

4.4 Arrays und for-Schleifen



- In vielen Anwendungen gibt es das Problem eine Folge von Daten des gleichen Typs zu verarbeiten.
 - Gegeben die Notenpunkte der Studierenden des Moduls Praktische Informatik 1. Gesucht ist die Durchschnittsnote.
- Arrays repräsentieren Folgen von Datenelementen des gleichen Typs
 - Mathematisch: x_0, x_1, x_2, x_3
 - Java: $x[0], x[1], x[2], x[3]$
 - In Java können wir mit Hilfe einer Variable auf alle Folgeelemente eines Arrays zugreifen.
 - Im Gegensatz zu den bisher bekannten Variablen, muss der Speicherplatz eines Arrays **im Programm explizit reserviert** werden.

4.4.1 Der Array Datentypen

- Voraussetzung
 - T ist bereits ein bekannter Datentyp
- Formale Definition des Datentyps und seine Operationen

SORT $T[]$

OPS

length : array	→ int
new : int	→ T-array
[] : T-array x int	→ T

Deklaration

- Zu jedem beliebigen Typ T kann ein Array-Typ definiert werden.
 - **T[]**
 - Beispiele für Typen
 - **int[]**
 - **double[]**
- Wie für jeden anderen Typ können zu einem Array-Typ Variablen deklariert werden.
 - **int[] x;** // x ist eine Variable für eine Folge von ganzen Zahlen
 - **double[] r;** // r ist eine Variable für eine Folge von Gleitpunktzahlen
- Im Gegensatz zu Variablen primitiver Datentypen **verweisen** diese Variablen nur **auf den Speicherplatz** eines Arrays (Details später).
 - Wir sprechen dann von einer **Referenzvariablen**.
 - Der Speicherplatz für das Array wird durch die Variablendeklaration noch nicht reserviert.

Beispiel

```
/** Erster Entwurf einer Methode zur Speicherung der Zahlen 1,2,3,... in einem Array
 *  und der Berechnung der Summe.
 */
int gaussSumme () {
    int[ ] arr;
    int sum = 0;
    int i = 0;
    // Code zur Speicherplatzreservierung und Initialisierung des Arrays arr
    // Code zur Berechnung der Summe
    return sum;
}
...
```

Erzeugung eines Arrays

- Die **Speicherplatzreservierung** für Arrays erfolgt durch den **new-Operator**
 - **arr = new int[10];**
// Folge arr[0], arr[1], ... , arr[9] wird erzeugt; **arr** verweist auf die Folge
 - Bei dem new-Operator muss der **Typ des Arrays** und die **Größe des Arrays** zwischen den eckigen Klammern angegeben werden.
- Der new-Operator liefert als **Ergebnis die Speicheradresse** des Arrays. Diese Speicheradresse wird in der **Referenzvariable** hinterlegt.

// Erster Entwurf einer Methode zur Berechnung von einer Summe

```
int gaussSumme () {  
    int[] arr;  
    int sum = 0;  
    int i = 0;  
    arr = new int[10];  
    // Code für die Initialisierung  
    // Code zur Berechnung der Summe  
    return sum;  
}  
...
```

Initialisierung eines Arrays

- Ein Array kann elementweise initialisiert werden.
- Auf jedes Element des Array kann **schreibend zugegriffen** werden, in dem der **Selektionsoperator []** benutzt wird.

```
arr[1] = 2;
```

- Entsprechend kann auch durch Verwendung des Selektionsoperators **lesend** auf die Elemente des Array **zugegriffen** werden.

```
sum += arr[1];
```

```
// Erster Entwurf einer Methode zur Berechnung von einer Summe
```

```
int gaussSumme () {
```

```
    int[] arr;
```

```
// Deklaration der Array-Variable
```

```
    int sum = 0;
```

```
    int i = 0;
```

```
    arr = new int[10];
```

```
// Erzeugung des Arrays mit 10 Elementen
```

```
    while (i < 10) {
```

```
// Initialisierung (Wertzuweisung)
```

```
        arr[i] = ++i;
```

```
    }
```

```
    // Code zur Berechnung der Summe
```

```
    return sum;
```

```
}
```

```
...
```

