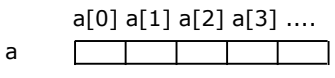
Arrays

Eindimensionale Arrays

Array = Tabelle gleichartiger Elemente



- Name **a** bezeichnet das gesamte Array.
- Elemente werden über Indizes angesprochen (z. B. **a[3]**).
- Indizierung beginnt bei 0.
- Elemente sind namenlose Variablen.

Deklaration

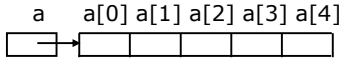
```
int[] a;
float[] b;
```

- Deklariert ein Array namens a (bzw. b).
- Seine Elemente sind vom Typ int (bzw. float).
- Seine Länge ist noch unbekannt.

Erzeugung

```
a = new int[5];
b = new float[10];
```

- Legt ein neues int- Array mit 5 Elementen an (aus dem Heap- Speicher).
- Weist seine Adresse a zu.



Array- Variablen enthalten Zeiger auf Arrays!

(Zeiger = Speicheradresse)

Arbeiten mit Arrays

Zugriff auf Arrayelemente

```
a[3] = 0;
int i =1;
a[ 3* i+ 1] = a[3];
```

- Arrayelemente werden wie Variablen benutzt.
- Index kann ein ganzzahliger Ausdruck sein.
- Laufzeitfehler, falls Array noch nicht erzeugt wurde.
- Laufzeitfehler, falls Index < 0 oder ≥ Arraylänge.

Arraylänge abfragen

```
int len = a. length;
```

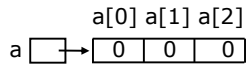
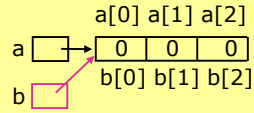
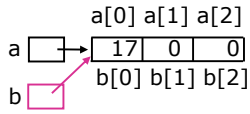
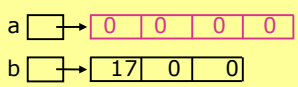
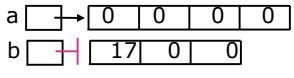
- length ist Standardoperator, der auf alle Arrays angewendet werden kann.
- Liefert Anzahl der Elemente (hier 5).

Beispiele

```
for (int i = 0; i < a. length; i++)      // Array einlesen
   a[ i] = In. readInt();

int sum = 0;                             // Elemente aufaddieren
for (int i = 0; i < a. length; i++)
    sum += a[ i];
```

Arrayzuweisung

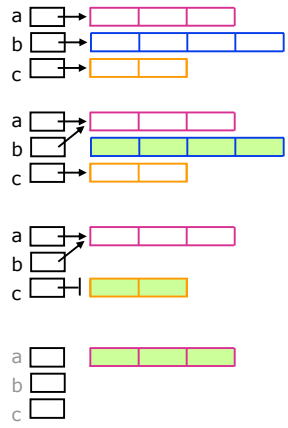
int[] a, b; a = new int[3];		Arrayelemente werden in Java standardmäßig mit 0 initialisiert .
b = a;		b bekommt denselben Wert wie a. Arrayzuweisung ist in Java Zeigerzuweisung!
a[0] = 17;		Ändert in diesem Fall auch b[0]
a= new int[4];		a zeigt jetzt auf neues Array.
b = null;		***null***: Spezialwert, der auf kein Objekt zeigt; kann jeder Arrayvariablen zugewiesen werden.

Freigeben von Arrayspeicher

Garbage Collection (Automatische Speicherbereinigung)

Objekte, auf die kein Zeiger mehr verweist, werden automatisch eingesammelt.
Ihr Speicher steht für neue Objekte zur Verfügung

```
static void P() {
   int[] a = new int[ 3];
   int[] b = new int[ 4];
   int[] c = new int[ 2];
   ...
   ...
   b = a;
   ...
   ...
  c = null;
  ...
  ...
  ...
}
```



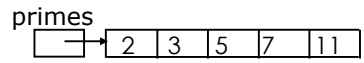
Kein Zeiger mehr auf dieses Objekt wird eingesammelt.

