Einführung in die Programmierung für NF

Arrays

ARRAY (REIHUNG)

Array

- In der Programmierung werden oft Tupel verschiedener Länge benutzt.

  Beispiel: Vektoren

  – (1.0, 1.0)
  – (0.2, 1.2, 7.0)

  Beispiel: Zeichenfolgen

  – ('L', 'M', 'U')
  – ('C', 'A', 'M', 'P', 'U', 'S')
- Es ist praktisch, direkt mit Tupeln zu arbeiten, anstatt z.B. eine einzelne Variable für jede Komponente einzuführen.
  (char a0 = 'L'; char a1 = 'M'; char a2 = 'U';)
- In Java gibt es einen Array-Typ, mit dem man Tupel wie ('L', 'M', 'U‘) mittels einer einzigen Variable verarbeiten kann.

Array

- Arrays repräsentieren Reihungen von Elementen mit wahlfreiem Zugriff:

- Man kann auf die Elemente einer Reihung einzeln zugreifen.

- Die Elemente werden durch ihren Index adressiert.

a: ('C', 'A', 'M', 'P', 'U', 'S')

Index: 0 1 2 3 4 5

- a kann als Abbildung aufgefasst werden

$$\texttt{a}: \{0, \ldots, 5\} \to \texttt{char}$$

$$\texttt{a}[\texttt{i}] = \begin{cases} \texttt{'C'} & \text{falls } \texttt{i} = 0 \\ \texttt{'A'} & \text{falls } \texttt{i} = 1 \\ \ldots & \\ \texttt{'S'} & \text{falls } \texttt{i} = 5 \end{cases}$$

Array

- Ein Array ist ein **Tupel von Komponentengliedern gleichen Typs**, auf die über einen Index direkt zugegriffen werden kann.
- Mathematisch kann eine Reihung mit *n* Komponenten vom Typ type als Abbildung

  $$I_n \longrightarrow \text{type}$$

  mit Definitionsbereich $I_n = \{0,1,2,...,n-1\}$ beschrieben werden.

  n bezeichnet die Länge der Reihung.
- Da type ein beliebiger Typ ist, können die Komponenten selbst wieder Arrays sein ⇒ **mehrdimensionale Arrays**

  'x' 'o' 'x‘

  'x' 'x' 'o‘ entspricht (('x', 'o', 'x'), ('x', 'x', 'o'), ('o', 'o', 'x'))

  'o' 'o' 'x‘

Array

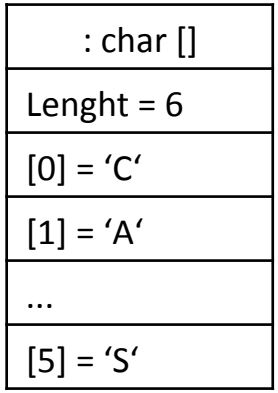
- Der Typ von Arrays mit Elementen vom Typ **type** wird in Java notiert durch:

  type[]

- Beispiele für Deklarationen von Arrayvariablen (ohne Initialisierung):

  int[] a; String[] args; char[][] t;

- UML-Darstellung für den Verbund, der ('C', 'A', 'M', 'P', 'U', 'S') repräsentiert:



- Ein Wert des Typs **type[]** repräsentiert ein Tupel mit Elementen vom Typ type als Verbund (Record) aus:
  - einer Reihe von Werten vom Typ type; und
  - einem ganzzahligen Attribut length, welches die Anzahl der Werte enthält.

Array - Zugriff

- a[i] bezeichnet den Zugriff auf die i-te Komponente von a

  Beispiel**:**
  - Mit a[0], a[1], …, a[5] kann man auf die Komponenten des Beispielarrays a zugreifen.
  - Diese Werte haben alle den Typ **char**.
  - Zugriff auf a[6], a[-1] oder a[100] löst einen Fehler aus.

```
a: ('C', 'A', 'M', 'P', 'U', 'S')
Index:  0   1   2   3   4   5
```

- a.length gibt die Länge des Arrays an

  Im Beispiel hat a.length den Wert 6

Array - Änderungen

- Man kann beliebige einzelne Buchstaben durch Zuweisungen ändern:

```
a[4] = 'E';
a[5] = 'R';
```

- Das ergibt ('C', 'A', 'M', 'P', 'E', 'R') als neuen Wert des Arrays. Außerdem hat a[4] nun den Wert 'E'.

- Die **Länge** eines Arrays kann **nicht** verändert werden.