Länge eines Arrays

- Manchmal ist in der Methode nicht bekannt, wie lang das Array ist.
- Die **Länge des Arrays** `arr` erhält man stets durch den Ausdruck
`arr.length`
- Diese **Schreibweise mit dem Punkt** werden wir später noch öfter benutzen.

```
// Erster Entwurf einer Methode zur Berechnung von einer Summe
int gaussSumme () {
    int[] arr;
    int sum = 0;
    int i = 0;
    arr = new int[10];
    while (i < arr.length) {
        arr[i] = ++i;
    }
    // Code zur Berechnung der Summe
    return sum;
}
...
```

Länge eines Arrays

- Manchmal ist in der Methode nicht bekannt, wie lang das Array ist.
- Die **Länge des Arrays** `arr` erhält man stets durch den Ausdruck
`arr.length`
- Diese **Schreibweise mit dem Punkt** werden wir später noch öfter benutzen.

```
// Erster Entwurf einer Methode zur Berechnung der Summe
int gaussSumme () {
    int[] arr;
    int sum = 0;
    int i = 0;
    arr = new int[10];
    while (i < arr.length) {
        arr[i] = ++i;
    }
    // Code zur Berechnung der Summe
    return sum;
}
...
```

Achtung: Die Indizes
beginnen bei 0. Daher hat
ein Array der Länge 10
die Indizes 0 .. 9.

4.4.2 Speicherplatz im Programm



- Jedes Programm besitzt zwei Arten von Speicher
 - **Stack-Speicher**
 - Dort werden beim Aufruf einer Methode der Speicherplatz für die Variablen der Methode abgelegt (**Methodeninstanz**).
 - **Heap-Speicher**
 - Dort werden die Arrays (und Objekte) abgelegt, die mit dem **new-Operator** erzeugt werden.
 - Diese Arrays können nicht direkt angesprochen werden, sondern nur indirekt über eine **Referenzvariable**.
 - Der Wert einer Referenzvariable kann entweder im Stack-Speicher oder im Heap-Speicher liegen.
- Beim Aufruf der Methode gaussSumme werden drei Variablen auf den Stack-Speicher abgelegt.

Stack-Speicher

Aufrufer

→ Aufruf von gaussSumme()

Stack-Speicher

Bezeichner	Wert
?	?
?	?
arr	<siehe nächste Folie>
sum	0
i	0

Stack-Speicher

Aufrufer

Stack-Speicher

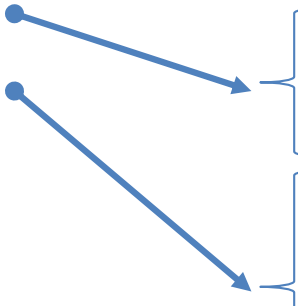
Bezeichner	Wert
?	?
?	?
arr	<siehe nächste Folie>
sum	0
i	0

Speicherplatz kann wieder verwendet werden, wenn Aufrufer eine neue Methode aufruft.