Kein Zeiger mehr auf dieses Objekt wird eingesammelt.

Am Methodenende werden lokale Variablen freigegeben, Zeiger ***a, b, c*** fallen weg, Objekt wird eingesammelt.

Initialisieren von Arrays

```
int[] primes = {2, 3, 5, 7, 11};
```



identisch zu

```
int[] primes = new int[ 5];
primes[ 0] = 2;
primes[ 1] = 3;
primes[ 2] = 5;
primes[ 3] = 7;
primes[ 4] = 11;
```

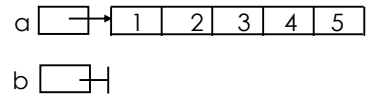
Initialisierung kann auch bei der Erzeugung erfolgen.

```
int[] primes ;
primes = new int[] {2, 3, 5, 7, 11};
```

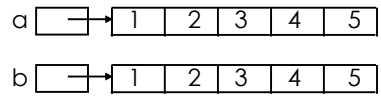
Kopieren von Arrays

int[ ] a = {1,2,3,4,5 };

int[ ] b;



b = (int[ ]) a. clone( ) ;



Typumwandlung nötig, da *clone* etwas vom Typ *Object[ ]* liefert.

Kommandozeilenparameter

Programmaufruf mit Parametern

java Programmname par_1 par_2 ... par_n

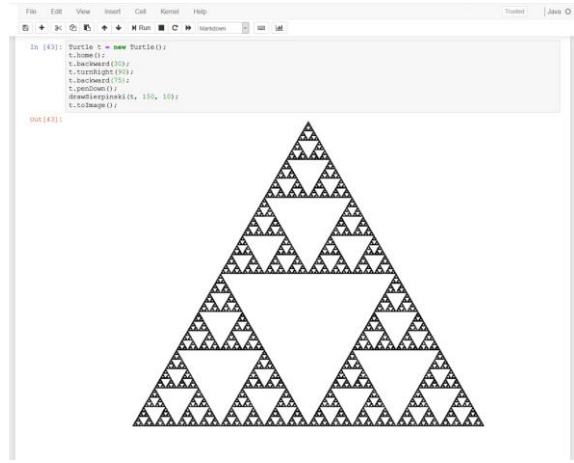
Parameter werden als String- Array an main- Methode übergeben

```
class Sample {
    public static void main (String[] arg) {
        for (int i = 0; i < arg. length; i++)
            System.out.println( arg[ i]);
            ...
    }
}
```

Aufruf z. B. : java Sample Anton /a 10

Ausgabe : Anton
/a
10

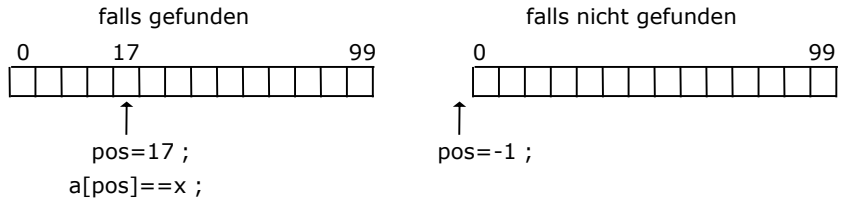
Interaktives Programmieren



Notebook: Arrays.ipynb

Beispiel: sequentielles Suchen

Suchen eines Werts x in einem Array mit 100 Einträgen



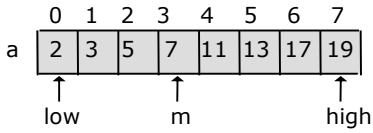

```
static int search (int[] a, int x) {
   int pos = a. length - 1;
   while (pos >= 0 && a[ pos] != x ) pos--;
   return pos;              // pos == -1 || a[ pos] == x
}
```

Achtung: **int[] a** wird nur als Zeiger übergeben.
Würde *search* etwas in a ändern (z. B. a[3] = 0;), würde sich diese Änderung auch auf das Array im Rufer auswirken.

