

Arrays und Schleifen

Javakurs 2013, 2. Vorlesung

Theresa Enhardt

basierend auf der Vorlage von
Mario Bodemann und Sebastian Dyroff

`wiki.freitagrunde.org`

4. März 2013



This work is licensed under the *Creative Commons Attribution-ShareAlike 3.0 License*.

- 1 Arrays
 - Was ist ein Array?
 - Array-Bauanleitung
 - Mehrdimensionale Arrays
 - Fehlerquellen

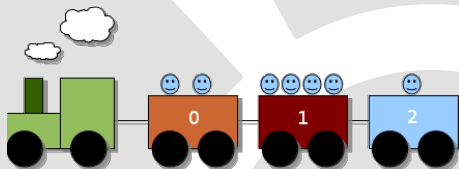
- 2 Schleifen
 - Warum Schleifen?
 - Die while-Schleife
 - Die for-Schleife
 - Fehlerquellen



4!

Was ist ein Array?

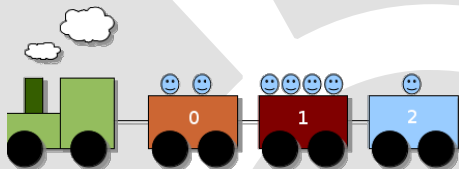
- **Beispiel:** Zug mit mehreren Wagons
Variable, die speichert, wie viele Leute in den Wagons sitzen



4!

Was ist ein Array?

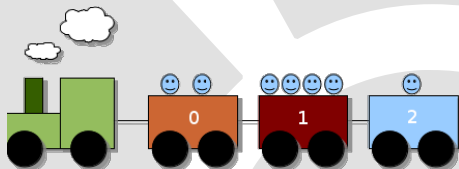
- **Beispiel:** Zug mit mehreren Wagons
Variable, die speichert, wie viele Leute in den Wagons sitzen



- Es geht besser als `zug0 = 2; zug1 = 4; zug2 = 2;`

Was ist ein Array?

- **Beispiel:** Zug mit mehreren Wagons
Variable, die speichert, wie viele Leute in den Wagons sitzen



- Es geht besser als `zug0 = 2; zug1 = 4; zug2 = 2;`
- **Array** von `int`:
Variable, die aus mehreren Zahlen "besteht", die automatisch durchnummeriert werden
- **Achtung:** Java beginnt bei der Nummerierung bei 0!

1 Arrays

- Was ist ein Array?
- **Array-Bauanleitung**
- Mehrdimensionale Arrays
- Fehlerquellen

2 Schleifen

- Warum Schleifen?
- Die while-Schleife
- Die for-Schleife
- Fehlerquellen



4!

Array-Bauanleitung

Arraysymbol

Typ

Ein neues Array erstellen

```
int [] zug = new int [3];
```

Variablenname

Länge

4!

Array-Bauanleitung

Arraysymbol

Typ

Ein neues Array erstellen

```
int [] zug = new int [3];
```

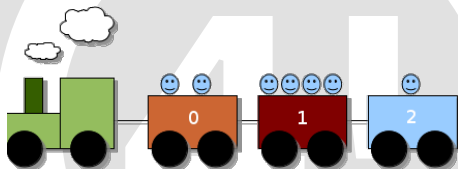
Variablenname

Länge

Mit Werten füllen

```
zug [0] = 2;  
zug [1] = 4;  
zug [2] = 1;
```

Index



Arrays mit anderen Datentypen

```
boolean [] istWagenvoll = new boolean [3];  
istWagenvoll [0] = false;  
istWagenvoll [1] = true;  
istWagenvoll [2] = false;
```

```
double wagenTemperatur [] = new double [3];  
wagenTemperatur [0] = 19.5;  
wagenTemperatur [1] = 20.2;  
wagenTemperatur [2] = 21.0;
```

- Datentyp kann durch jeden beliebigen ersetzt werden
- Werte müssen dann natürlich dazu passen
- Klammern (Arraysymbol) können überall stehen