Stack-Speicher

<Programmende>

Stack-Speicher

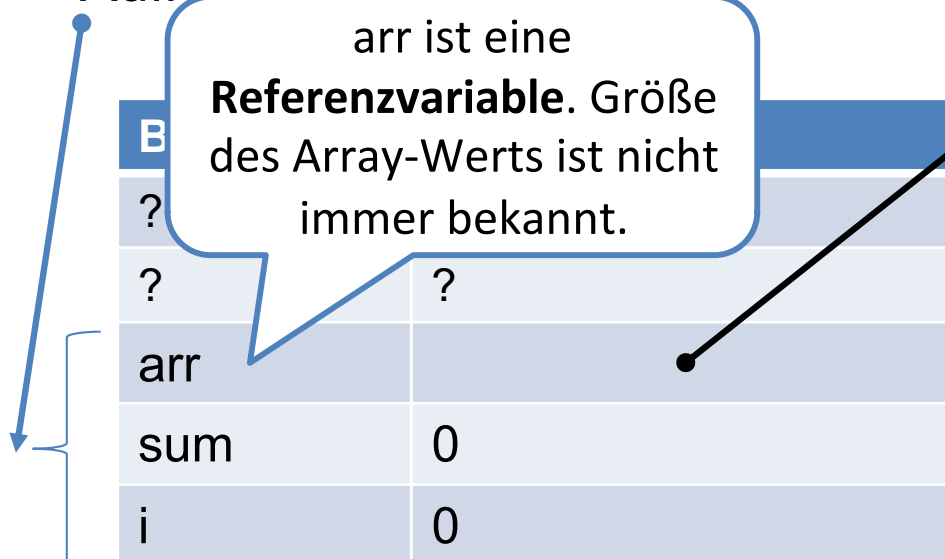


Bezeichner	Wert
?	?
?	?
arr	<siehe nächste Folie>
sum	0
i	0

Stack-Speicher

Aufrufer

→ Aufruf von `gaussSumme()`



Durch den Aufruf des `new`-Operators wird Speicherplatz im Heap-Speicher reserviert.

- `arr = new int[10];`

Adresse	Wert
1	?
2	?
...	...
41	?
42	0
43	0
44	0
45	0
46	0
47	0
48	0
49	0
50	0
51	0
52	?
...	

Werte der
Array-
Elemente

Heap-Speicher

Stack-Speicher

Aufrufer

→ Aufruf von gaussSumme()

Bezeichner	Wert
?	?
?	?
arr	
sum	0
i	0

Adresse	Wert
1	?
2	?
...	...
41	?
42	0
43	0
44	0
45	0
46	0
47	0
48	0
49	0
50	0
51	0
52	?
...	

Werte der
Array-
Elemente

Durch Verwendung des []-Operators können wir schreibend und lesend auf den Inhalt eines Arrays zugreifen.

- `arr[i] = i+1;`
- `arr2 = arr;`

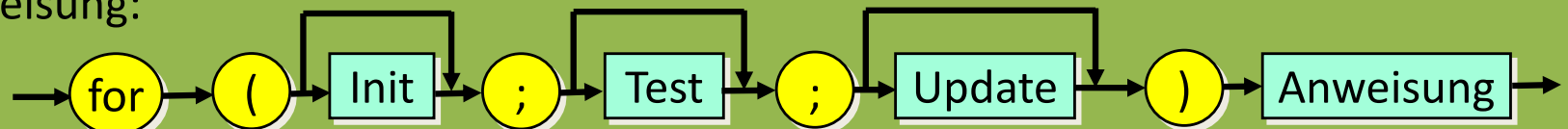
Speicheradresse (vereinfacht):
Array-Start + Index.

Heap-Speicher

4.4.3 for-Schleife

- Statt einer while-Schleife ist es oft einfacher eine for-Schleife zu benutzen.

for- Anweisung:



- Der Schleifenkopf der for-Schleife setzt sich zusammen aus
 - **Init** steht für eine oder mehrere durch Kommata getrennte Zuweisungen oder Variablendeklarationen mit Initialisierung.
 - `int i = 0`
 - **Test** steht für eine Bedingung, die meistens testet, ob in **Init** genannte Variablen eine Schranke überschreiten.
 - `i < max`
 - **Update** steht für eine oder mehrere durch Kommata getrennte Anweisungen die meist die in **Init** genannten Variablen verändern.
 - `i += 1`

for-Schleife und while-Schleife

- ▶ Die **for-Anweisung** ist **äquivalent** zu folgender **while-Anweisung** und standardisiert somit genau diesen Typ von while-Anweisungen:

```
{  
    Init;  
    while ( Test ) {  
        Anweisung;  
        Update;  
    }  
}
```

