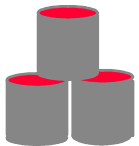
Grundlagen der Informatik
– Enums, Strings und Arrays –

Prof. Dr. Bernhard Schiefer

(basierend auf Unterlagen von Prof. Dr. Duque-Antón)

bernhard.schiefer@fh-kl.de
http://www.fh-kl.de/~schiefer



Inhalt

- Aufzählungen (Enums)
- Zeichenketten (Strings)
- Felder (Arrays)

Aufzählungstypen – Enum Types

- **Enums** (auch Aufzählungen genannt) sind Datentypen, die als Werte nur eine endliche geordnete Menge von Konstanten zulassen.
  - ⇨ Eine Variable eines Aufzählungstyps kann als Wert eine dieser Konstanten besitzen.
- Ein Aufzählungstyp trägt einen Namen. Bei der Definition des Typs werden die Aufzählungskonstanten in Form einer Liste angegeben.
  - ⇨ Beispiel:
    enum AmpelFarbe **{** ROT, GELB, GRUEN **}**
- **Enums** wurden in Java erst mit dem JDK 5.0 eingeführt und funktionieren daher mit älteren Versionen nicht.

Beispiel: Enums

```
class EnumTest {
   enum Wochentag {  MO, DI, MI, DO, FR, SA }

   public static void main (String[] args) {

      Wochentag tag= Wochentag.MI;

      switch (tag) {
         case MO  :  System.out.println ("Ein Montag"); break;
         case DI  :  System.out.println ("Ein Dienstag"); break;
         case MI  :  System.out.println ("Ein Mittwoch"); break;
         case DO  :  System.out.println ("Ein Donnerstag"); break;
         case FR  :  System.out.println ("Ein Freitag"); break;
         default  :  System.out.println ("Etwas anderes");
      } // switch
   } // main
} // class EnumTest
```

Strings

- Die Klasse **String** repräsentiert Zeichenketten
  - ⇨ Obwohl String kein primitiver Datentyp ist, kann diese Tatsache in Java zum Teil beim Umgang mit Zeichenketten „verschleiert“ werden.
  - ⇨ Die Klasse String bietet Möglichkeiten, komfortabel mit Zeichenketten zu arbeiten.
  - ⇨ Ohne genaue Kenntnisse der Objekt-Theorie wird daher nur der intuitive Umgang mit der Klasse erläutert.
  - ⇨ Die Elemente (Zeichen, char) einer Zeichenkette sind Unicode-Zeichen (2 Byte lang).

Erzeugung von String-Objekten

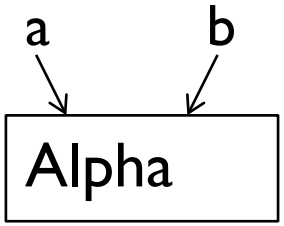
- Mit Hilfe der Klasse **String** können unveränderliche Zeichenketten bearbeitet werden.
- Zeichenketten können nach ihrer Konstruktion bzw. Initialisierung nicht mehr verändert werden.
  - ⇨ Es handelt sich also immer um String-Literale (konstante Zeichenketten), die als Objekte der Klasse String implementiert werden.
- String-Variablen sind eigentlich Referenzen
  - ⇨ die Variable enthält nur die Adresse des Ortes im Speicher, an dem der Inhalt des Strings steht.
  - ⇨ Wenn der Inhalt einer String-Variablen (Referenz) verändert wird, wird nur dort die Adresse eines neuen Objektes gespeichert – es ändert sich aber nichts am Inhalt des Objekts.

