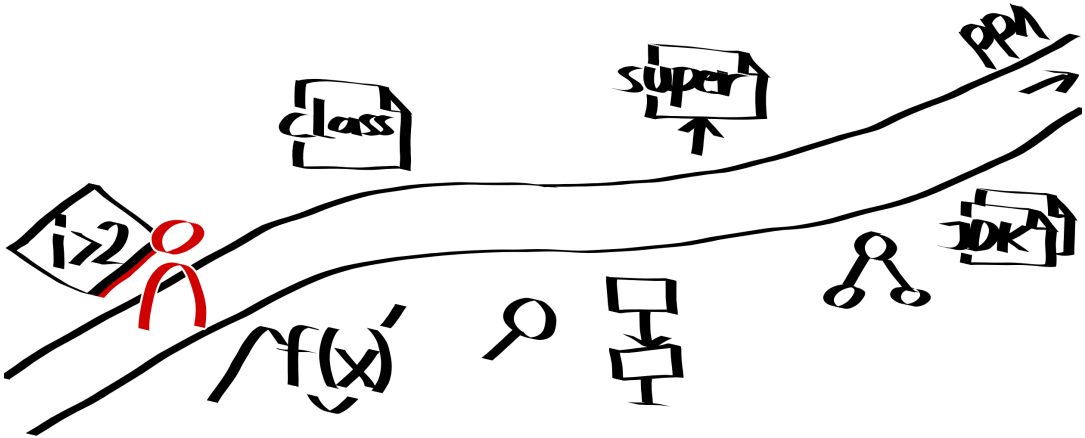
Kapitel 1: Grundlagen

Arrays

Wintersemester 2023/24

Stand 25. Oktober 2023, 06:41 Uhr

Wo stehen wir gerade?



Vorlesungsfeedback

Heinrich Heine
Universität Düsseldorf

- ø Zeit: 9 Stunden
- \+ Tempo, Schwierigkeit, auch für Anfänger:innen
- – Stream-Aussetzer
- ? keine vollen Punkte bei Random-Aufgabe (Würfel)
- ? break/continue

Arrays[1]

- Motivation: Speichern zusammengehöriger Werte vom selben Datentyp
  - Beispiele: Messwerte eines Tages, Repräsentation eines Vektors
- Analogie: Parkplatz mit durchnummerierten Stellplätzen, auf denen verschiedene Autos stehen (aber keine Fahrräder)

[1] Felder

Eigenschaften von Arrays

- Anzahl der Array-Elemente = Länge/Größe des Arrays (`.length`)
- Zugriff auf Elemente über Array-Name und Index
  - `myArray[0];`
  - ! erstes Element hat Index 0
- Größe eines Arrays im Nachhinein nicht änderbar

Index	0	1	2	3	4	5	6
Wert	6	2	4	−5	1	42	1

Länge des Arrays: 7

Deklaration, Allokation, anschließende Initialisierung

- `int[]`: Datentyp für ein Array von Integern
- `new`: Anforderung (Allokation) von neuem Speicherplatz (→ später mehr) ...
- `int[7]`: ... für 7 Integer

```
1  int[] numbers = new int[7];
2
3  numbers[0] = 6;
4  numbers[1] = 2;
5  numbers[2] = 4;
6  numbers[3] = -5;
7  numbers[4] = 1;
8  numbers[5] = 42;
9  numbers[6] = 1;
10
11 int sum = numbers[0] + numbers[1] + numbers[2] + numbers[3] + numbers[4] +
   ↪    numbers[5] + numbers[6];
```

- Standardwerte nach Allokation: `0` bzw. `false`

Deklaration, Allokation und gleichzeitige Initialisierung

- neue Bedeutung von geschweiften Klammern

```
1 int[] numbers = {6, 2, 4, -5, 1, 42, 1};
2
3 int sum = numbers[0] + numbers[1] + numbers[2] + numbers[3] + numbers[4] +
  ↪    numbers[5] + numbers[6];
```

Arrays & for-Schleifen

Heinrich Heine Universität Düsseldorf

- for-Schleifen gut geeignet, um Array-Elemente durchzugehen
- typischerweise `i` (= Index) als Zählvariable

Beispiel Java

```java
int[] numbers = {6, 2, 4, -5, 1, 42, 1};

int sum = 0;
for (int i = 0; i < numbers.length; i++) {
    sum += numbers[i];
}
```

Arrays ausgeben

- `System.out.println(numbers);` funktioniert (leider) nicht
  - macht etwas anders (→ später)
- stattdessen: Schleife benutzen

Beispiel (Fortsetzung) — Java

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

```
6, 2, 4, -5, 1, 42, 1,
```

Was läuft hier falsch?

```
1 for(int i = 0; i <= numbers.length; i++) {
2     System.out.print(numbers[i] + ", ");
3 }
```

Was läuft hier falsch?