- ▶ Eigentlich wird die for-Schleife nicht benötigt, sondern wird nur zur Vereinfachung der Programmierung angeboten.
- ▶ Jede der drei Komponenten einer for-Anweisung können auch leer sein. Daher ist folgende Anweisung die kürzest mögliche for-Anweisung:

```
for ( ; ; ) ;
```

Keine Abbruchbedingung!
Endlosschleife.

for-Schleife: Einfache Beispiele

- Das folgende Beispiel zeigt eine for-Schleife, die die **Fakultäts-Funktion** iterativ berechnet und für **i=1,2,...,5** ausgibt:

```
int fak = 1;
for (int i=1; i < 6; i++){
    fak = i*fak;
    System.out.println("Fakultät von " + i + " ist: " + fak);
}
```

- Analog die Fibonacci-Funktion:*

```
int fibo0 = 1, fibo1 = 1, fiboneu;
for (int i=2; i < 6; i++){
    fiboneu = fibo0 + fibo1;
    fibo0 = fibo1;
    fibo1= fiboneu;
}
```

Verschachtelte for-Schleife: Beispiele

- Schleifen können auch verschachtelt sein.

```
int max = 10;
for (int i=0; i < max; i++){
    for ( int k=0 ; k < max-i-1 ; k++ )
        System.out.print(" ");
    for ( int k=0 ; k < 2*i-1 ; k++ )
        System.out.print("*");
    System.out.println();
}
```

- In diesem Beispiel werden 10 Zeilen ausgegeben.
 - In der 1. Zeile 1 Stern
 - In der 2. Zeile 3 Sterne
 - In der 3. Zeile 5 Sterne
 - usw.
 - Die Sterne sollen zentriert werden, d.h.
 - Vor der Sternausgabe müssen auch noch passend viele Leerzeichen ausgegeben werden.

```
      *
     ***
    *****
   *********
  ***********
 *****
*****
*****
*****
*****
*****
*****
*****
```

for-Schleife und Arrays

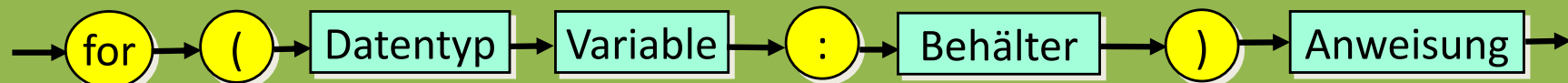
- for-Schleifen sind sehr gut geeignet, um Arrays zu durchlaufen.
 - Beachte: Die **Länge des Arrays** arr bekommt über **arr.length**

```
/** Erster Entwurf einer Methode zur Berechnung von einer Summe
int gaussSumme () {
    int[] arr;                // Deklaration einer Array-Variable
    int sum = 0;
    arr = new int[10];        // Reservierung des Speicherplatz
    for (int i = 0; i < arr.length; i++) { // Initialisierung
        arr[i] = i+1;
    }
    for (int i = 0; i < arr.length; i++) { // Lesender Zugriff auf das Array
        sum += arr[i];
    }
    return sum;
}
...

```

Syntax der foreach-Schleife

foreach- Anweisung:



- Dies ist die allgemeine Form der **foreach-Anweisung**. **Behälter** steht für den Namen einer Variablen einer **Behälter-Datenstruktur**.
 - Derzeit kennen wir in diesem Zusammenhang nur **Arrays** als Behälter.
- Der Datentyp am Anfang muss der Datentyp der Elemente des Behälters sein, also der Typ der Array Elemente.
 - Der folgende Code-Schnipsel zeigt die Schleife in unserem Beispiel bei der Berechnung der Summe.
 - Leider kann mit der foreach-Schleife **nicht schreibend** auf die Elemente des Arrays zugegriffen werden.

```
int[] arr = new int[10];
int sum = 0;
for (int i = 0; i < arr.length; i++) { // Hier müssen wir die alte for-Schleife verwenden.
    arr[i] = i+1; }
for (int elem : arr) {                // Hier funktioniert die neue Schleife.
    sum += elem; }                    // Lesender Zugriff ist erlaubt.
```