Erzeugung von String-Objekten

- String-Objekte (unbekannter Länge) können auf verschiedene Arten erzeugt werden:
  - ⇨ String nachname = "Hurtig";
  - ⇨ String vorname = new String ("Harry");
- Was passiert im folgenden Fall?
  - ⇨ String nachname;
    nachname = "Hurtig";
    nachname = "Meier";

String-Anmerkungen

- Eine Zeichenkette muss in Java in einer einzigen Zeile im Quelltext stehen, kann über mehrere Zeilen aber auch mit + verbunden werden.
  - ⇨ Beispiel: String alphabet = "ABCDEFGHIJKLMNOPQRST"
    + "UVWXYZ";
- Ein paar besondere Strings:
  - ⇨ "" : Leerer String
  - ⇨ "\"" : String bestehend aus einem einzelnen Anführungsstrich "
- String ist ein Referenztyp – mehrere Variablen können auf den gleichen Text verweisen:
  - ⇨ String a = "Alpha";
    String b = a;



Wichtige String-Methoden

- Die Klasse String enthält viele nützliche Methoden. Im folgenden werden einige wichtige davon vorgestellt:
  - **int** length **()** liefert die Länge der Zeichenkette.
  - **String** concat **(String** s**)** erzeugt einen neuen String, der den Inhalt von s an den des Strings der aufrufende Methode anhängt.
  - **String** substring **(int** begin**, int** end**)** erzeugt einen String, der alle Zeichen ab der Position begin bis end-1 enthält.
  - **boolean** equals **(Object** obj**)** liefert true, wenn ein obj ein String ist und das aktuelle String-Objekt (aufrufende Methode) und obj die gleiche Zeichenkette darstellen.
  - **boolean** endsWith **(String** s**)** liefert true, wenn die Zeichenkette im aktuellen String (aufrufende Methode) mit der Zeichenkette in s endet.

Wichtige String-Methoden

- **int** indexOf **(String** s**)** liefert die Position des ersten Zeichens des ersten Auftretens der Zeichenkette s im aktuellen String, andernfalls wird -1 zurückgegeben.
- **String** replace **(char** old**, char** new**)** erzeugt einen String, in dem jedes Auftreten des Zeichens old in der aktuellen Zeichenkette durch das Zeichen new ersetzt wird.
- **static String** valueOf **(Typ** x**)** wandelt x vom Typ in einen String um.
- **char** charAt **(int** index**)** liefert das Zeichen an der Position index der Zeichenkette. Das erste Zeichen hat die Position 0.

- Keine der Methoden verändert das String-Objekt!
  - Sie liefern alle nur einen Rückgabewert und erzeugen dabei (möglicherweise) neue String-Objekte.

Beispiel: Strings

```
class TestStrings {
   public static void main (String[ ] args)  {

      String nachname = "Schmitz";
      String vorname = new String ("Hugo");
      String name;

      // Verketten
      name = nachname + ", " + vorname;
      name = name.concat(" Egon");
      System.out.println("Name: " + name);

      // Laenge
      System.out.println("Länge: " + name.length () );

      // Extrahieren
      System.out.println("Anfang: " + name.substring (0, 13) );

      . . .
   } // main
} // class TestStrings
```

Beispiel: Strings II

```
class TestStrings {
   public static void main (String[ ] args)  {
      . . .
      // Vergleichen
      if (name.endsWith("Egon") )
         System.out.println("Name endet mit \"Egon\"" );

      if (name.startsWith("Schmitz") )
         System.out.println ("Nachname beginnt mit \"Schmitz\"" );

      // Suchen
      int position = name.indexOf("Hugo");
      if (position >= 0)
         System.out.println("Name enthält \"Hugo\" an Position "
                            + position);
      // Ersetzen
      String str = String.valueOf (3.57);
      System.out.println(str.replace ('.', ',') );
   } // main
} // class TestStrings
```

Beispiel: Strings III

```
class TestEqualStrings {
   public static void main (String[] args) {

      String name1 = "Hugo";
      String name2 = "Hugo";
      System.out.println(name1 +" == "+ name2 +" : "+ (name1 == name2));

      name2 = "Hu";
      System.out.println(name1 +" == "+ name2 +" : "+ (name1 == name2));

      name2 += "go";
      // Achtung!
      System.out.println(name1 +" == "+ name2 +" : "+ (name1 == name2));

      // Besser immer so:
      System.out.println("name1.equals(name2) : " + name1.equals(name2));
   } // main
} // class TestEqualStrings
```

