

Grundlagen der Programmierung

VL06: Referenzdatentypen

Prof. Dr. Samuel Kounev,
M.Sc. Norbert Schmitt



Danksagung

- Vorlesungsmaterialien von Prof. Dr. Detlef Seese wurden als Basis verwendet
- Unterstützung bei der technischen und inhaltlichen Gestaltung des Vorlesungsmaterials leisteten:

Jóakim v. Kistowski

Dietmar Ratz, Joachim Melcher, Roland Küstermann, Jana Weiner, Hagen Buchwald, Matthes Elstermann, Oliver Schöll, Niklas Kühl, Tobias Diederich

- **Felder** (*arrays*)
 - Deklaration
 - Erzeugung und Initialisierung
 - Mehrdimensionale Felder
 - Beispiele

- **Klassen** (*classes*)
 - Deklaration und Instanziierung
 - Objekte
 - Klassen als Referenzdatentyp
 - Geschachtelte Klassen
 - Elementklassen
 - Klassenvariablen

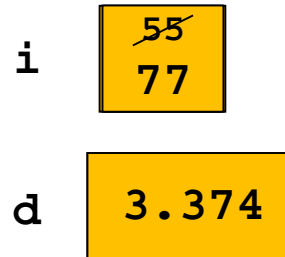
- Mit den bisher bekannten Mitteln ist es schwer große Datensammlungen anzulegen
- Beispiele hierfür sind:
 - Personaldateien
 - Stücklisten
 - Wetterdaten
 - Terminkalender

Bisher: elementare Datentypen, statisch im Speicher

im Programm

```
int i;
double d;
i = 55;
d = 3.374;
i = 77;
```

symbolisch notiert



im Speicher

Adresse

Inhalt

	≡
i: 133	55 77
	≡
d: 214	3.374
	≡

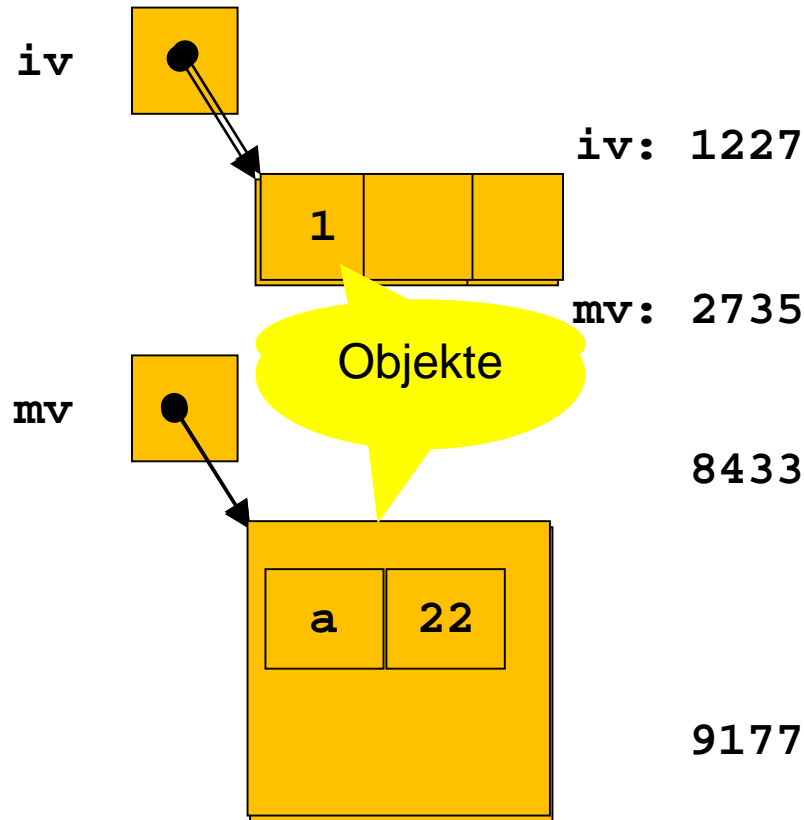
Jetzt: eigene Datentypen, dynamisch im Speicher → **Referenzdatentypen**

im Programm

symbolisch notiert

im Speicher

```
int[] iv;
MeinTyp mv;
iv = new int[3];
mv = new MeinTyp();
iv[0] = 1;
mv.a = 22;
```



Adresse

Inhalt

≡
8433
≡
9177
≡
1
≡
≡
22
≡

8433

9177

iv: 1227

mv: 2735

- Felder sind Objekte mit mehreren **Komponenten gleichen** Typs
- Die Komponenten eines Feldes haben keinen Namen, man greift über den Feldnamen und in `[` und `]` geklammerte **Indizes** auf sie zu
- **Beispiel:** Berechnung der Varianz von n einzulesenden Werten x_1, x_2, \dots, x_n (z.B. für $n = 100$):

Mittelwert

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i = \frac{1}{n} \cdot (x_1 + x_2 + \dots + x_n)$$