4.4.4 Besonderheiten bei Arrays

- Eine Zuweisung von Array-Variablen ist wie bei allen anderen Variablen möglich.

```
int[] arr1 = new int[5], arr2;
for (int i = 0; i < arr1.length; i++)
    arr1[i] = i+1;

arr2 = arr1;
```

- Aber was passiert dabei?
 - Der Wert der Variablen arr1 und arr2 ist eine Speicheradresse.
 - Somit wird bei der oberen Zuweisung dieser Wert von arr1 an arr2 übergeben.

Bezeichner	Wert
arr1	•
arr2	•

Adresse	Wert
1	?
2	?
...	...
41	?
42	1
43	2
44	3
45	4
46	5

Werte der
Array-
Elemente

Solche Zuweisungen passieren insbesondere beim Aufruf einer Methode mit einem Array als Parametervariable.

4.4.4 Besonderheiten bei Arrays

- Eine Zuweisung von Array-Variablen ist wie bei allen anderen Variablen möglich.

```
int[] arr1 = new int[5], arr2;
for (int i = 0; i < arr.length; i++)
    arr1[i] = i+1;

arr2 = arr1;
```

- Aber was passiert dabei?
 - zwei verschiedene Referenzvariablen können auf das gleiche Array zugreifen
 - arr1[1] = 6;
if (arr2[1] == 6) System.out.println("Achtung");
// erfüllt.

Bezeichner	Wert
arr1	●
arr2	●

Adresse	Wert
1	?
2	?
...	...
41	?
42	1
43	6
44	3
45	4
46	5

arr1[0] und
arr2[0]

arr1[1] und
arr2[1]
Elemente

Geteilte Referenzvariablen

- Wir ein Array an eine Methode übergeben, so kann diese den Arrayinhalt ändern

```
/** Methode zur Berechnung von einer Summe eines Arrays
 * @param arr ein Array mit ganzen Zahlen.
 * @return Die Summe der Zahlen des Arrays.
 */
int getSumme (int[] arr) {                // Array-Parametervariable
    for (int i = 1; i < arr.length; i++) {
        arr[0] += arr[i];
    }
    return arr[0];
}
```

Verändert den
Arrayinhalt.

```
// Aufruf in der jshell
int arr[] = {1, 2, 3};
int sum = getSumme(arr);
System.out.println("Summe = " + sum);
for (int e : arr) { System.out.print(e + ", "); }
```

Geänderter Inhalt auch
beim Aufrufer.

Array Literale

- Ähnlich wie bei anderen Typen können **konstante Arrays in einer speziellen Syntax** angegeben werden.
 - Die Array-Elemente stehen dabei mit Komma getrennt in einem Mengenklammerpaar.

`{1,2,3,4,5}`

- Diese Array-Literale können **ausschließlich** bei der
 - Deklaration einer lokalen Array-Variable
`int[] arr = {1,2,3,4,5};`
 - und beim Aufruf einer Methode verwendet werden. Hierbei muss aber noch der Typ des Arrays zusätzlich angegeben werden.
`long res = getSumme(new int[] {1,2,3,4,5});`

Arrays als Parametervariablen

- Die Methode bekommt ein Array übergeben und liefert die Summe zurück.

```
/** Methode zur Berechnung von einer Summe eines Arrays
 * @param arr ein Array mit ganzen Zahlen.
 * @return Die Summe der Zahlen des Arrays.
 */
long getSumme (int[] arr) {                // Array-Parametervariable
    long sum = 0;
    for (int elem : arr) {                // Neue Schleifensyntax
        sum += elem;
    }
    return sum;
}

// Aufruf in der jshell
long l = getSumme(new int[]{1,2,3});      // Literal als Parameter
System.out.println("Summe = " + l);
```