Arrays

- Bisher wurde immer mit *einfachen* Variablen gearbeitet:
  - ⇨ Jede Variable kann genau einen Wert enthalten.
  - ⇨ Viele Aufgabenstellungen erfordern jedoch den Umgang mit einer ganzen Reihe von Werten.
- Java unterstützt auch Variablen, die Mengen von Werten speichern können – sogenannte Arrays
- Anwendungsfälle:
  - ⇨ Liste mit allen Matrikelnummern der Teilnehmer(innen) dieser Vorlesung
  - ⇨ Speichern einer ganzen Matrix von `m x n` Werten, wobei m die Anzahl der Zeilen und n die Anzahl der Spalten beschreibt.
- „Arrays“ werden auf deutsch auch als „Felder“ bezeichnet

Arrays

- Ein Array ist ein Objekt, das aus Komponenten (Elementen) zusammengesetzt ist
  - ⇨ Jedes Element eines Arrays muss vom selben Datentyp sein
- Beispiel
  - ⇨ Darstellung eines Arrays, welches aus 5 int-Elementen besteht:

- In Java kann ein Array aus Elementen beliebiger Datentypen aufgebaut werden.
- Ein Element eines Arrays kann daher auch selbst wieder ein Array sein, so dass auf diese Weise ein mehrdimensionales Array entsteht.

Arrays anlegen

- Die Definition einer Array-Variablen bedeutet in Java nicht das Anlegen eines Arrays, sondern die Definition einer Referenzvariablen, die auf ein Array-Objekt zeigen kann.
  - ⇨ Dieses Array-Objekt muss im Heap mit dem **new**-Operator angelegt werden.
- Die allgemeine Form einer Definition einer Referenzvariablen zum Zugriff auf ein eindimensionales Array ist:
  - ⇨ Typname [ ] arrayName;
  - ⇨ **Beispiel:** `int[] feld;`
- Anlegen im Heap mit Hilfe des new-Operators
  - ⇨ **Beispiel:** `feld = new int[5];`
- Definition eines zweidimensionalen Feldes: z.B. int[ ] [ ] feld;

Arrays anlegen 2

- Wie in Java üblich, können die Schritte auch zusammengefasst werden.
  - ⇨ Falls zum Zeitpunkt der Deklaration bereits die Länge bekannt ist, können Deklaration und Erzeugung zusammengefasst werden.
    - Beispiel: **int [ ] vector = new int[5];**
  - ⇨ Zusätzlich ist auch die Initialisierung möglich, wobei sich die Länge aus der Anzahl der zugewiesenen Literale ergibt.
    - Beispiel: **int [ ] vector = {10, 4, 7, 1, 8};**
- In Java werden die **Grenzen** eines Arrays genau **überwacht**.
  - ⇨ Im Gegensatz zu anderen Programmiersprachen ist es in Java nicht möglich, über die Grenzen eines Arrays hinaus andere Speicherbereiche zu überschreiben oder auszulesen.
  - ⇨ Ein Zugriff außerhalb der Grenzen des Arrays führt zu dem Melden einer Ausnahme

Zugriff auf Array

- Der Zugriff auf ein Element eines Arrays erfolgt über den Array-Index
  - ⇨ Die Indizierung der Array-Elemente beginnt bei einem Array mit **n** Elementen mit **0** und endet bei **n - 1**
  - ⇨ `feld = new int[5];`
    das erste Element kann mit feld [0] und das letzte Element mit feld [4] angesprochen werden.
- Ein Vorteil von Arrays gegenüber mehreren einfachen Variablen ist, dass Arrays sich gut mit Schleifen bearbeiten lassen.
  - ⇨ Index als Laufvariable in einer Schleife:

```
for (int i = 0; i < 5; i++) {
   System.out.println (feld [i]);
} // for
```

  - ⇨ Spezielle Iterator-Schleife:

```
for (int val : feld) {
   System.out.println (val);
} // for
```

Array Länge

- Jedes Array verfügt über ein Attribut `length`, in dem seine Länge gespeichert ist:

```
int []  feld = new int[5];
System.out.println (feld.length); // Ausgabe: 5
```

```
class ArrayTagTest {
   public static void main (String[] args) {

      String[] tage = {"Mo", "Di", "Mi", "Do", "Fr", "Sa", "So"};

      for (int i = 0; i < tage.length; i++) {
         System.out.print (tage[i] + " ");
      } // for

   } // main
} // class ArrayTagTest
```

