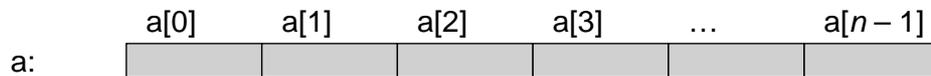


Kapitel 5: Arrays

- **Einführung**

Ein Array ist eine Reihung gleichartiger Objekte.



- Bezeichner a steht für das gesamte Array.
- Elemente werden über Indizes angesprochen: a[0], a[1], ... a[i], ...

- **Deklaration von Array-Variablen**

- Ein Arraytyp besteht aus dem Basistyp und eckigen Klammern “[]“:

```
int [] a;
```

```
float [] vector;
```
- Es wird noch kein Speicher für das Array belegt, die Länge ist unbekannt.
- Bevor ein Array benutzt werden kann, muss es erzeugt werden.

Erzeugung von Arrays

- **Erzeugen eines neuen Arrays**

```
int[] a = new int [5];
```

- legt neues int-Array mit 5 Elementen an und weist dessen Adresse a zu.
- Array-Variablen enthalten nicht Arrays, sondern Zeiger auf Arrays.



- **Länge von Arrays**

- Länge ist beliebig und nur durch verfügbaren Speicherplatz beschränkt.
- Länge eines jeden Arrays kann mit *length* abgefragt werden: `a.length`

- **Initialisierung mit Array-Konstanten**

```
double[] d = {0.17, 0.43, 5.78, 2.65}; // Array der Länge 4.
```

- **Äquivalent zu**

```
double[] d;
```

```
d = new double [4];
```

```
d[0] = 0.17;
```

```
d[1] = 0.43;
```

```
d[2] = 5.78;
```

```
d[3] = 2.65;
```

- **Array kann auch bei seiner Erzeugung initialisiert werden**

```
d = new int[] {0.17, 0.43, 5.78, 2.65};
```

Benutzung von Arrays

- Zugriff auf die Elemente
 - Elemente haben keine eigenen Namen, sie sind „anonyme“ Variablen.
 - Zugriff über []-Operator: $a[0] = 5$, $a[i] = a[2*i + 1] / a[0]$, ...
 - Gültiger Indexbereich: $0 \leq i \leq (a.length-1)$.
- Beispiel: Summe der Elemente im Array


```
int sum = 0;
for (int i = 0; i < a.length; i++) {
    sum = sum + a[i];
}
```
- Fehlermöglichkeiten
 - Zugriff auf Array, das noch nicht erzeugt ist.
 - Index liegt nicht im gültigen Bereich: $a[-1]$ oder $a[a.length]$

Zuweisen von Arrays

- Arrayvariablen sind Referenzen (d.h. Zeiger, Verweise, Adressen)
 - Bei Zuweisung wird nur Referenz kopiert, nicht der Inhalt eines Arrays.
 - Zuweisungen müssen typkompatibel sein (z.B. nicht $\text{float}[] \leftarrow \text{int}[]$).
 - Mehrere Arrayvariablen können auf dasselbe Array zeigen.

`int [] a = new int [5];` a: 

`int [] b = a;`
`/* jetzt ist a == b */` a:  b: 

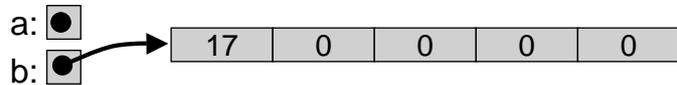
`a[0] = 17;`
`/* hier auch b[0] == 17 */` a:  b: 

Zuweisen von Arrays (2)

- Zuweisungen an Arrayvariablen überschreiben alte Referenzen

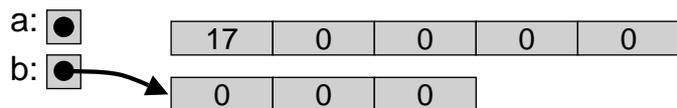
- Zuweisung der leeren Referenz

a = null;



- Zuweisung eines neuen Arrays

b = new int [3];



- Explizites Kopieren von Arrays möglich

- Mit Aufruf von clone(): `int[] b = a.clone();`
- Mit Systemmethode arraycopy: `System.arraycopy (a, 0, b, 0, a.length);`
Parameter: `void arraycopy (source, srcIndex, dest, destIndex, length);`

Arrays als Parameter

Vergleich von Arrays!!!