Standardwerte für Array-Elemente

- Bisher: Zugriff auf Variablen-Werte erst nach deren Initialisierung
- Ist das erlaubt?

```
int[] notenspiegel = new int[15];  
System.out.println(notenspiegel[14]);
```
- Ja, es ist erlaubt: Für Arrayelemente gelten nicht die gleichen Regeln wie für lokale Variablen.
- Aber: welchen Wert hat ein nicht-initialisiertes Arrayelement?
 - Java initialisiert Arrayelemente mit Standard-Werten:
 - 0, 0L für int, long, etc.
 - 0.0f, 0.0d für float, double
 - false für boolean
 - null für String, Arrays, etc.

Später mehr dazu.

Zuweisungs- und Inkrementoperatoren

- In Bezug auf die nötige Initialisierung unterscheiden sich Array-Elemente von lokalen Variablen.
- Die Zuweisungs- und Inkrementoperatoren funktionieren aber auch hier.

```
int arr[] arr = {1, 2, 3};  
arr[0]++;           // arr: {2, 2, 3}  
arr[1] *= 10;       // arr: {2, 20, 3}
```

Vergleich von Arrays

- Beim **Vergleich von** zwei Array-Variablen werden die **Referenzen** verglichen.
 - Beim folgenden Vergleich sind die Referenzen gleich. Somit ergibt der Vergleich also **true**

```
int[] a = { 17, 42, 47 };  
int[] b = a;  
System.out.println(a == b);
```

- *Beim folgenden Vergleich sind die Referenzen verschieden, da b eine Kopie von a referenziert.*
 - *Der Vergleich ergibt also **false** obwohl a eine identische Kopie von b ist!*

```
int[] a = { 17, 42, 47 };  
int[] b = new int[3];  
for (int i = 0; i < 3; i++)  
    b[i] = a[i];  
System.out.println(a == b);
```

Die Nullreferenz

- Eine Array-Variable kann mit Default-Wert **null** initialisiert werden.

```
int[] a = null;
```

- Eine bisher anders verwendete Array-Variable kann auch durch Zuweisung einer null-Referenz **außer Betrieb genommen werden**.

```
a = null;
```

- Dabei können Arrays entstehen, die nicht mehr referenziert werden.
 - Diese Arrays können nicht mehr genutzt werden. Sie sind also Datenmüll.
 - Die „Java-Müllabfuhr“ (**garbage collector**) sammelt den Müll ein und der Speicherplatz kann **recycelt** werden.

```
int[] a = { 17, 42, 47 };  
...  
a = null;
```

a: **null**

nicht mehr
referenzierter
Müll

17

42

47



- Benutzen wir später nochmal a, kommt es zu einem Abbruch des Programms. Eine sogenannte Exception, genauer NullPointerException, wird geworfen.