Array mit anderer Länge

```
int zug2 [] = new int [6];
zug2 [0] = 5;
zug2 [1] = 23;
zug2 [2] = 9;
zug2 [3] = 0;
zug2 [4] = 42;
zug2 [5] = 7;

int zugLaenge = zug.length; // 3
int zug2Laenge = zug2.length; // 6
```

- Längere Arrays mit mehr Werten müssen so angelegt werden
- Sonst tritt Fehler auf (dazu später mehr)
- Array „merkt sich“, wie lang es ist
- Länge kann mit `arrayname.length` abgerufen werden

Alles in einer Zeile

```
String [] planeten = {"Merkur", "Venus", "Erde", "Mars"};
```

- Aufzählung der Werte in geschweiften Klammern
- Länge des Arrays ist damit 4, wird automatisch erkannt

4!

Deklaration, Initialisierung und Definition in drei Zeilen

```
String mehrPlaneten []; // Deklaration  
mehrPlaneten = new String [4]; //Initialisierung  
mehrPlaneten [0] = "Jupiter"; // Definition
```

- **Deklaration** = „Diese Variable existiert.“ (Inhalt noch unbekannt)
- **Initialisierung** = „Die Variable hat diese Länge.“
(Wichtig für Speicherverwaltung)
- **Definition** = „Die Variable hat diesen Inhalt.“

1 Arrays

- Was ist ein Array?
- Array-Bauanleitung
- **Mehrdimensionale Arrays**
- Fehlerquellen

2 Schleifen

- Warum Schleifen?
- Die while-Schleife
- Die for-Schleife
- Fehlerquellen



4!

Mehrdimensionale Arrays

Wie würde man eine Matrix realisieren?

$$\begin{array}{c} 0 \quad 1 \quad 2 \quad 3 \\ 0 \left(\begin{array}{cccc} 46 & 795 & 13 & 468 \\ 1 \left(\begin{array}{cccc} 965 & 648 & 5 & 60 \\ 2 \left(\begin{array}{cccc} 67 & 464 & 84 & 541 \end{array} \right) \end{array} \right) \end{array} \right)$$

Ansatz: Ein Array, in dem Arrays enthalten sind...

Mehrdimensionale Arrays

Erzeugen einer Matrix

```
int matrix [][] = new int [3][4];
```

- Mehrere Arrayklammern hintereinander → Mehrere Dimensionen
- Hier: 3 Zeilen, 4 Spalten
- Jeder Wert hat nun 2 Indizes anstatt einem

Zuweisung der Werte

```
matrix [0][0] = 46;  
matrix [0][1] = 795;  
matrix [0][2] = 13;  
matrix [0][3] = 468;  
  
matrix [1][0] = 965;  
matrix [1][1] = 648;
```

...

	0	1	2	3
0	46	795	13	468
1	965	648	5	60
2	67	464	84	541

Erzeugen einer Matrix: zeilenweise

```
int matrix [][] = new int [3][4];
int zeile0 [] = { 46, 795, 13, 468 };
int zeile1 [] = { 965, 648, 5, 60 };
int zeile2 [] = { 67, 464, 84, 541 };

matrix [0] = zeile0 ;
matrix [1] = zeile1 ;
matrix [2] = zeile2 ;
```

- Zuerst: Leere Matrix mit 3 Zeilen und 4 Spalten
- Dann: Neues Hilfsarray `zeile0` der Länge 4
- Weitere Arrays `zeile1` und `zeile2` analog dazu
- Schließlich: Zuweisen als 0., 1. und 2. Zeile der Matrix

Erzeugen einer Matrix mit nur einer Zuweisung

```
int matrix [][ ] = { { 46, 795, 13, 468 },  
                    { 965, 648, 5, 60 },  
                    { 67, 464, 84, 541 } };
```

- Variante mit der wenigsten Schreibarbeit
- Das kann alles auf eine Zeile (Umbrüche nur der Lesbarkeit halber)
- Dimensionen (3 Zeilen, 4 Spalten) implizit gegeben, d.h. automatisch erkannt
- Zuweisung sieht ursprünglicher Matrix sehr ähnlich

1 Arrays

- Was ist ein Array?
- Array-Bauanleitung
- Mehrdimensionale Arrays
- Fehlerquellen

2 Schleifen

- Warum Schleifen?
- Die while-Schleife
- Die for-Schleife
- Fehlerquellen



4!