Beispiel: ArrayRandomTest

```
class ArrayRandomTest {
   public static void main (String[] args) {

      int [] zufall = new int [10];

      for (int i = 0; i < zufall.length; i++) {
         zufall [i] = (int) (Math.random () * 100.);
         // Math.random () liefert einen (zufälligen) Double-Wert
         // zwischen 0 und 1: 0 <= x < 1
      } // for

      for (int val : zufall) {
         System.out.print (val + " ");
      } // for

   } // main
} // class ArrayRandomTest
```

Wichtige Eigenschaften von Arrays in Java

- Arrays sind stets Objekte, die zur Laufzeit im *Heap* angelegt werden.
  - ⇨ Die Dimension (Länge des anzulegenden Arrays) kann zur Laufzeit angegeben werden.
  - ⇨ Ist das Array angelegt, dann kann seine Länge nicht mehr verändert werden.
- Die Definition einer Array-Variablen bedeutet in Java nicht das Anlegen eines Arrays, sondern die Definition einer Referenz-Variablen, die auf ein Array-Objekt zeigen kann.
  - ⇨ Das Anlegen selbst erfolgt auf dem Heap mit dem new-Operator
  - ⇨ Die Indizierung beginnt dann mit dem ersten Element bei 0.

Wichtige Eigenschaften von Arrays in Java

- Im Grunde kann in Java nur ein (eindimensionales) Array
  - ⇨ aus Elementen eines einfachen Datentyps oder
  - ⇨ aus Elementen eines Referenztyps angelegt werden.
- Ein Element eines Arrays kann aber selbst auch wieder ein Array sein
  - ⇨ Auf diese Weise können mehrdimensionale Arrays produziert werden

Initialisierung von Arrays

- Beim Anlegen einer Array-Variablen werden die Feldinhalte mit einem typabhängigen Standardwert belegt

Datentyp	Standardwert
boolean	false
byte	0
short, int, long	0
float, double	0.0
char	\n0000

- Erinnerung: Bei einfachen Variablen war eine anwendungsspezifische Initialisierung notwendig
  - ⇨ Beispiel: `int val = 0;`
- Achtung: In vielen anderen Programmiersprachen sind die Variablen mit einem Zufallswert belegt!

Eigenschaften mehrdimensionaler Arrays

- Veranschaulichung eines Arrays ist subjektiv
  - ⇨ Ob eine Dimension eine Zeile (Horizontale Dimension) oder Spalte (Vertikale Dimension) darstellt, ist eine Frage der Interpretation.
- Im Grunde kann jede beliebige Anzahl von Dimensionen auf den 1-dimensionalen Fall abgebildet werden.
  - ⇨ Falls 2 Dimensionen gegeben sind, könnten z.B. alle Zeilen hintereinander geschrieben werden. Auf diese Weise ergibt sich eine entsprechend längere einzelne Zeile.
- Ungleichmäßige Arrays:
  - ⇨ Dimensionen der Unterebenen müssen nicht gleich sein, z.B. im 2-dimensionalen Fall kann die Matrix auch ein Dreieck oder anderes unförmiges Gebilde darstellen.

Anlegen von Arrays

- Zur Erinnerung: Schrittweises angelegen:
  - **Deklaration des Arrays**: Definition einer Referenzvariablen, die auf das Array-Objekt zeigen soll: **Typ [ ]** arrayName.
    Z.B. **int [ ]** vector**;**
  - **Erzeugung eines Arrays** (d.h. Array-Objekts), mit der new-Anweisung: **new Typ [Länge]**
    Z.B. **vector = new int[5];**
  - **Initialisierung des Arrays**: Dazu werden die Array-Elemente mit Werten belegt. Z.B. **vector [0] = ...; vector [4] = ...**
- Beispiel für eine zwei-dimensionale Matrix:
  - **Deklaration des Arrays**: int [ ] [ ] matrix;
  - **Erzeugung eines Arrays**: matrix = new int[2][3];
  - **Initialisierung des Arrays**:

```
for (int x = 0; x < 2; x++)
    for (int y = 0; y < 3; y++)
        matrix [x][y] = -1;
```