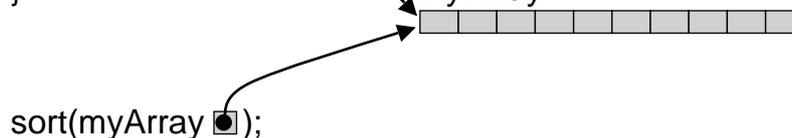
- Basistypen als Parameter (*call by value*)

- Werte der aktuellen Parameter werden in formale Parameter kopiert.
- Änderungen auf den Kopien wirken sich nicht auf Originale aus.
- Rückgabe von Ergebnissen durch Parameter ist **nicht** möglich.

- Arrays als Parameter (*call by reference*)

- Beim Aufruf wird der Zeiger übergeben, das Array wird nicht kopiert.
- Änderungen im Array werden für aufrufende Methode wirksam.

```
static void sort (int[] a) { // Kopie des Zeigers
    ... a[i] = a[i-1]; ...
}
```



Beispiel: Kürzen von Brüchen

- Bisher: keine Ergebnisrückgabe möglich

- Ergebnisrückgabe in neuem Array

```
static int[] reduce (int[] fraction) {  
    int t = ggt (fraction[0], fraction[1]);  
    int[] result = { fraction[0] / t, fraction[1] / t };  
    return result;  
}
```

- Ergebnisrückgabe in ursprünglichem Array

```
static void reduce (int[] fraction) {  
    int t = ggt (fraction[0], fraction[1]);  
    fraction[0] = fraction[0] / t;  
    fraction[1] = fraction[1] / t;  
}
```

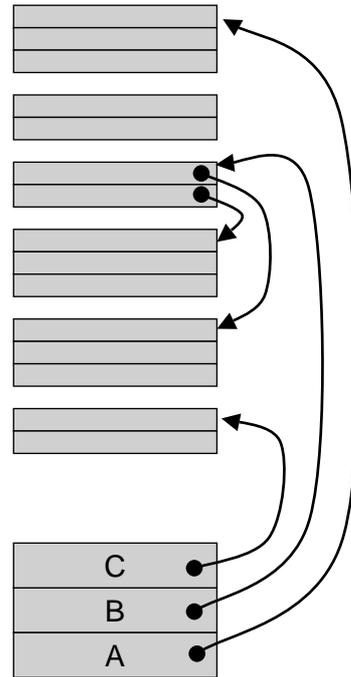
Speicherverwaltung in Programmen

- Statisches Segment
 - für Programmcode, statische Variablen, Array- und Stringkonstanten.
 - Lebensdauer der Objekte von Programmbeginn bis Programmende.
 - Speicherbelegung bleibt über den Programmablauf hinweg konstant.
- Stapel (Stack, Keller)
 - für lokale Variablen und Funktionsergebnisse.
 - Lebensdauer der Objekte jeweils vom Methodenaufruf bis zum Methodenende.
 - Speicherbelegung pulsiert im Rhythmus der Methodenaufrufe.
- Halde (Heap)
 - für dynamisch erzeugte Objekte wie Arrays, Strings u.a.
 - Lebensdauer eines Objektes ab Erzeugung (**new**), bis es nicht mehr erreichbar ist; explizite Freigabe nicht möglich, sondern automatische Freigabe von Zeit zu Zeit.
 - Speicherbelegung ist dynamisch und für das System nicht vorhersehbar.

Freispeicherverwaltung: Garbage Collection

- Prinzip
 - Dynamische Objekte leben nur solange sie erreichbar sind.
 - Suche statt nicht-erreichbarer Objekte die erreichbaren Objekte.
- Ablauf
 - Markiere rekursiv alle Objekte, die vom Heap aus (direkt oder indirekt) erreichbar sind.
 - Gib den gesamten verbliebenen Speicher frei.

```
int[] A;    // int [6]
int[][] B; // int [3][2]
float[] C; // float [2]
```



Beispiel: Sieb des Erathostenes

- Aufgabe: Berechne die Primzahlen von 1 bis n
 - Sieb: Betrachte natürliche Zahlen von 2 bis n .
 - Erste Zahl im Sieb (= 2) ist eine Primzahl; streiche alle Vielfachen.
 - Nächste verbliebene Zahl (= 3) ist eine Primzahl; streiche alle Vielfachen.
 - usw; streiche jeweils die Vielfachen der nächsten Primzahl.
- Implementierung
 - Stelle natürliche Zahlen von 0 bis n in einem boolean-Array *prime* dar.
 - Setze anfangs alle Zahlen auf "prim" (= true) und streiche dann wie oben.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
		t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t
		t	t	-	t	-	t	-	-	-	t	-	t	-	-	-	t	-	t	-

