Beispiel

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, A^T = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

Info

- Wie immer, überlegen Sie zuerst, bevor Sie programmieren. Führen Sie den Ansatz, den Sie sich überlegt haben, zuerst an einer kleinen Beispielmatrix (s.o.) mit Bleistift und Papier durch.
- Wie müsste die Aufgabenstellung (und die Lösung) verändert werden, wenn man auch nicht-quadratische Matrizen zulässt?

Mögliche Lösung:

```
/**
 * Matrix transponieren.
 *
 * @author Rudolf Berrendorf
```

```
 * @version 1.0
 */
public class MatrixTransponieren {
    public static void transponieren(double[][] a) {
        for (int i = 0; i < a.length; i++) {
            // Eine Haelfte mit anderer Haelfte tauschen
            for (int j = 0; j < i; j++) {
                // Dreiecke tauschen mit Spiegel-elementen der anderen Hälfte
                // (Mitte) Diagonale ist dabei Symmetrieachse
                double tmp = a[i][j];
                a[i][j] = a[j][i];
                a[j][i] = tmp;
            }
        }
    }

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

    public static void main(String[] args) {
        double[][] a = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        System.out.println("vorher: ");
        zeigeMatrix(a);
        transponieren(a);
        System.out.println("nachher: ");
        zeigeMatrix(a);
    }
}
```

Aufgabe 7: Zahlen zu Text

Entwickeln Sie eine Methode, die für eine Eingabezahl zwischen 20 und 99 den Zahltext zu dieser Zahl als Ergebnis liefert, also einen String. Zusatzaufgabe: Überlegen Sie, wie man die Zahlen kleiner als 20 und größer als 99 behandeln könnte.

Beispiel

Zu 45 würde der Text `fuenfundvierzig` geliefert.

Mögliche Lösung:

```
/**
 * Erzeugung von Zahltexten zu Zahlen.
 *
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Zahltext {
    public static String erzeugen(int zahl) {
        String[] zehnerNamen = { "", "", "zwanzig", "dreissig", "vierzig", "fünfzig",
                                 "sechzig", "siebzig", "achtzig", "neunzig" };
        String[] einerNamen = { "", "ein", "zwei", "drei", "vier", "fuenf", "sechs",
                                "sieben", "acht", "neun" };
```

```
        int zehner = zahl / 10;
        int einer = zahl % 10;

        return (einer == 0) ? zehnerNamen[zehner]
                        : einerNamen[einer] + "und" + zehnerNamen[zehner];
    }

    public static void main(String[] args) {
        for (int i = 20; i < 100; i++) {
            System.out.println("Zahl: " + i + " hat Zahltext: " + erzeugen(i));
        }
    }
}
```

Aufgabe 8: Geschlossenes Polygon 👾

Gegeben sind n Punkte im zweidimensionalen reellen Raum mit ihren Koordinaten $(x_i, y_i), i = 0, ..., n-1$, worüber ein geschlossenes Polygon definiert wird. Dies bedeutet, dass vom 0. Punkt eine Verbindung/Linie zum 1. Punkt existiert, vom 1. Punkt zum 2. Punkt usw. und vom letzten Punkt zurück zum 0. Punkt. Erfüllt ein solches Polygon bestimmte zusätzliche Eigenschaften (auf die hier nicht weiter eingegangen wird), so spricht man von einem einfachen Polygon. Zu einem einfachen Polygon kann man die durch das Polygon eingeschlossene Fläche A wie folgt berechnen:

$$A = |\frac{\sum_{i=0}^{n-1}((x_i + x_{i+1}) \cdot (y_{i+1} - y_i))}{2}|$$

Hierbei ist der Indexausdruck $i + 1$ jeweils modulo n zu sehen. Für $n = 4$ ist also der letzte Summand $((x_3 + x_0) \cdot (y_0 - y_3))$ weil $3 + 1$ *modulo* $4 = 0$ ist.

Geben Sie eine Java-Methode `public static float flaecheBerechnen(float[][] coord)` (die Methode muss genau so definiert sein!) in einer Klasse `FlaecheBerechnen` an, die für ein zweidimensionales Feld `coord` mit Punktangaben die entsprechende Fläche berechnet und als Ergebnis (Rückgabewert) der Methode liefert (nicht mit `System.out.println` auf dem Bildschirm ausgeben!). Das Feld `coord` hat für `n` Punkte `n` Zeilen und zwei Spalten. Zu jedem Punkt i existieren damit zwei Angaben: `coord[i][0]` gibt die x-Koordinate des i-ten Punktes an und `coord[i][1]` die y-Koordinate des i-ten Punktes.