Heinrich Heine Universität Düsseldorf

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

```
Exception java.lang.ArrayIndexOutOfBoundsException: Index 7 out of bounds for
  length 7
```

- Programmabsturz, wenn Index nicht vorhanden
  - Laufzeitfehler `ArrayIndexOutOfBoundsException`

for-each-Schleifen[2]

Heinrich Heine Universität Düsseldorf

- Abkürzung für das Durchgehen aller Array-Elemente ab Index 0
- Vermeidung von Fehlern wie `i <= numbers.length`
- `int number: numbers`: „für jeden Integer `number` in `numbers` mache Folgendes“

```
1 for (int number: numbers) {
2     System.out.print(number + ", ");
3 }
```

- Einschränkung ggü. for-Schleife:
  - nur lesender Zugriff auf Array-Elemente
  - feste Reihenfolge (beginnt immer vorne)
  - nicht brauchbar, wenn Wert von Index-Zählvariable anderweitig benötigt

[2] enhanced for loop, foreach statement

args revisited

- `args` in main-Methode auch ein Array
- von Java automatisch mit Konsolenargumenten vorbefüllt

ArgsSumProduct.java — Java

```java
public class ArgsSumProduct {
    public static void main(String[] args) {
        double[] numbers = new double[args.length];
        for(int i = 0; i < args.length; i++) {
            numbers[i] = Double.parseDouble(args[i]);
        }
        double sum = 0;
        for(double number: numbers) {
            sum += number;
        }
        double product = 1;
        for(double number: numbers) {
            product *= number;
        }
        System.out.println("Summe: " + sum + ", Produkt: " + product);
    }
}
```

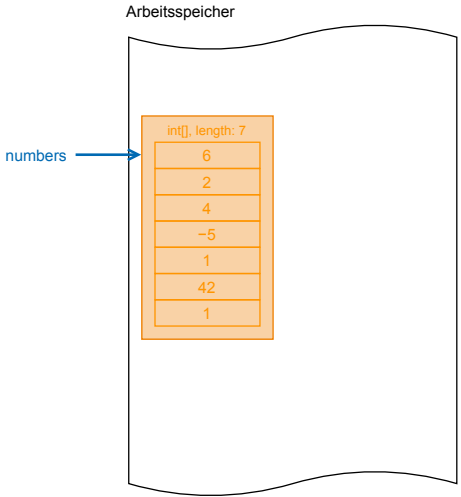
args: Beispielaufruf

- $0 + 4 + 0{,}5 + (-1) = 3{,}5$
- $1 \cdot 4 \cdot 0{,}5 \cdot (-1) = -2{,}0$

```
% java ArgsSumProduct 4 .5 -1
Summe: 3.5, Produkt: -2.0
```

Ablage im Arbeitsspeicher

- Arrays = zusammenhängendes Stück Speicher
- ermöglicht schnellen Zugriff auf Array-Elemente über Basisadresse + Index
- später: mehr zu Speicher & Zugriffszeiten



Anwendungsbeispiel: Sieb des Eratosthenes

- Ziel: Bestimmung alle Primzahlen kleiner oder gleich n.
- Primzahl: natürliche Zahl, die genau zwei Teiler hat
- Algorithmus (naive Variante):
  - Schreibe alle Zahlen von 2 bis n auf
  - Beginne bei 2 und streiche alle Vielfachen bis n
  - Wiederhole mit allen weiteren Zahlen, bis Ende der Zahlenliste erreicht

Beispiel: $n = 25$

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

Anwendungsbeispiel: Sieb des Eratosthenes

Heinrich Heine Universität Düsseldorf

- Ziel: Bestimmung alle Primzahlen kleiner oder gleich n.
- Primzahl: natürliche Zahl, die genau zwei Teiler hat
- Algorithmus (naive Variante):
  - Schreibe alle Zahlen von 2 bis n auf
  - Beginne bei 2 und streiche alle Vielfachen bis n
  - Wiederhole mit allen weiteren Zahlen, bis Ende der Zahlenliste erreicht