Wo liegt der Fehler?

```
1 public class ArrayFehler1 {  
2     public static void main(String [] args) {  
3  
4         int [] zug;  
5         zug[0] = 2;  
6         zug[1] = 4;  
7         zug[2] = 1;  
8     }  
9 }
```

4!

Wo liegt der Fehler?

```
1 public class ArrayFehler1 {  
2     public static void main(String [] args) {  
3  
4         int [] zug;  
5         zug[0] = 2;  
6         zug[1] = 4;  
7         zug[2] = 1;  
8     }  
9 }
```

Compilerfehler

```
$ javac ArrayFehler1.java
```

```
ArrayFehler1.java:5: variable zug might not have been  
                                                                    initialized
```

```
zug[0] = 2;
```

```
^
```

```
1 error
```

Lösung I: Array initialisieren

Fehler behoben

```
1 public class ArrayFehler1 {  
2     public static void main(String [] args) {  
3  
4         int [] zug = new int[3];  
5         zug[0] = 2;  
6         zug[1] = 4;  
7         zug[2] = 1;  
8     }  
9 }
```

Nicht vergessen, das Array zu initialisieren!

Wo liegt der Fehler?

```
1 public class ArrayFehler2 {  
2     public static void main(String[] args) {  
3  
4         int [] zug = new int [3];  
5         zug[1] = 4;  
6         zug[2] = 1;  
7         zug[3] = 5;  
8     }  
9 }
```

4!

Wo liegt der Fehler?

```
1 public class ArrayFehler2 {  
2     public static void main(String [] args) {  
3  
4         int [] zug = new int [3];  
5         zug[1] = 4;  
6         zug[2] = 1;  
7         zug[3] = 5;  
8     }  
9 }
```

Kein Compilerfehler

```
$ javac ArrayFehler2.java
```

```
$
```

Wo liegt der Fehler?

```
1 public class ArrayFehler2 {  
2     public static void main(String [] args) {  
3  
4         int [] zug = new int [3];  
5         zug[1] = 4;  
6         zug[2] = 1;  
7         zug[3] = 5;  
8     }  
9 }
```

Kein Compilerfehler

```
$ javac ArrayFehler2.java  
$
```

Aber: Laufzeitfehler beim Ausführen

```
$ java ArrayFehler2  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3  
at ArrayFehler2.main(ArrayFehler2.java:7)  
$
```


Lösung II: Richtige Indizes benutzen

Indizes waren falsch

```
1 public class ArrayFehler2 {  
2     public static void main(String [] args) {  
3  
4         int [] zug = new int [3];  
5         zug[0] = 4;  
6         zug[1] = 1;  
7         zug[2] = 5;  
8     }  
9 }
```

Denkt dran: Zählung beginnt bei 0!

Tipp zu Lösung II: Auf Arraygrenze testen

Fehler kann nicht mehr auftreten

```
int index = 3;
if (index < zug.length) {
    zug[index] = 5;
}
```

- Anzahl der Elemente: `arrayname.length`
 - Zählung beginnt bei 0
- Index muss immer kleiner als Länge sein
- Bei der Zuweisung testen, ob wir innerhalb der Arraygrenzen sind
- Keine `ArrayIndexOutOfBoundsException` mehr.

Zusammenfassung: Arrays

- Array = Reihe von Werten mit gemeinsamem Namen + Index

Deklaration: `typ name [];`

Initialisierung: `name = new typ [länge];`

Definition: `name [index] = wert;`

Zugriff: `name [index]`

- Deklaration und Definition auch in einer Zeile möglich:

`typ name [] = { wert, wert, wert, ... };`

- Länge eines Arrays: `name.length`
- Aufpassen: Java fängt bei 0 an zu zählen!