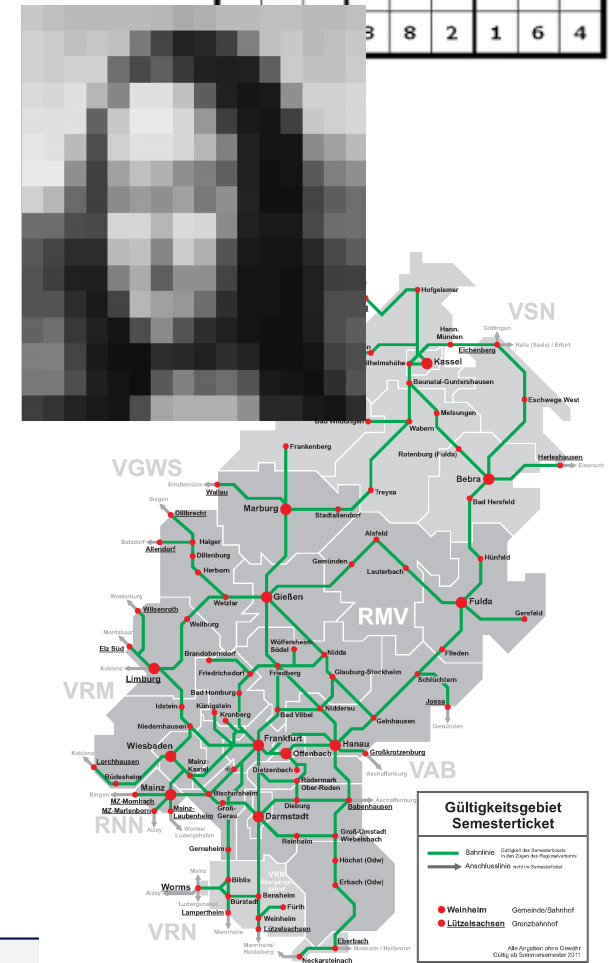
4.4.5 Mehrdimensionale Arrays

- Arrays kann man mit beliebigem Komponententyp T bilden.
 - Insbesondere kann der Datentyp selbst wieder ein Array sein.
 - Der zum Datentyp T[] gehörende Array-Datentyp ist somit:

$T[[]]$.

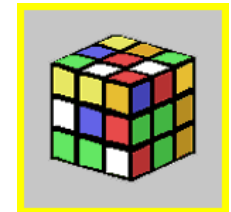
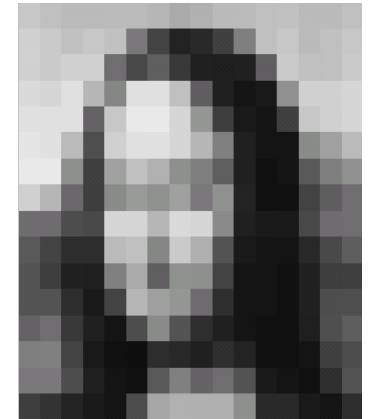
- Wir sprechen dann von einem zweidimensionalen und allgemein von mehrdimensionalen Arrays.
- Man benutzt mehrdimensionale Arrays z.B. zur Speicherung und Bearbeitung von
 - Bildern
 - Graphen
 - Wer ist mit wem im Netzwerk befreundet?
 - Welche Städte haben eine direkte Zugverbindung?
 - Distanztabellen

5	3	7	8	2	4	6	9	1
8	4	2	1	6	9	7	3	5
1	9	6	5	7	3	2	4	8
7	8	3	2	4	1	9	5	6
6	5	9	7	3	8	4	1	2
2	1	4	6	9	5	3	8	7
4	6	1	9	5	7	8	2	3
3	2	8	4	1	6	5	7	9
3	8	2	1	6	4			



Deklaration und Erzeugung

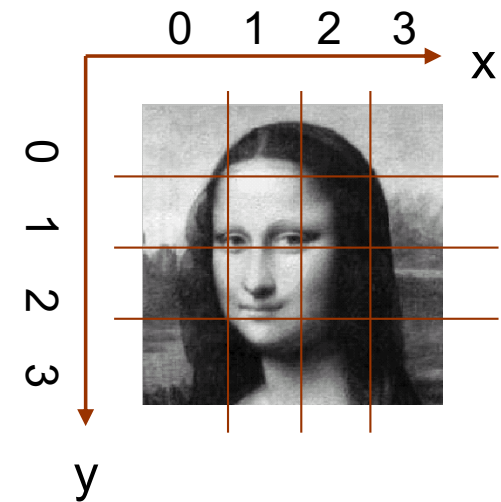
- Deklaration der Array-Variablen
 - `int [][] greyMonaLisa;`
 - `int [][] bildschirm;`
 - `Color[][][] rubik ;`
- Erzeugung eines Arrays
 - `bildschirm = new int [1024][748];`
 - `rubik = new Color[3][3][3];`
- Deklaration einer Array-Variable mit direkter Initialisierung
 - `boolean[][] xorTabelle =`
 `{{false, true},{true,false}};`
 - `int[][] entfernung = {`
 `{ 0, 213, 419, 882},`
 `{213, 0, 617, 720},`
 `{419, 617, 0, 521},`
 `{882, 720, 521, 0}`
 `};`