Varianz

$$v = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 = \frac{1}{n} \cdot ((x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \dots + (x_n - \bar{x})^2)$$

```
import java.util.Scanner;
public class Varianz1 {
    public static void main(String[] args){
        int n = 16;
        double x1, x2, x3, x4, x5, x6, x7, x8,
               x9, x10, x11, x12, x13, x14, x15, x16;
        double mw, va;
        Scanner input = new Scanner(System.in);
        System.out.println(n + " Werte eingeben:");
        x1 = input.nextDouble();
        x2 = input.nextDouble();
        x3 = input.nextDouble();
        x4 = input.nextDouble();
        x5 = input.nextDouble();
        x6 = input.nextDouble();
        x7 = input.nextDouble();
        x8 = input.nextDouble();
        x9 = input.nextDouble();
        x10 = input.nextDouble();
        x11 = input.nextDouble();
        x12 = input.nextDouble();
        x13 = input.nextDouble();
        x14 = input.nextDouble();
        x15 = input.nextDouble();
        x16 = input.nextDouble();
        mw = (x1 + x2 + x3 + x4 +
              x5 + x6 + x7 + x8 +
              x9 + x10 + x11 + x12 +
              x13 + x14 + x15 + x16) / n;
        va = Math.pow(x1-mw,2) + Math.pow(x2-mw,2) +
              Math.pow(x3-mw,2) + Math.pow(x4-mw,2) +
              Math.pow(x5-mw,2) + Math.pow(x6-mw,2) +
              Math.pow(x7-mw,2) + Math.pow(x8-mw,2) +
              Math.pow(x9-mw,2) + Math.pow(x10-mw,2) +
              Math.pow(x11-mw,2) + Math.pow(x12-mw,2) +
              Math.pow(x13-mw,2) + Math.pow(x14-mw,2) +
              Math.pow(x15-mw,2) + Math.pow(x16-mw,2);
        va = va / n;
        System.out.println("Varianz: " + va);
    }
}
```

```
import java.util.Scanner;
public class Varianz2 {
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        System.out.print("Wieviele Werte? n = ");
        int n = input.nextInt();
        double[] x = new double[n];

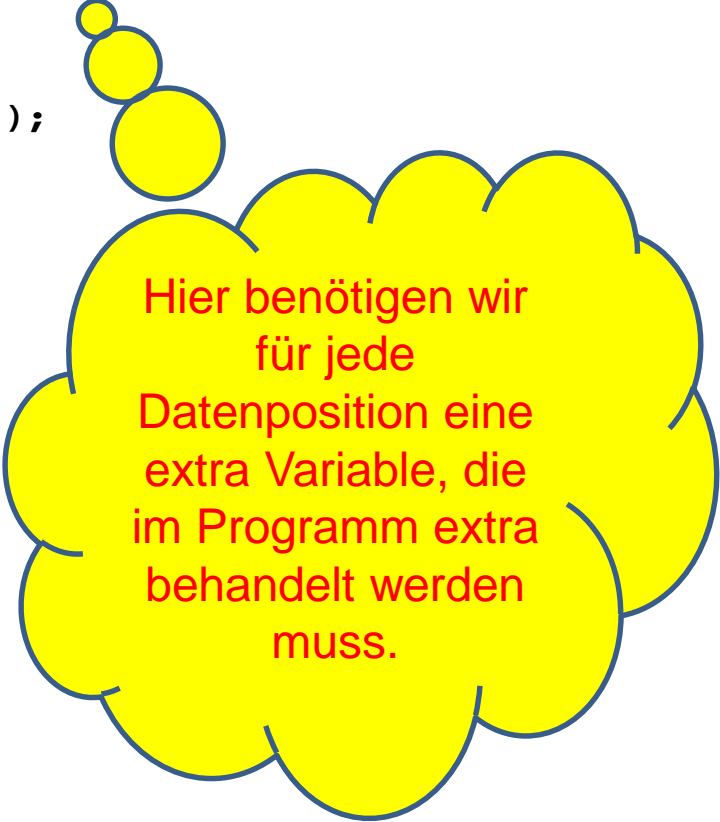
        double mw, va;
        for (int i=0; i < n; i++) {
            System.out.print("x[" + i + "] = ");
            x[i] = input.nextDouble();
        }

        mw = 0;
        for (int i=0; i < n; i++)
            mw = mw + x[i];
        mw = mw / n;
        va = 0;
        for (int i=0; i < n; i++)
            va = va + Math.pow(x[i]-mw,2);

        va = va / n;
        System.out.println("Varianz: " + va);
    }
}
```

Java


```
import java.util.Scanner;
public class Varianz1 {
    public static void main(String[] args){
        int n = 16;
        double x1, x2, x3, x4, x5, x6, x7, x8,
               x9, x10, x11, x12, x13, x14, x15, x16;
        double mw, va;
        Scanner input = new Scanner(System.in);
        System.out.println (n + " Werte eingeben:");
        x1 = input.nextDouble();
        x2 = input.nextDouble();
        x3 = input.nextDouble();
        x4 = input.nextDouble();
        x5 = input.nextDouble();
        x6 = input.nextDouble();
        x7 = input.nextDouble();
        x8 = input.nextDouble();
        x9 = input.nextDouble();
        x10= input.nextDouble();
        x11= input.nextDouble();
        x12= input.nextDouble();
        x13= input.nextDouble();
```



Hier benötigen wir
für jede
Datenposition eine
extra Variable, die
im Programm extra
behandelt werden
muss.

```
x14= input.nextDouble();
x15= input.nextDouble();
x16= input.nextDouble();
mw = (x1  + x2  + x3  + x4  +
      x5  + x6  + x7  + x8  +
      x9  + x10 + x11 + x12 +
      x13 + x14 + x15 + x16) / n;
va = Math.pow(x1-mw,2)  + Math.pow(x2-mw,2)  +
      Math.pow(x3-mw,2)  + Math.pow(x4-mw,2)  +
      Math.pow(x5-mw,2)  + Math.pow(x6-mw,2)  +
      Math.pow(x7-mw,2)  + Math.pow(x8-mw,2)  +
      Math.pow(x9-mw,2)  + Math.pow(x10-mw,2) +
      Math.pow(x11-mw,2) + Math.pow(x12-mw,2) +
      Math.pow(x13-mw,2) + Math.pow(x14-mw,2) +
      Math.pow(x15-mw,2) + Math.pow(x16-mw,2);
va = va / n;
System.out.println("Varianz: " + va);
}
```

```
import java.util.Scanner;

public class Varianz2 {
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        System.out.println("Wieviele Werte? n = ");
        int n = input.nextInt();
        double[] x = new double[n];

        double mw, va;
        for (int i=0; i < n; i++)
            x[i] = input.nextDouble();
        mw = 0;
        for (int i=0; i < n; i++)
            mw = mw + x[i];
        mw = mw / n;
        va = 0;
        for (int i=0; i < n; i++)
            va = va + Math.pow(x[i]-mw,2);
        va = va / n;
        System.out.println("Varianz: " + va);
    }
}
```

Im Feld sind alle Daten
enthalten und erlauben
eine einheitliche
Behandlung im
Programm.