- 1 Arrays
 - Was ist ein Array?
 - Array-Bauanleitung
 - Mehrdimensionale Arrays
 - Fehlerquellen

- 2 Schleifen
 - Warum Schleifen?
 - Die while-Schleife
 - Die for-Schleife
 - Fehlerquellen



4!

Der Anfang einer Schleife



Der Anfang einer Schleife

Aufgabe:

Schreibe ein Javaprogramm, das 5,4,3,2,1 auf der Konsole ausgibt.

```
$ java Countdown5
5
4
3
2
1
Los!
$
```

Countdown von 5 abwärts...

Der Anfang einer Schleife

Der triviale Ansatz

```
1 public class Countdown5 {  
2     public static void main(String [] args) {  
3  
4         System.out.println("5");  
5         System.out.println("4");  
6         System.out.println("3");  
7         System.out.println("2");  
8         System.out.println("1");  
9         System.out.println("Los!");  
10  
11     }  
12 }
```

Der Anfang einer Schleife

Aufgabe:

Nun schreibe einen Countdown von 1000 abwärts zu 0.

Sehr schmerzhaft...

```
1 public class Countdown1000 {
2     public static void main(String [] args
3         ) {
4         System.out.println(" 1000");
5         System.out.println(" 999");
6         System.out.println(" 998");
7         System.out.println(" 997");
8         System.out.println(" 996");
9         System.out.println(" 995");
10        System.out.println(" 994");
11        System.out.println(" 993");
12        System.out.println(" 992");
```

- Unnötige Schreibarbeit
- Copy & Paste
- Sehr fehleranfällig
- ...

...
Das muss doch besser gehen!

- 1 Arrays
 - Was ist ein Array?
 - Array-Bauanleitung
 - Mehrdimensionale Arrays
 - Fehlerquellen

- 2 Schleifen
 - Warum Schleifen?
 - **Die while-Schleife**
 - Die for-Schleife
 - Fehlerquellen



4!

Die while-Schleife: Countdown

Countdown mit einer Variable

```
1 public class Countdown {  
2     public static void main(String [] args) {  
3  
4         int counter = 1000;  
5  
6  
7         System.out.println(counter);  
8         counter = counter - 1;  
9  
10  
11     }  
12 }
```

Die while-Schleife: Countdown

Countdown mit einer Variable

```
1 public class Countdown {
2     public static void main(String [] args) {
3
4         int counter = 1000;
5
6         while (counter > 0) {
7             System.out.println(counter);
8             counter = counter - 1;
9         }
10        System.out.println("Los!");
11    }
12 }
```

- Wiederhole Zeile 7 und 8, solange Bedingung hinter while wahr ist
- counter zählt bei jedem Durchlauf um 1 runter

Die while-Schleife: Countdown

Countdown mit einer Variable

```
1 public class Countdown {
2     public static void main(String [] args) {
3
4         int counter = 1000;
5
6         while (counter > 0) {
7             System.out.println(counter);
8             counter = counter - 1;
9         }
10        System.out.println("Los!");
11    }
12 }
```

- Wiederhole Zeile 7 und 8, solange Bedingung hinter while wahr ist
- counter zählt bei jedem Durchlauf um 1 runter
- Wenn counter == 0 ist, ist Bedingung counter > 0 nicht mehr wahr und die Schleife ist zu Ende → Letzte Ausgabe ist 1

Die while-Schleife: Fakultät berechnen

Aufgabe:

Berechne die Fakultät einer Zahl! (Beispiel: 4)

$$n! = 1 * 2 * \dots * (n - 2) * (n - 1) * n$$

$$4! = 1 * 2 * 3 * 4$$

Hochzählen von 1 bis zur Zahl...

```
int zaehler = 1;
int zahl = 4;

while ( zaehler <= zahl ) {
    zaehler = zaehler + 1;
}
```

Die while-Schleife: Fakultät berechnen

Berechnen der Fakultät

```
1 public class Fakultaet {
2     public static void main(String [] args) {
3
4         int zaehler = 1;
5         int zahl = 4;
6         int ergebnis = 1; //Denn schon 0! = 1
7
8         while (zaehler <= zahl) {
9             ergebnis = ergebnis * zaehler;
10            zaehler = zaehler + 1;
11        }
12        System.out.println("Ergebnis: " + ergebnis);
13    }
14 }
```