Array - Beispiele

- Beispiel 1

```
public static void main(String[] args) {
 System.out.println("Argumentanzahl: " + args.length);
 for (int i = 0; i < args.length; i++) {
     System.out.println("Argument " + i + ": "+ args[i]);
 }
}
```

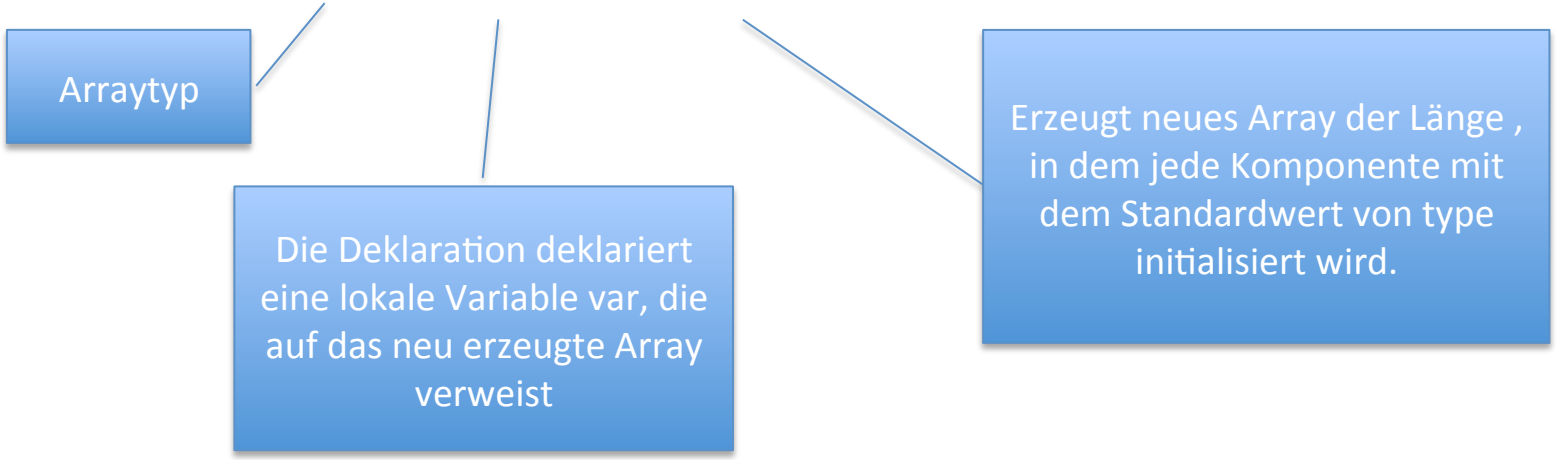
- Beispiel 2

```
char[] a = new char[6];
 a[0] = 'C';
 a[1] = 'A';
 a[2] = 'M';
 a[3] = 'P';
 a[4] = 'U';
 a[5] = 'S';
```

Array - Deklaration

- **Deklaration** eines neuen Arrays mit Elementen vom Typ type

```
type[] var = new type[n];
```



- Durch die Deklaration wird ein Array mit n Elementen im Heap angelegt. Diese Elemente werden mit Standardwerten initialisiert.

Array

- Man kann mit den Ausdrücken var[0],..., var[n-1] auf die einzelnen Komponenten von var zugreifen und diese so lesen und verändern.
- Standardwert (Defaultwert)

  von int : 0

  von double : 0.0

  von boolean : false

Array - Initialisierung

- Durch Zuweisung an die Komponenten:

```
type[] var = new type[n];
var[0] = v0;
...
var[n-1] = vn-1;
```

- Durch sofortige Zuweisung:

```
type[] var = new type[]{v0, ..., vn-1};
```

Array - Initialisierung

- Beispiel:

```
char[] a = new char[]{'C', 'A', 'M', 'P', 'U', 'S'};
```

Typ von a ist **char**[], d.h. der Typ eines eindimensionalen Arrays mit Elementen aus char.

- Man kann das Array initialisieren durch Einzelzuweisungen der Werte:

```
char[] a = new char[6];
a[0] = 'C';        a[3] = 'P';
a[1] = 'A';        a[4] = 'U';
a[2] = 'M';        a[5] = 'S';
```

Array – Was passiert hier?

```
char[] a = new char[] {'C', 'A', 'M', 'P', 'U', 'S'};
char[] b = new char[] {'L', 'M', 'U'};
```

```
b = a;
b[0] = 'T';
b[1] = 'E';
b[2] = 'M';
```

Array – Was passiert hier?

```
char[] a = new char[] {'C', 'A', 'M', 'P', 'U', 'S'};
char[] b = new char[] {'L', 'M', 'U'};


   b = a;
   b[0] = 'T';
   b[1] = 'E';
   b[2] = 'M`;

// b und a sind beide {'T','E','M','P','U','S'}
```

Array & for-Schleifen

- Die Länge eines Arrays steht in dem Attribut length.

```
int x = 10;
int[] myArray = new int[x*x+1];
int laenge = myArray.length;
```