- **Def: Feld** (Array)
 - beschränkte Sammlung von Elementen **identischen Typs**
 - Feldelemente sind indiziert (beginnend mit **0** bis zu einer oberen **Grenze**)
- Typ und Grenze sind in der Deklaration bzw. beim Einrichten des Feldes zu spezifizieren

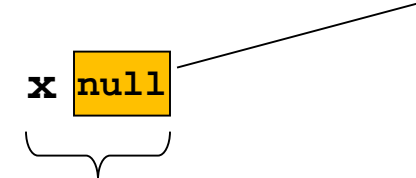
- Eine Feldvariable (Variable eines Feldtyps) wird deklariert durch Angabe des Komponententyps gefolgt von leeren eckigen Klammern und dem Namen (Bezeichner) der Variable

Beispiele:

```
double[] x;
```

null-Referenz

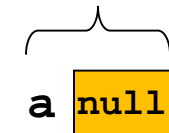
x null



Referenzvariable

```
int[] a;
```

a null



```
boolean[] z;
```

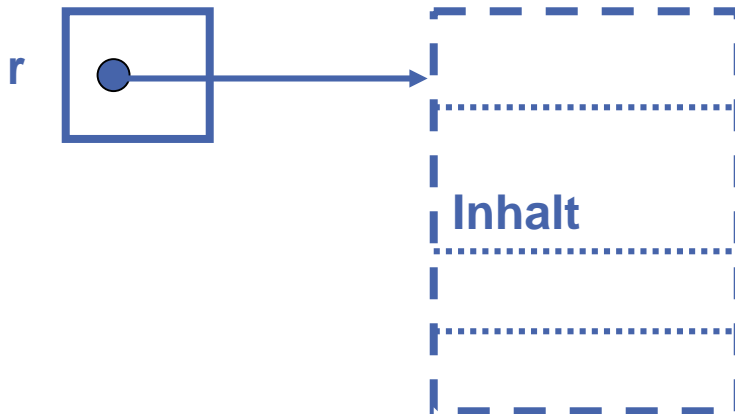
z null



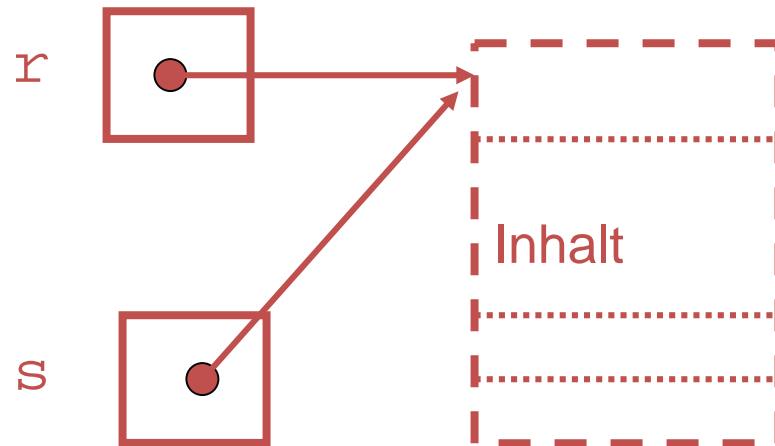
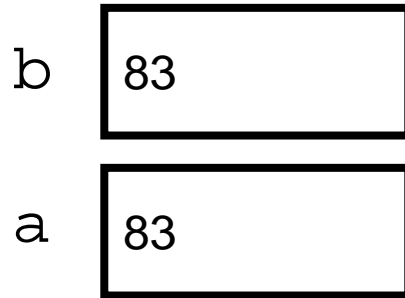
Graphische Notation für Variablen **einfacher Typen**:



Graphische Notation für Variablen von **Referenztypen**:



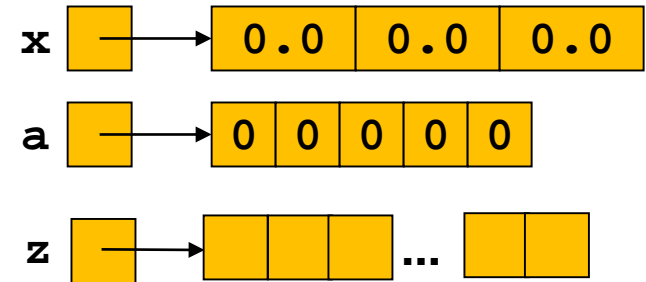
- Zwei Variablen einfachen Typs sind zu einem Zeitpunkt t gleich, wenn sich ihr Inhalt zum Zeitpunkt t gleich ($b == a$).



Zwei Referenzvariablen sind gleich, wenn die Referenzen (d.h. bei der visuellen Darstellung die Pfeile) die gleichen Ziele haben ($r == s$)

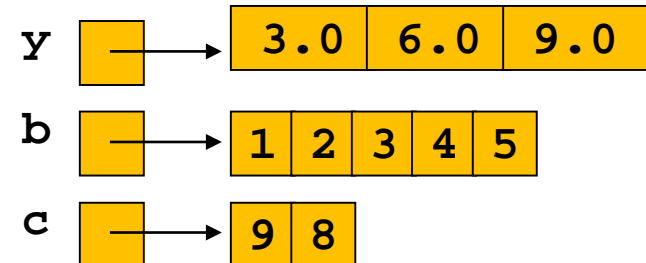
- **Dynamisch:** Ein Feld wird erzeugt durch den **new-Operator** gefolgt vom Komponententyp und in eckigen Klammern eingeschlossenen Längenangaben

```
x = new double[3];  
a = new int[5];  
System.out.print("n = ");  
int n = input.nextInt() // java.util.Scanner  
z = new boolean[n];
```



- **Statisch:** Ein Feld wird **in Verbindung mit seiner Deklaration** erzeugt durch Aufzählung seiner Komponenten in geschweiften Klammern

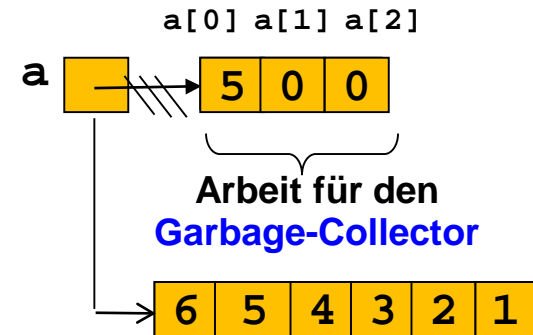
```
double[] y = {3.0,6.0,9.0};  
int[] b = {1,2,3,4,5};  
int[] c = new int[] {9,8};
```