- Erster Durchlauf: $\text{ergebnis} = 1 * 1 (= 1)$
- Zweiter Durchlauf: $\text{ergebnis} = 1 * 2 (= 2)$
- Dritter Durchlauf: $\text{ergebnis} = 2 * 3 (= 6)$
- Vierter Durchlauf: $\text{ergebnis} = 6 * 4 (= 24)$
- Schluss: Ergebnis: 24

- 1 Arrays
 - Was ist ein Array?
 - Array-Bauanleitung
 - Mehrdimensionale Arrays
 - Fehlerquellen

- 2 Schleifen
 - Warum Schleifen?
 - Die while-Schleife
 - **Die for-Schleife**
 - Fehlerquellen



4!

Die for-Schleife

Initialisierung

Bedingung

Herunterzählen

Countdown von 1000 abwärts mit for-Schleife

```
for (int counter = 1000; counter > 0; counter--) {  
    System.out.println(counter);  
}  
System.out.println("Los!");
```

4!

Die for-Schleife

Initialisierung

Bedingung

Herunterzählen

Countdown von 1000 abwärts mit for-Schleife

```
for (int counter = 1000; counter > 0; counter--) {  
    System.out.println(counter);  
}  
System.out.println("Los!");
```

- Kopf der Schleife besteht aus 3 Komponenten:
 - Zählvariable `counter` wird dort initialisiert
 - Abbruchbedingung wird festgelegt
 - Hoch-/Runterzählen der Laufvariable bei jedem Durchlauf
- Statt `counter = counter - 1` kann man auch `counter--` schreiben

for-Schleife

```
1 public class Gedicht {
2     public static void main(String[] args) {
3
4         String[] gedicht = new String[4];
5
6         gedicht[0] = "Ein Mops kam in die Kueche";
7         gedicht[1] = "und stahl dem Koch ein Ei.";
8         gedicht[2] = "Da nahm der Koch die Kelle";
9         gedicht[3] = "und schlug den Mops zu Brei.";
10
11         for (int i = 0; i < gedicht.length; i++) {
12             System.out.println( gedicht[i] );
13         }
14     }
15 }
```

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (\dots, \dots, \dots, \dots)$$

$$max_{temp} = 0$$

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (\dots, \dots, \dots, \dots)$$

$$max_{temp} = 46$$

- Aktuelle Spalte: 0
- Aktuelle Zeile: 0
- Aktueller Wert:
46
- Vergleiche mit
bisherigem
Maximum 0:
46 > 0
- Neues temporäres
Maximum
gefunden: 46

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (\dots, \dots, \dots, \dots)$$

$$max_{temp} = 965$$

- Aktuelle Spalte: 0
- Aktuelle Zeile: 1
- Aktueller Wert: 965
- Vergleiche mit bisherigem Maximum 46: $965 > 46$
- Neues temporäres Maximum gefunden: 965

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (\dots, \dots, \dots, \dots)$$

$$max_{temp} = 965$$

- Aktuelle Spalte: 0
- Aktuelle Zeile: 2
- Aktueller Wert: 67
- Vergleiche mit bisherigem Maximum 965: $67 < 965$
- Kein neues temporäres Maximum, bleibt 965

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (965, \dots, \dots, \dots)$$

$$max_{temp} = 0$$

- Ende der Spalte erreicht
- Keine weiteren Kandidaten: Temporäres Maximum ist damit endgültig
- Vorbereitung für nächste Spalte: Setze temporäres Maximum wieder auf 0

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (965, \dots, \dots, \dots)$$

$$max_{temp} = 795$$

- Aktuelle Spalte: 1
- Aktuelle Zeile: 0
- Aktueller Wert: 795
- Vergleiche mit bisherigem Maximum 0: $795 > 0$
- Neues temporäres Maximum gefunden: 795