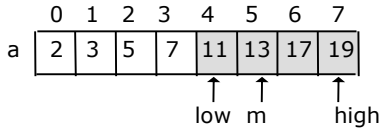
Beispiel: binäres Suchen

- Schneller als sequentielles Suchen.
- Array muß allerdings sortiert sein.

z. B. Suche von 13



- Index des mittleren Elements bestimmen (m = (low + high) / 2)
- 13 > a[m] zwischen a[m+ 1] und a[high] weitersuchen



Binäres Suchen

```
static int binarySearch (int[] a, int x) {
   int low = 0;
   int high = a. length - 1;
   while (low <= high) {
      int m = (low + high) / 2;
      if (a[ m] == x) return m;
      else if (x > a[ m]) low = m + 1;
           else  high = m - 1; /*   x < a[ m]  */
   } /*  low > high  */
   return -1;
}
```

2 3 3 5 7 7 8 10 11 15 16 17

2 3 3 5 7 7 8 10 11 15 16 17

2 3 3 5 7 7 8 10 11 15 16 17

2 3 3 5 7 7 8 10 11 15 16 17

- Suchraum wird in jedem Schritt halbiert.
- Bei n Arrayelementen sind höchstens log2(n) Schritte nötig, um jedes Element zu finden

n	seq. Suchen	bin. Suchen
10	10	4
100	100	7
1000	1000	10
10000	10000	14

Primzahlenberechnung: Sieb des Erathostenes

1. "Sieb" wird mit den natürlichen Zahlen ab 2 gefüllt.
   2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, ...

2. Erste Zahl im Sieb ist Primzahl. Entferne sie und alle ihre Vielfachen.
   (2), 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, ...
   3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, ...

3. Wiederhole Schritt 2
   (3), 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, ...
   5, 7, 11, 13, 17, 19, 23, 25, ...

... Wiederhole Schritt 2
   (5), 7, 11, 13, 17, 19, 23, 25, ...
   7, 11, 13, 17, 19, 23, ...

Implementierung

Sieb = *boolean*- Array, Zahl *i* im Sieb ⇔ sieve[i] == *true*

0	1	2	3	4	5	6	7	8	9	
false	false	true	true	true	true	true	true	true	true	.....

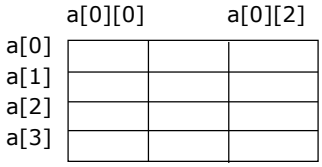
Zahl *i* entfernen: sieve[i] = *false*

0	1	2	3	4	5	6	7	8	9	
false	false	false	true	false	true	false	true	false	true	.....

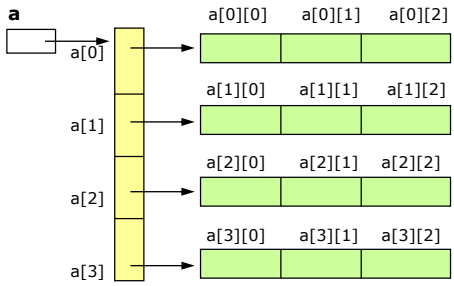
```
static void printPrimes (int max) {
    boolean[] sieve = new boolean[ max + 1];
    for (int i = 2; i <= max; i++) sieve[ i] = true;
    for (int i = 2; i <= max; ) {
          System.out.print( i + " ");              // i is prime
          for (int j = i; j <= max; j = j + i)  sieve[ j] = false;
          while (i <= max  &&  !sieve[ i])   i++;
     }
}
```