Zugriff auf die Vektor-Elemente

- Die Elemente eines Arrays der Größe n werden von 0 bis n-1 durchnummeriert.
  Der Zugriff auf ein Element erfolgt über seinen Index:
  - ⇨ **Arrayname [Ausdruck]**, wobei Ausdruck den Rückgabetyp int haben muss.
  - ⇨ **Arrayname [Ausdruck1][Ausdruck2]** im mehrdimensionalen Fall.
- Die Anzahl der Elemente eines Arrays kann über das Attribut length auch im mehrdimensionalen Fall abgefragt werden.
  Beispiel:
  - ⇨ `int[][] matrix = new int[2][3];`
  - ⇨ `matrix.length` liefert den Wert 2
  - ⇨ `matrix[0].length` liefert den Wert 3.

Beispiel: 2x3-Matrix

- Gegeben sei eine Matrix mit m = 2 Zeilen und n = 3 Spalten, also eine sogenannte 2 x 3 - Matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

- aij bezeichnet den Wert in der i-ten Zeile und j-ten Spalte
  - ⇨ Veranschaulichung:

int	int	int
int	int	int

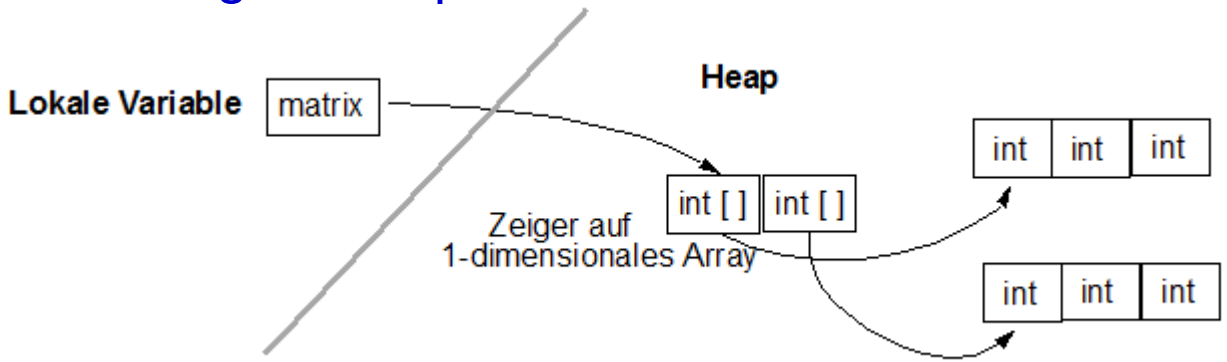
- Zur Verarbeitung von Arrays brauchen wir also
  - ⇨ eine Referenz-Variable A, die auf den Anfang des Arrays zeigt, und
  - ⇨ eine Indizierung, mit der das Feld korrekt durchwandert werden kann, etwa:
    A [i,j] = aij.

Beispiel: 2-dimensionales Array

```
class ArrayZweiTest {
   public static void main (String[ ] args) {

      int [][] matrix = { {11, 12, 13}, {21, 22, 23} };

      for (int i = 0; i < matrix.length; i++) {

         for (int j = 0; j < matrix[i].length; j++) {
            System.out.print(matrix [i][j] + " ");
         } // for j

         System.out.println ();
      } // for i

   } // main
} // class ArrayZweiTest
```

Beispiel: 2-dimensionales Array

- Visualisierung des Beispiels:



- Im Heap sind nun insgesamt drei (ein + zwei) 1-dimensionale Array-Objekte angelegt:
  - ⇨ Ein Array bestehend aus 2 Elementen von Zeiger auf int[] und
  - ⇨ Zwei Arrays bestehend jeweils aus 3 Elementen von int-Werten.
- Generell gilt: Zweite Dimension ist unabhängig von erster Dimension (gilt generell für alle Dimensionen)