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (965, \dots, \dots, \dots)$$

$$max_{temp} = 795$$

- Aktuelle Spalte: 1
- Aktuelle Zeile: 1
- Aktueller Wert: 648
- Vergleiche mit bisherigem Maximum 795: $648 < 795$
- Kein neues Maximum, bleibt 795

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (965, \dots, \dots, \dots)$$

$$max_{temp} = 795$$

- Aktuelle Spalte: 1
- Aktuelle Zeile: 2
- Aktueller Wert: 464
- Vergleiche mit bisherigem Maximum 795: $464 < 795$
- Kein neues Maximum, bleibt 795

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (965, 795, \dots, \dots)$$

$$max_{temp} = 0$$

- Ende der Spalte erreicht
- Keine weiteren Kandidaten: Temporäres Maximum ist damit endgültig
- Vorbereitung für nächste Spalte: Setze temporäres Maximum wieder auf 0

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (965, 795, \dots, \dots)$$

$$max_{temp} = 13$$

- Aktuelle Spalte: 2
- Aktuelle Zeile: 0
- Aktueller Wert: 13
- Vergleiche mit bisherigem Maximum 0: $13 > 0$
- Neues temporäres Maximum gefunden: 13

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (965, 795, \dots, \dots)$$

$$max_{temp} = 13$$

- Aktuelle Spalte: 2
- Aktuelle Zeile: 1
- Aktueller Wert: 5
- Vergleiche mit bisherigem Maximum 13:
5 < 13
- Kein neues Maximum, bleibt 13

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (965, 795, \dots, \dots)$$

$$max_{temp} = 84$$

- Aktuelle Spalte: 2
- Aktuelle Zeile: 2
- Aktueller Wert:
84
- Vergleiche mit
bisherigem
Maximum 13:
84 > 13
- Neues temporäres
Maximum
gefunden: 84

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (965, 795, 84, \dots)$$

$$max_{temp} = 0$$

- Ende der Spalte erreicht
- Keine weiteren Kandidaten: Temporäres Maximum ist damit endgültig
- Vorbereitung für nächste Spalte: Setze temporäres Maximum wieder auf 0

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (965, 795, 84, \dots)$$

$$max_{temp} = 468$$

- Aktuelle Spalte: 3
- Aktuelle Zeile: 0
- Aktueller Wert: 468
- Vergleiche mit bisherigem Maximum 0: $468 > 0$
- Neues temporäres Maximum gefunden: 468

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (965, 795, 84, \dots)$$

$$max_{temp} = 468$$

- Aktuelle Spalte: 3
- Aktuelle Zeile: 1
- Aktueller Wert: 60
- Vergleiche mit bisherigem Maximum 468: $60 < 468$
- Kein neues Maximum, bleibt 468

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (965, 795, 84, \dots)$$

$$max_{temp} = 541$$

- Aktuelle Spalte: 3
- Aktuelle Zeile: 2
- Aktueller Wert: 541
- Vergleiche mit bisherigem Maximum 468: $541 > 468$
- Neues temporäres Maximum gefunden: 541

for-Schleife: Matrix-Beispiel

Beispiel: Finde das Maximum aus jeder Spalte und speichere die Ergebnisse in einem neuen Array!

$$matrix = \begin{pmatrix} 46 & 795 & 13 & 468 \\ 965 & 648 & 5 & 60 \\ 67 & 464 & 84 & 541 \end{pmatrix}$$

$$maximum = (965, 795, 84, 541)$$

$$max_{temp} = 0$$

- Ende der Spalte erreicht
- Keine weiteren Kandidaten:
Temporäres Maximum ist damit endgültig
- Ende der Matrix erreicht -
Aufgabe erfüllt!
- Temporäres Maximum wird nicht mehr gebraucht, Wert egal

for-Schleife: Matrix-Beispiel

Implementierung des Matrixbeispiels in Java

```
1 public class MatrixMax {
2     public static void main(String [] args) {
3         int matrix [][] = {{ 46, 795, 13, 468 },
4                             { 965, 648, 5, 60 },
5                             { 67, 464, 84, 541 }};
6
7         int maximum [] = new int [4];
8         int maxtemp = 0;
9
10
11
12
13
14
15
16
17
18
19     }
20 }
21 }
```