Beispiel: $n = 25$

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
– Vielfache von 2	2	3		5		7		9		11		13		15		17		19		21		23		25

Anwendungsbeispiel: Sieb des Eratosthenes

Heinrich Heine Universität Düsseldorf

- Ziel: Bestimmung alle Primzahlen kleiner oder gleich n.
- Primzahl: natürliche Zahl, die genau zwei Teiler hat
- Algorithmus (naive Variante):
  - Schreibe alle Zahlen von 2 bis n auf
  - Beginne bei 2 und streiche alle Vielfachen bis n
  - Wiederhole mit allen weiteren Zahlen, bis Ende der Zahlenliste erreicht

Beispiel: $n = 25$

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
– Vielfache von 2	2	3		5		7		9		11		13		15		17		19		21		23		25
– Vielfache von 3	2	3		5		7				11		13				17		19				23		25

Anwendungsbeispiel: Sieb des Eratosthenes

Heinrich Heine Universität Düsseldorf

- Ziel: Bestimmung alle Primzahlen kleiner oder gleich n.
- Primzahl: natürliche Zahl, die genau zwei Teiler hat
- Algorithmus (naive Variante):
  - Schreibe alle Zahlen von 2 bis n auf
  - Beginne bei 2 und streiche alle Vielfachen bis n
  - Wiederhole mit allen weiteren Zahlen, bis Ende der Zahlenliste erreicht

Beispiel: $n = 25$

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
– Vielfache von 2	2	3		5		7		9		11		13		15		17		19		21		23		25
– Vielfache von 3	2	3		5		7				11		13				17		19				23		25
– Vielfache von 4	2	3		5		7				11		13				17		19				23		25

Anwendungsbeispiel: Sieb des Eratosthenes

Heinrich Heine Universität Düsseldorf

- Ziel: Bestimmung alle Primzahlen kleiner oder gleich n.
- Primzahl: natürliche Zahl, die genau zwei Teiler hat
- Algorithmus (naive Variante):
  - Schreibe alle Zahlen von 2 bis n auf
  - Beginne bei 2 und streiche alle Vielfachen bis n
  - Wiederhole mit allen weiteren Zahlen, bis Ende der Zahlenliste erreicht

Beispiel: $n = 25$

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
– Vielfache von 2	2	3		5		7		9		11		13		15		17		19		21		23		25
– Vielfache von 3	2	3		5		7				11		13				17		19				23		25
– Vielfache von 4	2	3		5		7				11		13				17		19				23		25
– Vielfache von 5	2	3		5		7				11		13				17		19				23		

Anwendungsbeispiel: Sieb des Eratosthenes

- Ziel: Bestimmung alle Primzahlen kleiner oder gleich n.
- Primzahl: natürliche Zahl, die genau zwei Teiler hat
- Algorithmus (naive Variante):
  - Schreibe alle Zahlen von 2 bis n auf
  - Beginne bei 2 und streiche alle Vielfachen bis n
  - Wiederhole mit allen weiteren Zahlen, bis Ende der Zahlenliste erreicht

Beispiel: $n = 25$

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
– Vielfache von 2	2	3		5		7		9		11		13		15		17		19		21		23		25
– Vielfache von 3	2	3		5		7				11		13				17		19				23		25
– Vielfache von 4	2	3		5		7				11		13				17		19				23		25
– Vielfache von 5	2	3		5		7				11		13				17		19				23		
– Vielfache von 6	2	3		5		7				11		13				17		19				23		

Umsetzung in Java

Array-Index	0	1	2	3	4	5	6	7	8	9
`gestrichen?`	`true`	`true`	`false`	`false`	`false`	`false`	`false`	`false`	`false`	`false`

Umsetzung in Java

Array-Index	0	1	2	3	4	5	6	7	8	9
`gestrichen`?	`true`	`true`	`false`	`false`	`false`	`false`	`false`	`false`	`false`	`false`
– Vielfache von 2	`true`	`true`	`false`	`false`	`true`	`false`	`true`	`false`	`true`	`false`

Umsetzung in Java

Array-Index	0	1	2	3	4	5	6	7	8	9
`gestrichen`?	`true`	`true`	`false`	`false`	`false`	`false`	`false`	`false`	`false`	`false`
– Vielfache von 2	`true`	`true`	`false`	`false`	`true`	`false`	`true`	`false`	`true`	`false`
– Vielfache von 3	`true`	`true`	`false`	`false`	`true`	`false`	`true`	`false`	`true`	`true`