- Ein Feld der Länge n hat n Komponenten, die erste hat den Index 0 und die letzte den Index $n-1$
- Der Zugriff auf die Komponenten erfolgt über mit $[$ und $]$ geklammerten Index-Ausdrücke

Beispiele:

```
a = new int[3];  
a[0] = 5;  
a = new int[6];  
for (int i=0; i<6; i++)  
    a[6-i-1] = i+1;
```



- Ein Zugriff außerhalb der Indexgrenzen verursacht einen Laufzeitfehler: **ArrayIndexOutOfBoundsException**
z.B. wäre `a[-1]` oder `a[100]` unzulässig

- Zusätzlich zu den Feldkomponenten besitzt ein Feldobjekt eine spezielle Komponente (Instanz-Variable), die jeweils die aktuelle Feldlänge enthält. Sie heißt **length** und wird mit einem **.** angesprochen

- Beispiel:

```
// Variable „input“ ist vom Typ java.util.Scanner
System.out.print("Länge von x:");
int n = input.nextInt();
double[] x = new double[n];
System.out.print("Länge von y:");
n = input.nextInt();
double[] y = new double[n];

// Jetzt die Komponenten einlesen:
for (int i=0; i<x.length; i++) {
    System.out.print("x[" + i + "] = ");
    x[i] = input.nextDouble();
}
for (int i=0; i<y.length; i++) {
    System.out.print("y[" + i + "] = ");
    y[i] = input.nextDouble();
}
```

Deklaration eines Array (Syntax)

```
<Typ> [ ] <Feldname> = new <Typ> [ <Grenze> ] ;
```

```
<Typ> [ ] <Feldname> ; // eigentliche Deklaration  
<Feldname> = new <Typ> [ <Grenze> ] ; // Einrichten des Feldes
```

Grenze bezeichnet man auch als Feldlänge

<Typ> → Typ oder Klasse

Indizierung der Feldelemente läuft von 0 bis **<Grenze>-1**

Deklaration eines Array (Syntax)

```
<Typ> [ ] <Feldname> = new <Typ> [ <Grenze> ] ;
```

```
<Typ> [ ] <Feldname> ;
```

```
<Feldname> = new <Typ> [ <Grenze> ] ; // eigene Initialisierung
```

```
<Typ> [ ] <Feldname> = { <Werte> } ;
```

**Dynamische
Initialisierung**

**Statische
Initialisierung
Feld-Initialisierer
(array initializer)**

<Typ> → Typ oder Klasse

Indizierung der Feldelemente läuft von 0 bis **<Grenze>**-1

<Werte> ist eine Folge

w_0, w_1, \dots, w_{k-1} von Anfangswerten

Die Indexgrenzen erhält man aus deren Anzahl **k**

- Welche Fehler sollte man vermeiden?
- Typische Fehler sind:
 - fehlende Initialisierung
 - fehlerhafte Zuweisungen (Wert vs. Referenz)
 - Über- oder Unterschreitung der Feldgrenzen
 - Verwendung falscher Typen

```
int[] feld;
```

```
//feld[0] = 1;
```

```
//System.out.println(feld);
```

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

```
feld[0] = 0;
```

```
System.out.println(feld[0]);
```

```
if (feld[0]==0)
```

```
    System.out.println("feld[0] = " + feld[0]);
```

```
System.out.println("feld = " + feld);
```

Variable feld might
not have been
initialized

Variable feld might
not have been
initialized

Adresse:
[I@1a1c887

Wo liegt hier das Problem?

```
int max = 4;  
int[] feld;  
feld = new int[max];  
for (int i=0; i <= max; i++){  
    feld[i] = i;  
}
```

Laufzeitfehler:
`java.lang.ArrayIndexOut
OfBoundsException`

< geht
besser ist
`i < feld.length`

```
double[] b;
```

```
int j; // Geht hier long?
```



Nein!

```
j = 17;
```

```
b = new double[j];
```

```
b[15] = 3.14;
```

```
System.out.println("b[15]= " + b[15]);
```


// Was passiert, wenn man byte durch int ersetzt?

```
byte[] a;  
int i = 2;  
for (int j = 2; j<=25; j++)  
    i = i*2;  
System.out.println(i);  
a = new byte[i];
```



33554432

// Was passiert, wenn man byte durch int ersetzt?

```
int[] a;
```

```
int i = 2;
```

```
for (int j = 2; j<=25; j++)// versuche 26
```

```
    i = i*2;
```

```
System.out.println(i);
```

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

33554432

Laufzeitfehler:
`java.lang.OutOfMemoryError`

Die genauen Werte können mit der aktuellen
Speichergröße differieren!

- **Größe des Feldes**
 - durch die Speicherkapazität des Rechners und die Größe des Komponententyps beschränkt
 - fixiert, wenn das Feld angelegt wird, jedoch noch **nicht nach der Deklaration!**
 - kann nach dem **Einrichten des Feldes nicht verändert** werden
- bei der Erzeugung eines Feldes wird ein Verweis auf den Speicherbereich angelegt, in welchem die Feldelemente gespeichert werden
- Man beachte jedoch die Möglichkeit, einer **Variable** Felder **unterschiedlicher Länge** zuzuweisen

- Hier werden der Variable `a` Felder unterschiedlicher Länge zugewiesen!

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

```
int[] b;
```

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

```
a = b;
```

Feldlänge 3

Feldlänge 5

Feldlänge 5

?

Der Zuweisungsoperator kann für ganze Felder benutzt werden. Allerdings wird dabei nur die Referenz kopiert! Separate Kopien erzeugt man durch *cloning* (siehe später).

Was wird ausgegeben?

- Beim Vergleichen und Kopieren von Feldern muss Referenzstruktur beachtet werden!