Sieb des Erathostenes (2)

```
static void printPrimes (int n) {
    boolean[] prime = new boolean [n+1];
    int i, j;
    for (i = 2; i <= n; i++) prime [i] = true;           // Initialize 2..n as prime
    i = 2;
    while (i <= n) {
        System.out.println (i);                       // i is prime
        for (j = i; j <= n; j = j + i) prime [j] = false; // remove multiples of i
        while (i <= n && !prime[i]) i++;              // locate next prime
    }
}
```

Beispiel: Monatstage berechnen

- Bisher: Berechnung durch Anweisungen

- if-else-Kaskade
- switch-Anweisung

- Neue Alternative: Ergebnisse aus Array holen

```
static int[] dayTab = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
days = dayTab [month - 1];
if (month == 2 && leapyear) days = 29;
```

- Implementierung in Methodenrumpf verbergen

```
static int daysOfMonth (int month, boolean leapyear) { ... }
```

Beispiel: Binäre Suche

Rekursive Lösung

```
static int search (int q, int[] a, int first, int last) {
    int m = (first + last) / 2;
    if (first > last) return -1;    // empty
    if (q == a[m]) return m;      // found q
    else if (q < a[m])
        return search (q, a, first, m-1);
    else /* q > a[m] */
        return search (q, a, m+1, last);
}

static int search (int query, int[] a) {
    return search (query, a, 0, a.length - 1);
}
```

Beispiel: Binäre Suche

Iterative Lösung

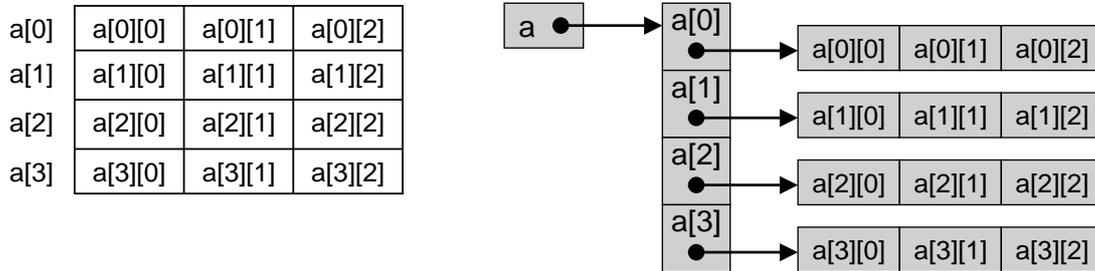
```
static int search (int q, int[] a) {
    int first = 0;
    int last = a.length - 1;
    while (last >= first) {
        int m = (last + first) / 2;
        if (q == a[m]) return m;
        else if (q < a[m]) last = m - 1;
        else /*q > a[m]*/ first = m + 1;
    }
    return -1;    // not found
}
```

Ohne return in der Schleife:

```
static int search (int q, int[] a) {
    int first = 0;
    int last = a.length - 1;
    int m;
    boolean notFound = true;
    while (last >= first && notFound) {
        m = (last + first) / 2;
        if (q == a[m]) notFound = false;
        else if (q < a[m]) last = m - 1;
        else /*q > a[m]*/ first = m + 1;
    }
    return notFound ? -1 : m;
}
```

Mehrdimensionale Arrays

- Zweidimensionale Arrays (= Matrizen) und deren Speicherung



- Deklaration

```
int [][] a = new int [4] [3]; // keine Initialisierung
int [][] m = { {1, 2, 3}, {4, 5, 6} }; // Initialisierung mit Konstanten
```

1	2	3
4	5	6

- Höhere Dimensionen

```
int [][][] q = new int [2][2][2]; // 3D: Quader
```

Beispiel: Matrixmultiplikation

```
/** matrix product of left and right */
static float[][] product (float[][] left, float[][] right) {
    float[][] prod = new float [left.length] [right[0].length];
    for (int i = 0; i < left.length; i++) {
        for (int j = 0; j < right[0].length; j++) {
            for (int k = 0; k < right.length; k++)
                prod_ij = prod_ij + left [i][k] * right [k][j];
            prod [i][j] = prod_ij;
        }
    }
    return prod ;
}
```