Umsetzung in Java

Array-Index	0	1	2	3	4	5	6	7	8	9
`gestrichen`?	`true`	`true`	`false`	`false`	`false`	`false`	`false`	`false`	`false`	`false`
– Vielfache von 2	`true`	`true`	`false`	`false`	`true`	`false`	`true`	`false`	`true`	`false`
– Vielfache von 3	`true`	`true`	`false`	`false`	`true`	`false`	`true`	`false`	`true`	`true`
– Vielfache von 4	`true`	`true`	`false`	`false`	`true`	`false`	`true`	`false`	`true`	`true`

Umsetzung in Java

Array-Index	0	1	2	3	4	5	6	7	8	9
`gestrichen`?	`true`	`true`	`false`	`false`	`false`	`false`	`false`	`false`	`false`	`false`
– Vielfache von 2	`true`	`true`	`false`	`false`	`true`	`false`	`true`	`false`	`true`	`false`
– Vielfache von 3	`true`	`true`	`false`	`false`	`true`	`false`	`true`	`false`	`true`	`true`
– Vielfache von 4	`true`	`true`	`false`	`false`	`true`	`false`	`true`	`false`	`true`	`true`
...										

Anwendungsbeispiel: Matrix mit Faktor multiplizieren

Heinrich Heine Universität Düsseldorf

$$c \cdot \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{pmatrix} = \begin{pmatrix} c \cdot a_{1,1} & c \cdot a_{1,2} & c \cdot a_{1,3} \\ c \cdot a_{2,1} & c \cdot a_{2,2} & c \cdot a_{2,3} \end{pmatrix}$$

mit $c, a_{i,j} \in \mathbb{R}$

Anwendungsbeispiel: Matrix mit Faktor multiplizieren

$$c \cdot \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{pmatrix} = \begin{pmatrix} c \cdot a_{1,1} & c \cdot a_{1,2} & c \cdot a_{1,3} \\ c \cdot a_{2,1} & c \cdot a_{2,2} & c \cdot a_{2,3} \end{pmatrix}$$

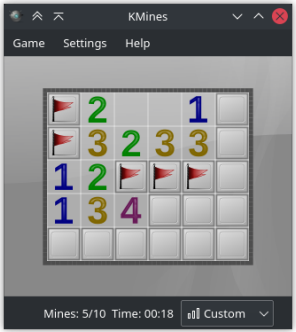
mit $c, a_{i,j} \in \mathbb{R}$

$$2 \cdot \begin{pmatrix} 3 & 7 & -5 \\ 2{,}5 & 5 & 10 \end{pmatrix} = \begin{pmatrix} 6 & 14 & -10 \\ 5 & 10 & 20 \end{pmatrix}$$

$\Rightarrow$ Wir brauchen etwas, um zweidimendionale Dinge in Java zu repräsentieren

Mehrdimensionale Arrays

- Motivation: Repräsentation mehrdimensionaler Objekte
  - Messwerte eines Tages verschiedener Messstationen
  - Repräsentation von Matrizen, Tensoren
  - Spielfelder
  - Bilder
- entsprechen in Java Arrays von Arrays
- theoretisch beliebig viele Dimensionen möglich
- Zugriff auf Elemente über Array-Name und mehrere Indices
  - `myArray[0][1]`: Zugriff auf Index 1 im ersten Sub-Array („Zeile 0, Spalte 1“)



Beispiel 2D-Array: Direkte Allokation & Initalisierung

Heinrich Heine Universität Düsseldorf

Beispiel — Java

```java
int[][] matrix = {
    {12, 54, 86, 12},
    {-2, -3, 65, 74},
    {93, 23, 73, 92},
};

System.out.println(matrix[1][2]);
```

```
65
```

	Spalte 2			
	12	54	86	12
Zeile 1	−2	−3	**65**	74
	93	23	73	92

Beispiel: Matrix mit Zufallszahlen initialisieren

Heinrich Heine Universität Düsseldorf

Beispiel (Fortsetzung) — Java

```java
int rows = 3;
int columns = 4;
double[][] matrix = new double[rows][columns];

for(int rowIndex = 0; rowIndex < matrix.length; rowIndex++) {
    for(int columnIndex = 0; columnIndex < matrix[rowIndex].length; columnIndex++) {
        matrix[rowIndex][columnIndex] = Math.random();
    }
}

// gib Element an Zeile 2, Index 3 aus (Zählung beginnt bei 0)
System.out.println(matrix[2][3]);
```