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

```
int[] b = {1, 2, 3, 4};
```

```
int[] c = a;
```

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

```
for (int i=0; i<a.length-1; i++)  
    d[i] = a[i];
```

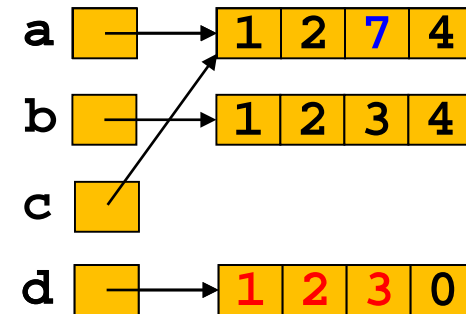
```
a[2] = 7;
```

```
System.out.println(a == b);
```

```
System.out.println(a == c);
```

```
System.out.println(a == d);
```

```
System.out.println(b[2] + " " + c[2] + " " + d[3]);
```



Ausgabe:

false

true

false

3 7 0

Semantik von $\langle \text{Feld} \rangle [\langle \text{Ausdruck1} \rangle] = \langle \text{Ausdruck2} \rangle ;$

$\langle \text{Ausdruck1} \rangle$ auswerten liefert Wert1

// beachte Inkremente aus Ausdruck1

$\langle \text{Ausdruck2} \rangle$ auswerten liefert Wert2

Zuweisung $\langle \text{Feld} \rangle [\text{Wert1}] = \text{Wert2};$ ausführen

// beachte Inkremente aus Ausdruck2

Was passiert hier?

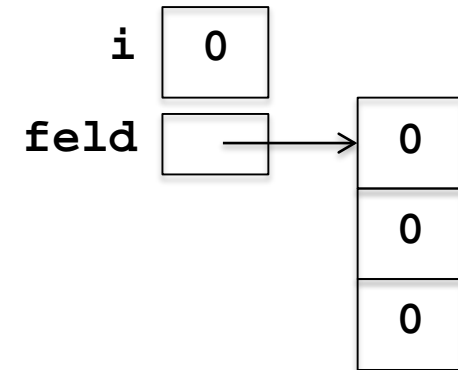
```
int i = 0;  
int [] feld = {0,0,0};  
feld[++i + i++] = i++ + ++i;  
System.out.println(feld[0]);  
System.out.println(feld[1]);  
System.out.println(feld[2]);  
System.out.println(i);
```

```
int i = 0;  
int[] feld = {0,0,0};  
feld[++i + i++] = i++ + ++i;  
System.out.println(feld[0]);  
System.out.println(feld[1]);  
System.out.println(feld[2]);  
System.out.println(i);
```

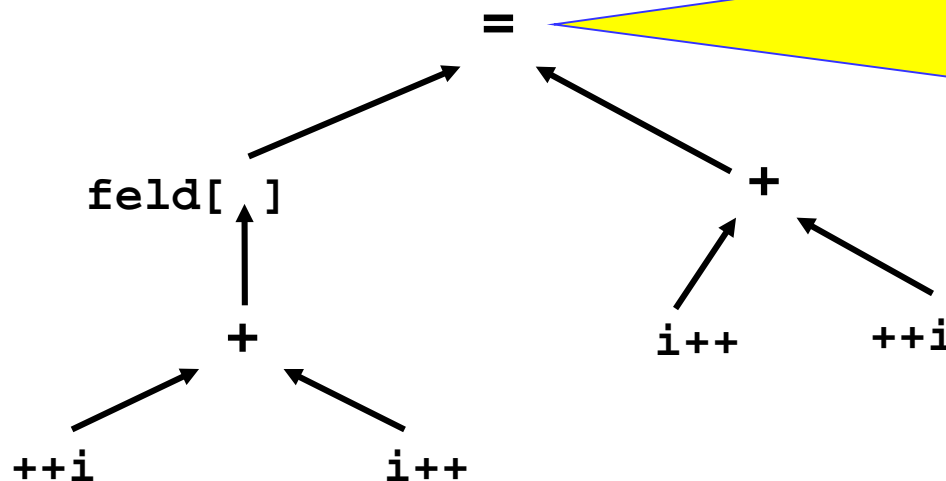
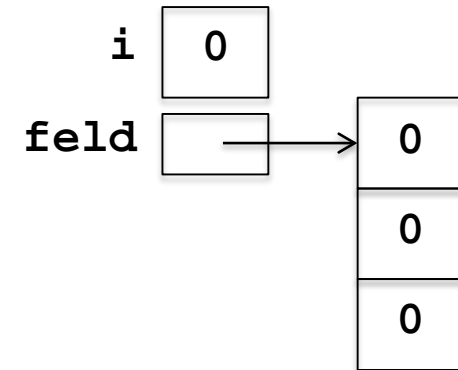
i

0

```
int i = 0;  
int[] feld = {0,0,0};  
feld[++i + i++] = i++ + ++i;  
System.out.println(feld[0]);  
System.out.println(feld[1]);  
System.out.println(feld[2]);  
System.out.println(i);
```

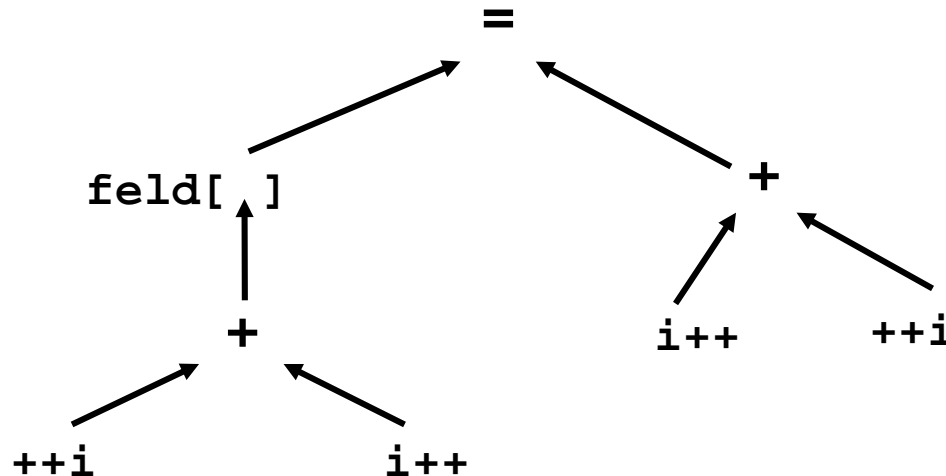
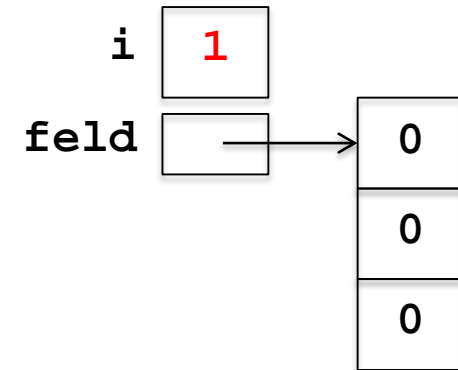


```
int i = 0;
int[] feld = {0,0,0};
feld[++i + i++] = i++ + ++i;
System.out.println(feld[0]);
System.out.println(feld[1]);
System.out.println(feld[2]);
System.out.println(i);
```