Fortsetzung Beispiel: 2-dimensionales Array

```
class TestArrayZweiFortsetzung {
   public static void main (String[ ] args) {

      int [][] matrix;
      matrix = new int [2][ ];
      matrix [0] = new int [3];
      matrix [1] = new int [3];

      for (int zeile = 0; zeile < matrix.length; zeile++) {

         for (int spalte = 0;  spalte < matrix[zeile].length; spalte++) {
            matrix [zeile][spalte] = zeile + spalte;
            // bzw. irgendein sinnvoller Ausdruck
         } // for spalte

      } // for zeile

   } // main
} // class TestArrayZweiFortsetzung
```

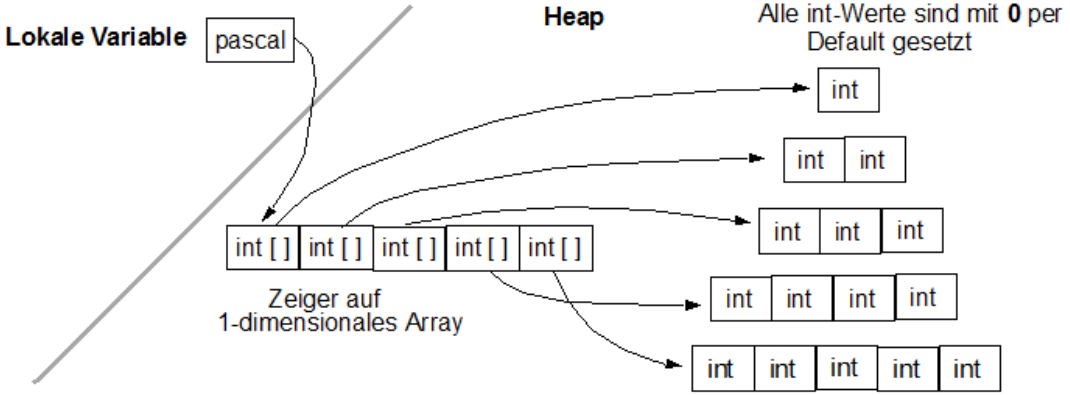
Anmerkung Beispiel: 2-dim. Array & Scanner

- Typischerweise muss bei dieser Art von Beispielen vor der Speicherplatz-Reservierung der einzelnen Dimensionen die konkrete Länge abgefragt werden. Dieser Sachverhalt könnte in Java wie folgt mit dem **Scanner** umgesetzt werden (Einlesen von Tastatur):

⇨
```
Scanner sc = new Scanner (System.in); // Scanner wird initialisiert
System.out.print ("Gib Anzahl der Zeilen an: ");
int zeile = sc.nextInt ();                    // Nutzer gibt ganze Zahl auf Konsole
                                              // ein -> Einlesen als int „zeile“
System.out.println ( ) ;
System.out.print ("Gib Anzahl der Spalten an: ");
int spalte = sc.nextInt ();
sc.close();                                   // Schließen des Scannerobjektes
```

Ungleichmäßige Arrays

- In Java müssen Arrays nicht immer eine „rechteckige“ bzw. „quaderförmige“ Struktur bilden.
- In Abhängigkeit der Problemstellung sind ungleichmäßige Arrays wie z.B. eine Dreiecksstruktur möglich.
- Beispiel:
  - ⇨ Umsetzung des Pascal’schen Dreieck zur geometrischen Darstellung der Binomialkoeffizienten.
  - ⇨ In der i-ten Zeile stehen genau i Werte. Das folgende Programm baut eine solche Struktur auf und gibt die Anzahl der benötigten Speicherplätze aus.