for-Schleife: Matrix-Beispiel

Implementierung des Matrixbeispiels in Java

```
1 public class MatrixMax {  
2     public static void main(String [] args) {  
3         int matrix [][] = {{ 46, 795, 13, 468 },  
4                             { 965, 648, 5, 60 },  
5                             { 67, 464, 84, 541 }}};  
6  
7         int maximum [] = new int [4];  
8         int maxtemp = 0;  
9  
10        for (int j = 0; j < 4; j++) {  
11  
12  
13  
14  
15  
16  
17  
18  
19        }  
20    }  
21 }
```

for-Schleife: Matrix-Beispiel

Implementierung des Matrixbeispiels in Java

```
1 public class MatrixMax {
2     public static void main(String [] args) {
3         int matrix [][] = {{ 46, 795, 13, 468 },
4                             { 965, 648, 5, 60 },
5                             { 67, 464, 84, 541 }};
6
7         int maximum [] = new int [4];
8         int maxtemp = 0;
9
10        for (int j = 0; j < 4; j++) {
11            for (int i = 0; i < 3; i++) {
12
13
14
15
16            }
17
18        }
19    }
20 }
21 }
```

for-Schleife: Matrix-Beispiel

Implementierung des Matrixbeispiels in Java

```
1 public class MatrixMax {
2     public static void main(String [] args) {
3         int matrix [][] = {{ 46, 795, 13, 468 },
4                             { 965, 648, 5, 60 },
5                             { 67, 464, 84, 541 }};
6
7         int maximum [] = new int [4];
8         int maxtemp = 0;
9
10        for (int j = 0; j < 4; j++) {
11
12            for (int i = 0; i < 3; i++) {
13                if (maxtemp < matrix[i][j]) {
14                    maxtemp = matrix[i][j];
15                }
16            }
17
18        }
19
20    }
21 }
```

for-Schleife: Matrix-Beispiel

Implementierung des Matrixbeispiels in Java

```
1 public class MatrixMax {
2     public static void main(String [] args) {
3         int matrix [] [] = {{ 46, 795, 13, 468 },
4                             { 965, 648, 5, 60 },
5                             { 67, 464, 84, 541 }};
6
7         int maximum [] = new int [4];
8         int maxtemp = 0;
9
10        for (int j = 0; j < 4; j++) {
11
12            for (int i = 0; i < 3; i++) {
13                if (maxtemp < matrix[i][j]) {
14                    maxtemp = matrix[i][j];
15                }
16            }
17            maximum[j] = maxtemp;
18            maxtemp = 0;
19        }
20    }
21 }
```

Fertig!

- 1 Arrays
 - Was ist ein Array?
 - Array-Bauanleitung
 - Mehrdimensionale Arrays
 - Fehlerquellen

- 2 Schleifen
 - Warum Schleifen?
 - Die while-Schleife
 - Die for-Schleife
 - Fehlerquellen



4!

Wo liegt der Fehler?

```
1 public class SchleifenFehler1 {
2     public static void main(String [] args) {
3
4         int grenze = 10;
5         int zahl = 1;
6
7         while (zahl < grenze) {
8             // Ist Zahl ungerade?
9             if (zahl % 2 == 1) {
10                System.out.println(zahl);
11                zahl++;
12            } } } }
```

Wo liegt der Fehler?

```
1 public class SchleifenFehler1 {  
2     public static void main(String[] args) {  
3  
4         int grenze = 10;  
5         int zahl = 1;  
6  
7         while (zahl < grenze) {  
8             // Ist Zahl ungerade?  
9             if (zahl % 2 == 1) {  
10                System.out.println(zahl);  
11                zahl++;  
12            } } } }
```

Kein Compilerfehler

```
$ javac SchleifenFehler1.java
```

Aber Endlosschleife

```
$ java SchleifenFehler1  
[...]
```

Lösung III: Auf Laufvariablen achten!