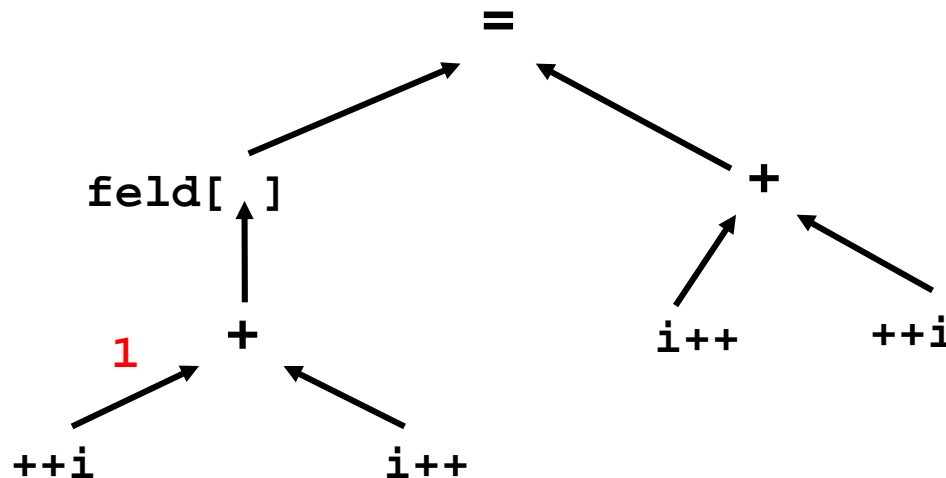
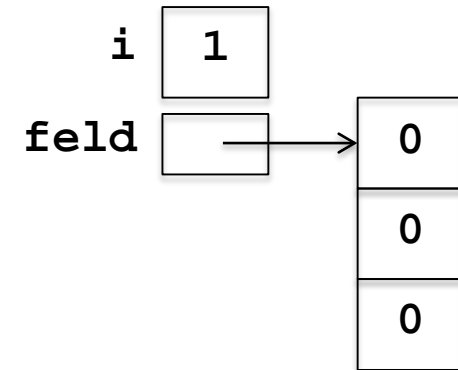


Hinweis: Der dem Ausdruck zugeordnete Baum wird in einem **Postorder-Durchlauf** (Linker Teilbaum, Rechter Teilbaum, Wurzel: Abkürzung **LRW**) durchlaufen. (siehe später)

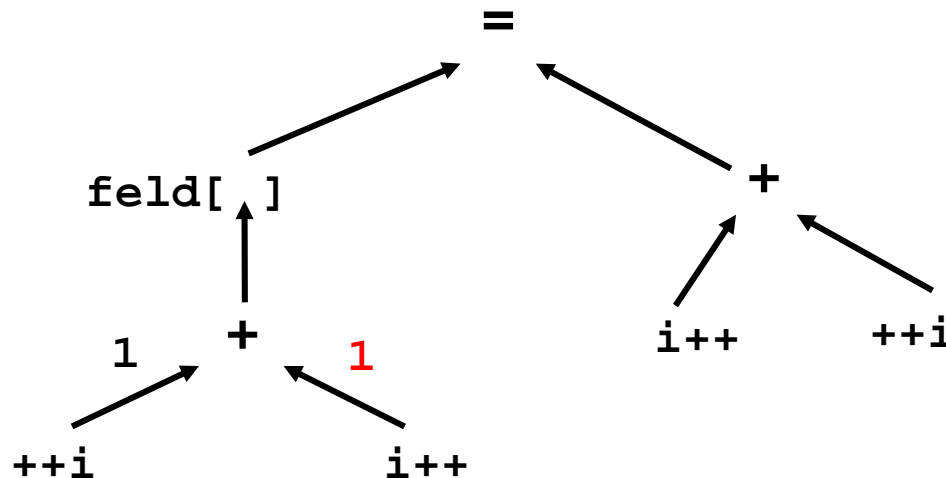
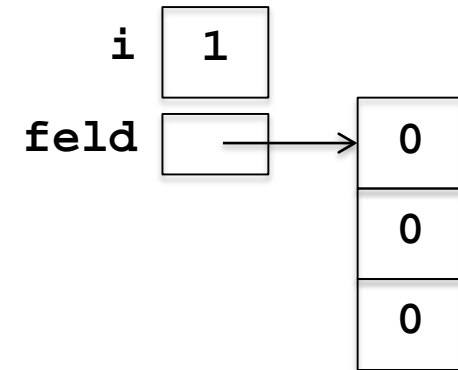
```
int i = 0;  
int[] feld = {0,0,0};  
feld[++i + i++] = i++ + ++i;  
System.out.println(feld[0]);  
System.out.println(feld[1]);  
System.out.println(feld[2]);  
System.out.println(i);
```