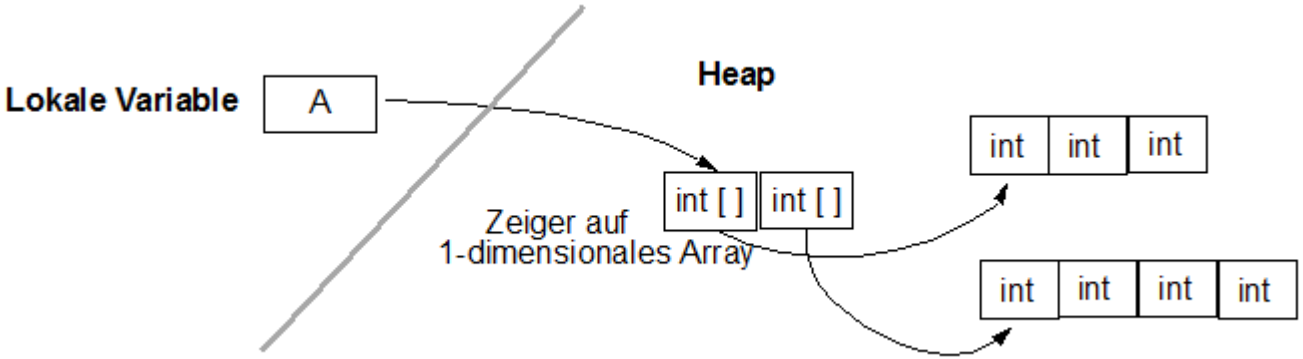


Beispiel: Pascal‘sches Dreieck

```
class ArrayPascalTest {
   public static void main (String[] args) {
      int n = 5; // Anzahl der Zeilen in Pascal-Dreieck

      // Deklaration und Erzeugung der ersten Dimension
      int [][] pascal = new int [n][];

      for (int i = 0; i < n; i++)   // Erzeugung der zweiten Dimension
         pascal[i] = new int [i+1];

      int anzFelder = 0;
      for (int i = 0; i < n; i++)
         for (int j = 0; j < pascal[i].length; j++)
            anzFelder++;

      System.out.println ("Anzahl der Felder insgesamt: " + anzFelder);

   } // main
} // class ArrayPascalTest
```

Mehrdimensionale Arrays

- In Java sind beliebige mehrdimensionale Arrays möglich, z.B.:
  - ⇨ int [] [] [] dreiDimMatrix = new int [10] [20] [30];
- Wie wir gesehen haben, müssen in Java die mehrdimensionalen Arrays nicht unbedingt rechteckig sein.
  - ⇨ Die unterschiedlichen Längen der einzelnen Elemente können mit Hilfe des length-Attributs ermittelt werden.
  - ⇨ Beispiel: int [] [] A = new int [2] [ ]; A[0] = new int [3]; A[1] = new int [4];
  - ⇨ Wie sehen die folgenden Werte aus? A.length, A[0].length und A[1].length



Offene Arrays

- In Java können auch offene Arrays erzeugt werden, d.h. Arrays bei denen die Länge einzelner Dimensionen nicht angegeben wird:
  - ⇨ Dazu werden bei der Speicherplatz-Allokation (new-Operator) einfach die eckigen Klammern leer gelassen:
    - Beispiel:
      int [] [] [] matrix = **new int** [5] [3] [];
- In Java sind die folgenden Konstrukte **nicht erlaubt**!
  - ⇨ In der ersten Dimension eines Arrays muss immer ein Wert zugewiesen werden.
    - Gegenbeispiel:
      int [] [] [] matrix = **new int** [] [] [];
  - ⇨ Nach einer leeren eckigen Klammer darf kein Wert in einer der nachfolgenden Klammern mehr angegeben werden:
    - Gegenbeispiel:
      int [] [] [] matrix = **new int** [5] [] [4];

Arrays: java.util

- Die Klasse Arrays aus dem Paket java.util bietet Methoden zum komfortablen Arbeiten mit Arrays. Beispiele:
  - static **void** *fill* (Typ[] feld, Typ w)
    - Alle Elemente aus feld erhalten den Wert w
  - static **void** *fill* (Typ[] feld, **int** from, **int** to, Typ w)
    - Alle Elementen ab Index from bis Index to-1 in feld erhalten den Wert w
  - static **void** *sort* (Typ[] feld)
    - die Elemente des Arrays feld werden aufsteigend sortiert
  - static **int** *binarySearch* (Typ[] feld, Typ schluessel)
    - durchsucht das Array feld nach dem Wert schluessel
    - im positiven Fall wird der entsprechende Index zurück geliefert
    - im negativen Fall wird ein negativer Index zurückgeliefert
    - die Elemente in feld müssen aufsteigend sortiert sein.