Beispiel: Bildbearbeitung

- Graphik als Matrix von Grauwerten

```
int[][] monaGrey = { {21,26,72,66},
                     {38,22,33,60},
                     {50,59,59,63},
                     {23,45,72,80} } ;
```



- Aufhellen

```
for (int x = 0; x < breite; x++)
    for(int y = 0; y < hoehe; y++)
        monaGrey[x][y] = monaGrey[x][y]*9/10 ;
```

- Negieren

```
for (int x = 0; x < breite; x++)
    for(int y =0; y < hoehe; y++)
        monaGrey[x][y] = 255-monaGrey[x][y] ;
```



- Schwarzweiss

```
for (int x = 0; x < breite; x++)
    for(int y =0; y < hoehe; y++)
        monaGrey[x][y] = (monaGrey[x][y] < 128)? 0: 255;
```

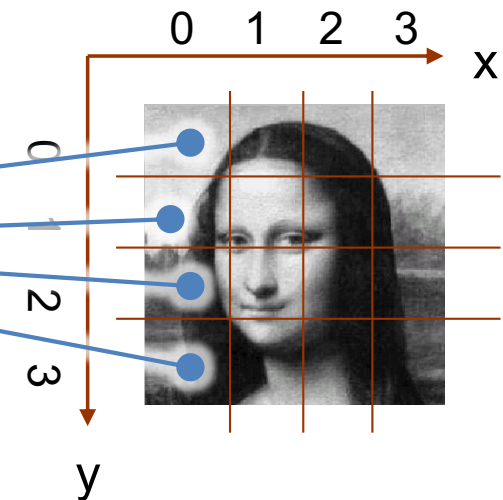


Fragezeigepoperator
(BoolExpr) ? Expr1 : Expr2

Beispiel: Bildbearbeitung

- Graphik als Matrix von Grauwerten

```
int[][] monaGrey = { {21,28,72,58},
                     {38,22,33,60},
                     {50,59,59,63},
                     {23,45,72,80} } ;
```



- Aufhellen

```
for (int x = 0; x < breite; x++)
    for(int y = 0; y < hoehe; y++)
        monaGrey[x][y] = monaGrey[x][y]*9/10 ;
```

- Negieren

```
for (int x = 0; x < breite; x++)
    for(int y =0; y < hoehe; y++)
        monaGrey[x][y] = 255-monaGrey[x][y] ;
```



- Schwarzweiss

```
for (int x = 0; x < breite; x++)
    for(int y =0; y < hoehe; y++)
        monaGrey[x][y] = (monaGrey[x][y] < 128)? 0: 255;
```



Fragezeigepoperator
(BoolExpr) ? Expr1 : Expr2

Speicherrepräsentation von Arrays

- Ein zweidimensionales Array ist ein Array von Spalten
- Ein dreidimensionales Array ist ein Array von zweidimensionalen Arrays
- ...

```
int[][] monaGrey =
{ {21,26,72,66},
  {38,22,33,60},
  {50,59,59,63},
  {23,45,72,80} } ;
```

Bezeichner	Wert
monaGrey	42

Stack-Speicher

1. Spalte

2. Spalte

Adresse	Wert
1	?
...	...
42	46
43	50
44	54
45	58
46	21
47	26
48	72
49	66
50	38
51	22
52	33
...	...

Heap-Speicher

Zusammenfassung

- Array als Datentyp
- Stack- und Heap-Speicher
 - Explizite Erzeugung von Arrays im Heap-Speicher
- Referenzvariablen
 - Verweisen auf ein Array
 - Lesender und schreibender Zugriff
- Mehrdimensionale Arrays
- Neue Schleifen
 - for-Schleife
 - foreach-Schleife