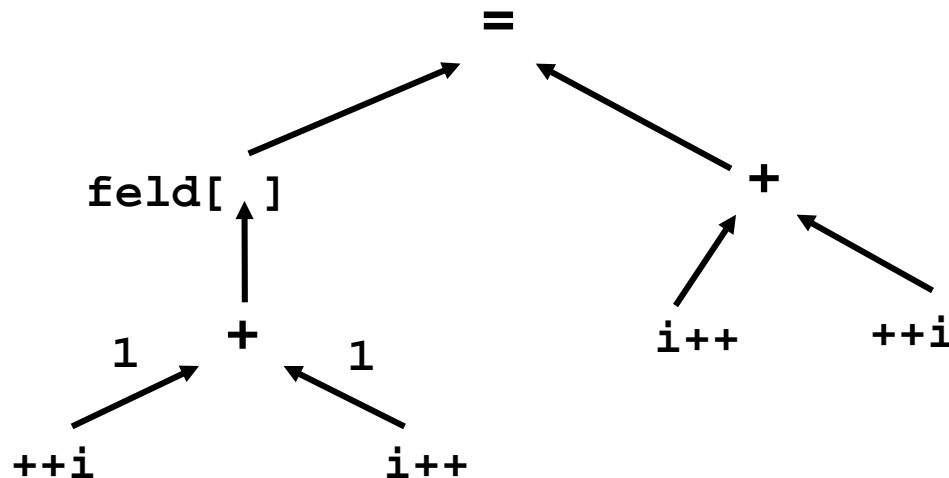
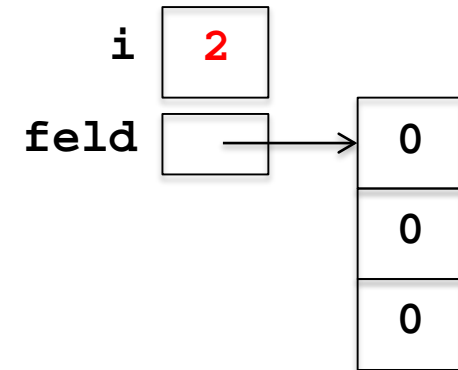
```
int i = 0;  
int[] feld = {0,0,0};  
feld[++i + i++] = i++ + ++i;  
System.out.println(feld[0]);  
System.out.println(feld[1]);  
System.out.println(feld[2]);  
System.out.println(i);
```



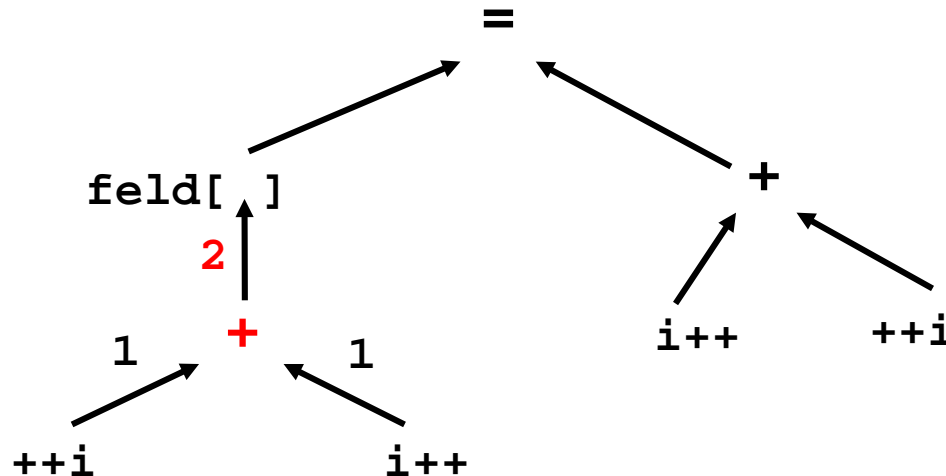
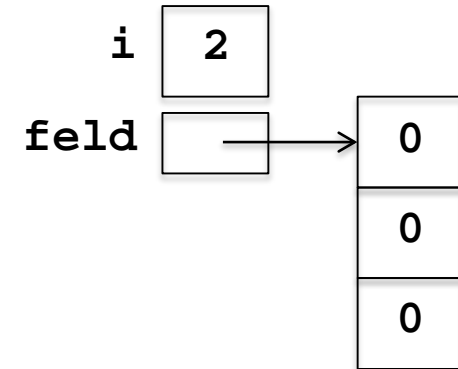
```
int i = 0;  
int[] field = {0,0,0};  
field[++i + i++] = i++ + ++i;  
System.out.println(field[0]);  
System.out.println(field[1]);  
System.out.println(field[2]);  
System.out.println(i);
```



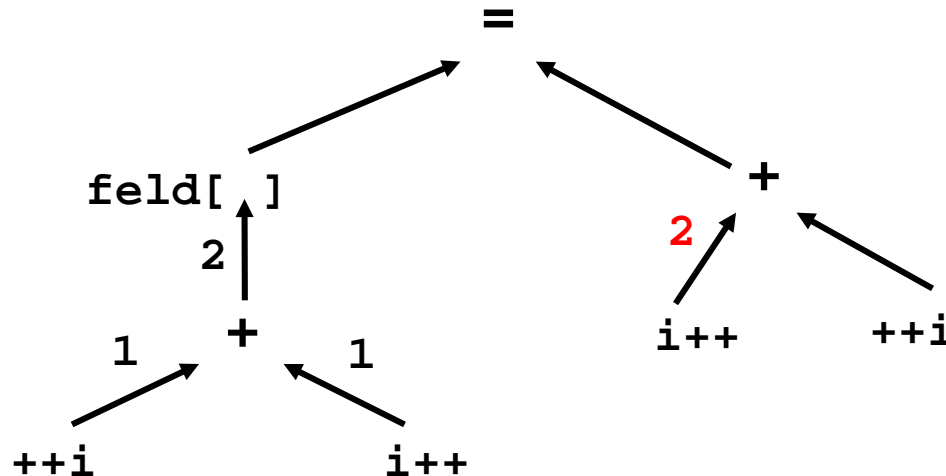
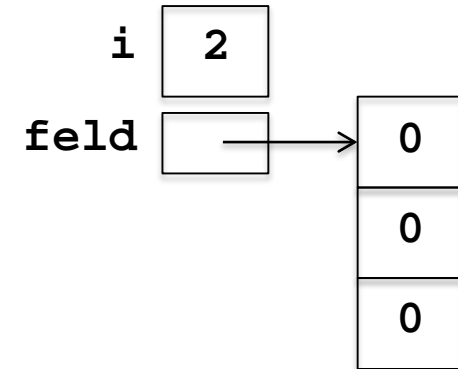
```
int i = 0;
int[] feld = {0,0,0};
feld[++i + i++] = i++ + ++i;
System.out.println(feld[0]);
System.out.println(feld[1]);
System.out.println(feld[2]);
System.out.println(i);
```