Beispiel

Das Einheitsquadrat von $(0, 0)$ bis $(1, 1)$ besitzt die vier Eckpunkte $(0, 0), (1, 0), (1, 1), (0, 1)$ und hat einen Flächeninhalt von

$$\begin{aligned} A &= \frac{((0+1)\cdot(0-0)) + ((1+1)\cdot(1-0)) + ((1+0)\cdot(1-1)) + ((0+0)\cdot(0-1))}{2} \\ &= \frac{(1\cdot 0) + (2\cdot 1) + (1\cdot 0) + (0\cdot(-1))}{2} \\ &= 1 \end{aligned}$$

Aufgabe 9: Vektorwinkel 👾

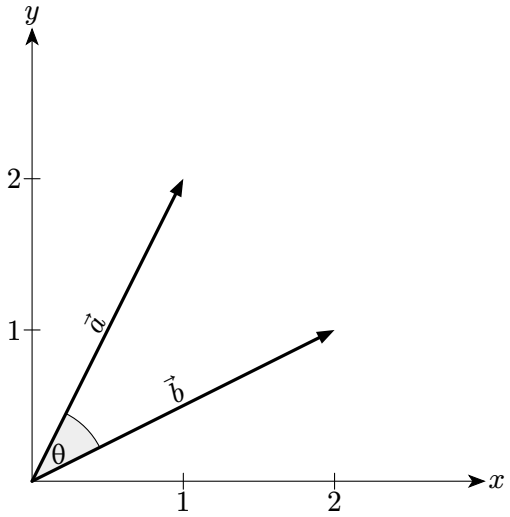
Für zwei reelle Vektoren $\vec{a} = (a_1, ..., a_n)$ und $\vec{b} = (b_1, ..., b_n) \in \mathbb{R}^n$ lässt sich der Winkel θ zwischen diesen Vektoren bestimmen über die Formel:

$$\cos(\theta) = \frac{\langle \vec{a}, \vec{b} \rangle}{\|\vec{a}\| \cdot \|\vec{b}\|}$$

wobei $\langle \vec{a}, \vec{b} \rangle = \sum_{i=1}^{n} a_i b_i$ das Skalarprodukt ist und $\|\vec{a}\| = \sqrt{\langle \vec{a}, \vec{a} \rangle}$ die Norm eines Vektors.

Ein Skalarprodukt ist also ein Wert, der die Summe über mehrere Produkte $a_i \cdot b_i$ ist. Folglich bezeichnet $\langle \vec{a}, \vec{a} \rangle$ das Skalarprodukt eines Vektors $\vec{a}$ mit sich selbst.

Siehe nachfolgende Skizze für ein Beispiel zur Aufgabenstellung mit zwei Vektoren $\vec{a} = (1, 2)$ und $\vec{b} = (2, 1)$ mit $\theta \approx 36,9\ Grad$ im zweidimensionalen Raum:



Geben Sie in einer Java-Klasse Vektorwinkel eine Java-Methode `public static double winkel(double[] a, double[] b)` an, die für zwei beliebig aber gleich große Felder `a, b`, die zwei Vektoren darstellen, den Winkel in Grad berechnet, der durch die beiden Vektoren eingeschlossen wird. Entwickeln Sie eigene Methoden zur Berechnung des Skalarprodukts und der Norm für beliebig lange Vektoren, die Sie sinnvoll zur Winkelberechnung einsetzen. Da das Ergebnis der trigonometrischen Funktion in Java in Radiant ist, nutzen Sie bei sich eine weitere Methode, die eine Umrechnung von Radiant nach Grad bewirkt. Die Umrechnungsformel für einen Winkel θ gegeben in Radiant zu einer Winkelangabe in Grad ist: $grad = \theta \cdot \frac{180}{\pi}$.

Beispiel

- Für die beiden Vektoren $(1, 0)$ und $(1, 1)$ ist der Winkel 45 Grad.
- Für die beiden Vektoren $(1, 2)$ und $(2, 1)$ ist der Winkel 36.86989764584404 Grad.
- Für die beiden Vektoren $(1, 0, 0)$ und $(0, 1, 0)$ ist der Winkel 90 Grad.
- Für die beiden Vektoren $(1, 1, 1, 1)$ und $(1, 2, 3, 4)$ ist der Winkel 24.094842552110695 Grad.

Info

- Die Arcuscosinusfunktion, die Sie benötigen (oben in der Formel steht ja $\cos(\theta) = ...$ und Sie wollen θ), lässt sich in Java über `Math.acos(x)` aufrufen, die Wurzelfunktion über `Math.sqrt(x)`, π über `Math.PI`.
- Die Wert in Java für das erste Beispiel unten ist bei einer Berechnung `45.00000000000001` (statt 45). Der Praktomat erlaubt solche Toleranzen.