Beispiel: Matrix-Ausgabe mit for-Schleife

Heinrich Heine Universität Düsseldorf

Beispiel (Fortsetzung) — Java

```java
for(int rowIndex = 0; rowIndex < matrix.length; rowIndex++) {
    for(int columnIndex = 0; columnIndex < matrix[rowIndex].length; columnIndex++) {
        System.out.print(matrix[rowIndex][columnIndex]);
        System.out.print(" ");
    }
    System.out.println();
}
```

```
0.5977491359561252 0.9718878446233457 0.31861362776283875 0.9481385983980317
0.00904006927255252 0.9883123404476388 0.22553115582825878 0.47582204499289316
0.3769222209809959 0.41246601323075904 0.23858132948348354 0.3452413485892122
```

Beispiel: Matrix-Ausgabe mit for-each-Schleife

Heinrich Heine Universität Düsseldorf

- übersichtlicher: uns interessieren Zählvariablen eigentlich nicht
- weniger fehleranfällig: falsche Abbruchbedingung, Zeilen/Spalten vertauschen, . . .

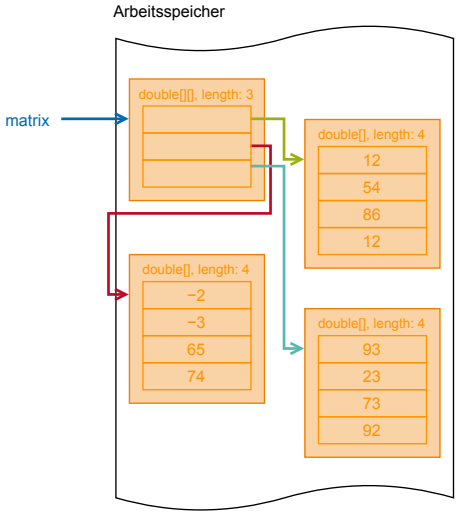
Beispiel (Fortsetzung) — Java

```java
for(double[] row: matrix) {
    for(double element: row) {
        System.out.print(element);
        System.out.print(" ");
    }
    System.out.println();
}
```

```
0.5977491359561252 0.9718878446233457 0.31861362776283875 0.9481385983980317
0.009040069272552 0.9883123404476388 0.22553115582825878 0.47582204499289316
0.3769222209809959 0.41246601323075904 0.23858132948348354 0.3452413485892122
```

2D-Arrays: Ablage im Arbeitsspeicher

- In Java: Mehrdimensionales Array = Array von Arrays



2D-Arrays müssen nicht rechteckig sein!

Heinrich Heine Universität Düsseldorf

- eher selten, aber möglich

```java
// allokiere Speicherplatz für Referenzen auf 3 Integer-Arrays
int[][] weiredArray = new int[3][];

// Abkürzung: Allokation + Initalisierung
weiredArray[0] = new int[]{12, 54};
weiredArray[1] = new int[]{12, 54, 86, 12};
weiredArray[2] = new int[]{12, 54, 86};

System.out.println(weiredArray[0].length);
System.out.println(weiredArray[1].length);
System.out.println(weiredArray[2].length);
```

```
2
4
3
```

Anmerkungen zur Syntax

- eckige Klammern dürfen auch hinter Variablennamen stehen
  → verwirrend, wenn mehrere Variablen auf einmal deklariert werden

```
1  // verwirrend:
2  int a[], b; // Integer-Array a, Integer b
3
4  // noch verwirrender:
5  int[] c[], d; // 2D-Integer-Array c, Integer-Array d
```

- bei mehrdimensionalen Array-Zugriffen darf auch geklammert werden (untypisch)

```
7  int[][] e = new int[3][4];
8  // untypische Notation:
9  System.out.println((e[2])[3]); // entspricht e[2][3]
10
```

Lernziele

Heinrich Heine
Universität Düsseldorf

Sie können am Ende der Woche ...

- (mehrdimensionale) Arrays **anlegen**.
- (mehrdimensionale) Arrays in Berechnungen **verwenden**.
- for-(each-)Schleifen **schreiben**, um auf die Elemente eines Arrays zuzugreifen.

Wichtige Begriffe

Heinrich Heine
Universität Düsseldorf

Array	length	[0]
new int[10]	{1,2,3}	for-each