Mehrdimensionale Arrays

Zweidimensionales Array = Matrix



In Java als Array von Arrays implementiert



Deklaration und Erzeugung

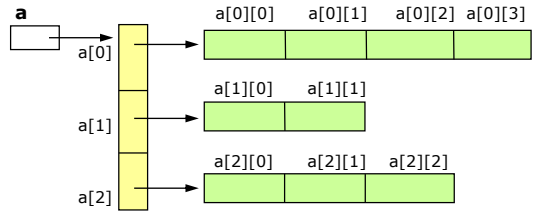
```
int[][] a;
a = new int[ 4][ 3];
```

Zugriff

```
a[ i][ j] = a[ i][ j+ 1];
```

Mehrdimensionale Arrays

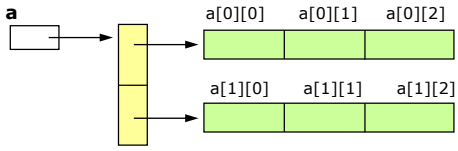
Zeilen können unterschiedlich lang sein (das ist aber selten sinnvoll)



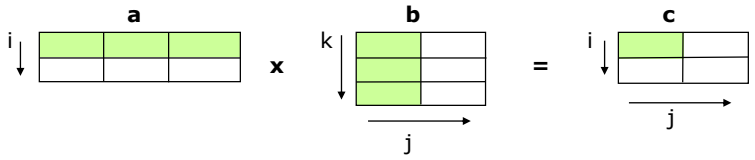

```
int[][] a = new int[ 3][ ];
a[ 0] = new int[ 4];
a[ 1] = new int[ 2];
a[ 2] = new int[ 3];
```

Initialisierung

```
int[][] a = {{ 1, 2, 3},{ 4, 5, 6}};
```



Beispiel: Matrixmultiplikation




```
static float[][] matrixMult (float[][] a, float[][] b) {
    float[][] c = new float[ a. length][ b[ 0]. length];
    for (int i = 0; i < a. length; i++)
        for (int j = 0; j < b[ 0]. length; j++) {
            float sum = 0;
            for (int k = 0; k < b. length; k++)
                sum += a[ i][ k] * b[ k][ j];
            c[ i][ j] = sum;
        }
    return c;
}
```

Iterator-Form der for-Anweisung *Java5*

In einer Schleife sollen alle Werte eines Arrays verwendet werden.

```
 ……
int[] primes ={2,3,5,7,11,13,17}
for(int i=0;i<primes.length;i++)   System.out.println( primes[i] );
……
```

Seit Java 1.5
gibt es hierzu eine spezielle äquivalente Form der for-Anweisung.

```
 ……
int[] primes ={2,3,5,7,11,13,17}
for( int p: primes )   System.out.println( p );
……
```

Methoden mit variabler Parameterzahl

```
static int sum( int[] values ){
            int result =0;
            for (int i =0; i < values.length; i++) result += values[i];
            return result;
           }
```

```
int res = sum( new int[] {1,2,5,9})    // Anwenden der Methode sum
```

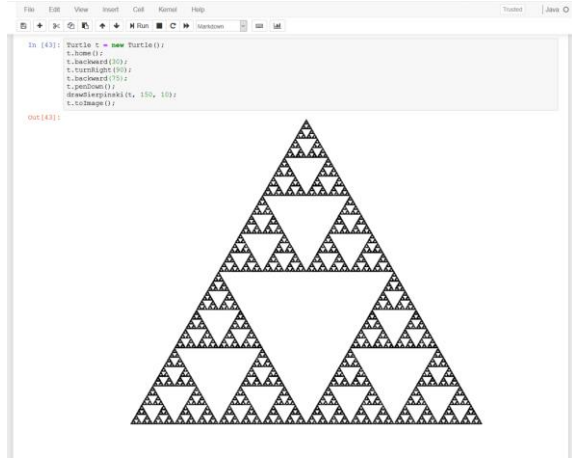
Es hierzu eine äquivalente Form.

```
static int sum2( int... values ){
            int result =0;
            for (int i =0; i < values.length; i++) result += values[i];
            return result;
           }
```

```
int res = sum2(1,2,5,9)                // Anwenden der Methode sum2
```

Wird vom Compiler automatisch in obige Form umgewandelt!

Interaktives Programmieren



Notebook: Arrays.ipynb