- For-Schleifen eignen sich gut, um Arrays zu durchlaufen und zu verändern.

```
for (int k = 0; k < myArray.length; k++) {
   myArray[k] = 2 * myArray[k];
}
```

Array & for-Schleifen

- Lineare Suche nach einem Element in einem Array — mit vorzeitigem Verlassen:

```
int element = 16;
int k=0;
while (k < myArray.length && myArray[k] != element) {
    k++;
}
boolean found = (k < myArray.length);
```

Array - Übung

- Legen Sie ein int-Array an.
- Füllen Sie dieses mit den Werten:

  5 , 35 , 75 , 83
- Jetzt ändern Sie in einem Aufruf den Wert an der Stelle 0 auf 100 und lassen diesen Wert ausgeben.
- Anschließend lassen Sie die Werte an den Stellen einzeln in folgender Form ausgeben:

  „Das Array „Arrayname“ hat an der Stelle „Stelle“ den Wert „Wert“.“

Array - Suche nach dem Index des minimalen Elements

- Beispielarray: (3, -1, 15, 1, -1)

Array	3	-1	15	1	-1
Index	0	1	2	3	4

- **Algorithmus:**
  - Bezeichne mit minIndex den Index eines kleinsten Elements.
  - Initialisierung: minIndex = 0
  - Durchlaufe das ganze Array von links nach rechts. Im i-ten Schritt vergleiche das Arrayelement mit Index minIndex (d.h. a[minIndex]) mit dem Wert des aktuellen Elements (d.h. a[i]).

    Falls a[i] < a[minIndex] setze minIndex = i.

Array - Suche nach dem Index des minimalen Elements

Java Implementierung:

Es sei ein Array **int**[] a gegeben.

```
int minIndex = 0;
for (int i = 1; i < a.length; i++) {
// Wir fangen gleich bei i = 1 an, da a[0] < a[0] falsch ist.
 if (a[i] < a[minIndex]) {
     minIndex = i;
 }
}
int minElem = a[minIndex];
/* minElem ist der Wert des kleinsten Elements des Arrays;
 *minIndex ist der kleinste Index des kleinsten Elements */
```

Array – Binäre Suche eines Elements e

- a sei ein **geordnetes** Array mit den Grenzen j und k,

  d.h. a[i] ≤ a[i+1] für i = j,…,k-1; also z.B.:

a:	…	3	7	13	15	20	25	28	…
		j	j+1				k		

- **Algorithmus:**

  Um den Wert e in a zu suchen, teilt man das Array in der Mitte und vergleicht e mit dem Element in der Mitte:

  - Ist e < a[mid], so sucht man weiter im linken Teil a[j],…,a[mid-1]
  - Ist e = a[mid], so hat man das Element gefunden
  - Ist e > a[mid], so sucht man weiter im rechten Teil a[mid+1],…, a [k]

Array – Binäre Suche eines Elements e

- Schritt 1

Index	0	1	2	3	4	5	6	7
	3	7	13	15	20	25	28	30
	j			mid				k

- Schritt 2

Index	0	1	2	3	4	5	6	7
	3	7	13	15	20	25	28	30
					j	mid		k

- Schritt 3

Index	0	1	2	3	4	5	6	7
	3	7	13	15	20	25	28	30
	j			J,mid,k				k

Array – Binäre Suche eines Elements e

int e ist das Element, das im **geordneten** Array **int**[] a gesucht wird.

```
int j = 0;                    // linke Grenze
int k = a.length - 1;         // rechte Grenze
boolean found = false;        // wurde das Element schon gefunden?
while (!found && j <= k) {    // solange Array nicht leer und e nicht gefunden
 int mid = j + (k - j)/2;     // Mitte des Arrays
 /* Ist e kleiner das mittlere Element, so machen wir mit dem
  * linken Teilarray weiter. Ist es größer, dann mit dem rechten.
  * Anderenfalls haben wir den Wert im Array gefunden. */
 if (e < a[mid]) {
     k = mid - 1;
 } else if (e == a[mid]) {
     found = true;
 } else {
     j = mid + 1;
 }
}
boolean result = found;
```

Array - Zusammenfassung

- Arrays (Reihungen) sind mathematisch gesehen endliche Abbildungen von einem Indexbereich auf einen Elementbereich.
- Im Speicher werden Arrays repräsentiert als Zeiger auf Vektoren (vgl. später Objekte)
- Klassische Suchalgorithmen sind
  - die binäre Suche in einem geordneten Array und
  - die Suche nach dem Index mit dem kleinsten Element in einem ungeordneten Array.

Vielen Dank für Ihre Aufmerksamkeit