Problem: Laufvariable wird an einer Stelle hochgezählt, die irgendwann nicht mehr erreicht wird

Fehler behoben

```
1 public class SchleifenFehler1 {
2     public static void main(String [] args) {
3
4         int grenze = 10;
5         int zahl = 1;
6
7         while (zahl < grenze) {
8             // Ist Zahl ungerade?
9             if (zahl % 2 == 1) {
10                System.out.println(zahl);
11            }
12            zahl++;
13        }
14    }
15 }
```

Immer darauf achten, dass die Laufvariable hochgezählt wird!
Gute formatierte Klammerung und Einrückung hilft.

Wo liegt der Fehler?

```
1 public class SchleifenFehler2 {
2     public static void main(String[] args) {
3
4         int grenze = 9;
5         int zahl = 11;
6
7         while (zahl != grenze) {
8             zahl++;
9         }
10    }
11 }
```

4!

Wo liegt der Fehler?

```
1 public class SchleifenFehler2 {  
2     public static void main(String[] args) {  
3  
4         int grenze = 9;  
5         int zahl = 11;  
6  
7         while (zahl != grenze) {  
8             zahl++;  
9         }  
10    }  
11 }
```

Kein Compilerfehler

```
$ javac SchleifenFehler2.java
```

Aber Endlosschleife

```
$ java SchleifenFehler2  
[...]
```

Lösung IV: Auf Variablenbereich achten!

Problem: Laufvariable wird beim Hochzählen nie den Wert von „grenze“ erreichen, da sie schon von Anfang an höher ist

Fehler behoben

```
1 public class SchleifenFehler2 {  
2     public static void main(String [] args) {  
3  
4         int grenze = 9;  
5         int zahl = 11;  
6  
7         while (zahl < grenze) {  
8             zahl++;  
9         }  
10    }  
11 }
```

- Abbruchbedingung so gestalten, dass sie irgendwann erfüllt wird
- Im Zweifelsfall wird Schleife ganz übersprungen

Alternative Lösung IV: Abbruch mit break

Der letzte Ausweg: break

```
1 public class SchleifenFehler2Break {
2     public static void main(String[] args) {
3
4         int grenze = 9;
5         int zahl = 11;
6
7         while (zahl != grenze) {
8             zahl++;
9
10            if (zahl > 1000) {
11                break; // Zahl zu groß – Abbruch
12            }
13        }
14        System.out.println("Fertig.");
15    }
16 }
```

- Herausspringen aus der innersten aktuellen Schleife
- Danach wird darunter weitergemacht (Hier Ausgabe „Fertig.“)

Wo liegt der Fehler?

```
1 public class SchleifenFehler3 {  
2     public static void main(String [] args) {  
3  
4         for (int i = 0; i < 10; i++) {  
5             System.out.println("Hallo " + i);  
6         }  
7         System.out.println("Tschüss " + i);  
8     }  
9 }
```

4!

Wo liegt der Fehler?

```
1 public class SchleifenFehler3 {  
2     public static void main(String [] args) {  
3  
4         for (int i = 0; i < 10; i++) {  
5             System.out.println("Hallo " + i);  
6         }  
7         System.out.println("Tschüss " + i);  
8     }  
9 }
```

Compilerfehler

```
$ javac SchleifenFehler3.java  
SchleifenFehler3.java:7: cannot find symbol  
symbol   : variable i  
location: class SchleifenFehler3  
System.out.println("Tschüss " + i);  
                             ^  
  
1 error
```

i ist nur innerhalb der for-Schleife gültig.

Zusammenfassung: Schleifen

- Schleifen:
 - Folge von Befehlen
 - Werden immer wieder ausgeführt
 - Wiederholung, solange eine bestimmte Bedingung erfüllt ist
 - Oft mit Zähl- oder Laufvariable (oft *i* genannt)

- Syntax:

while-Schleife: `while (bedingung)`
`{ befehle }`

for-Schleife: `for (initialisierung;`
`bedingung;`
`hochzählen)`
`{ befehle }`