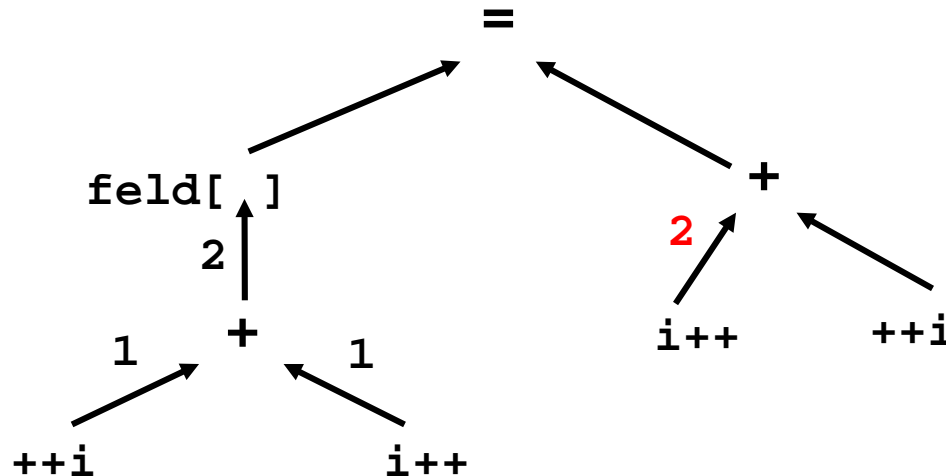
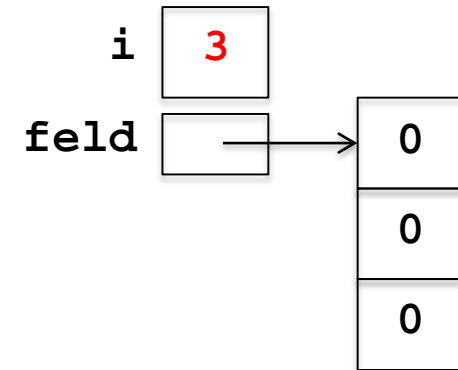
```
int i = 0;
int[] feld = {0,0,0};
feld[++i + i++] = i++ + ++i;
System.out.println(feld[0]);
System.out.println(feld[1]);
System.out.println(feld[2]);
System.out.println(i);
```



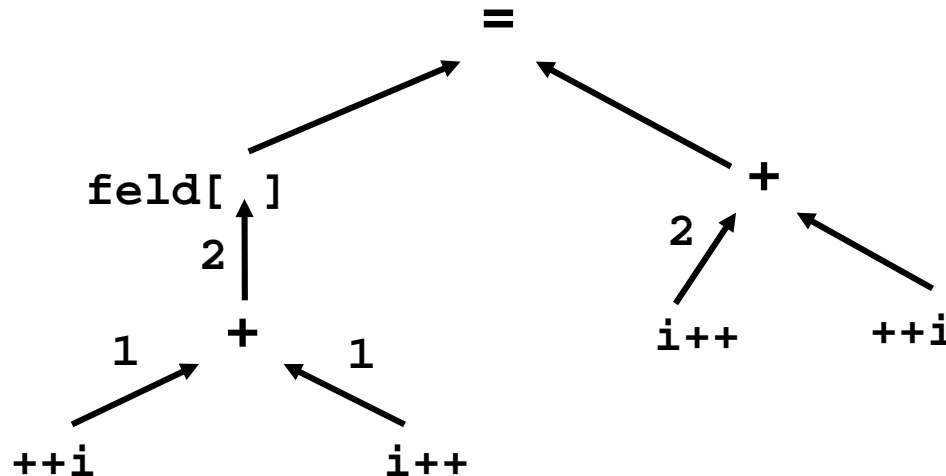
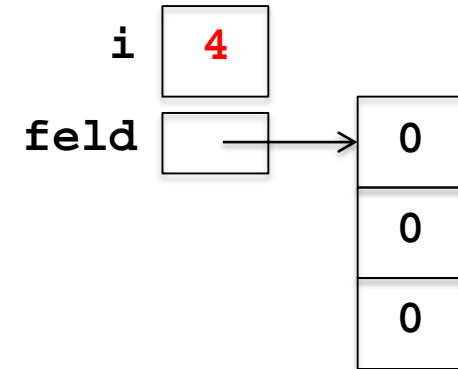
```
int i = 0;
int[] field = {0,0,0};
field[++i + i++] = i++ + ++i;
System.out.println(field[0]);
System.out.println(field[1]);
System.out.println(field[2]);
System.out.println(i);
```



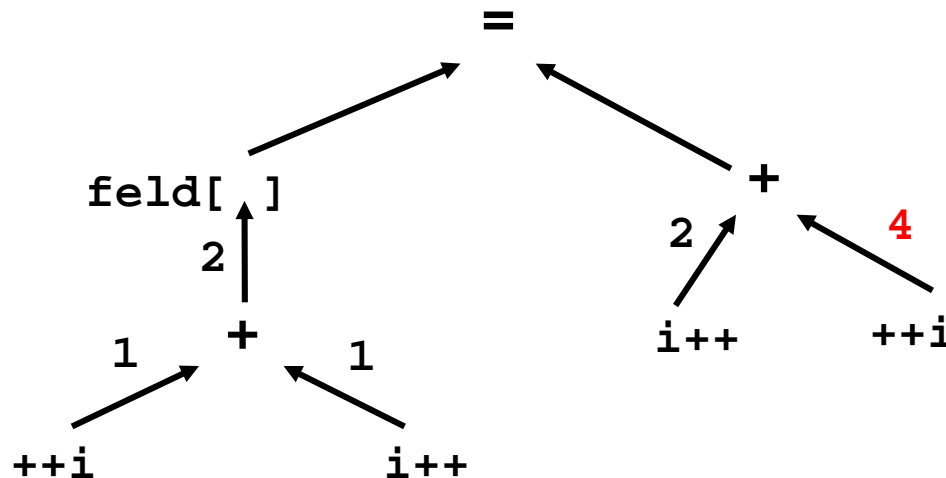
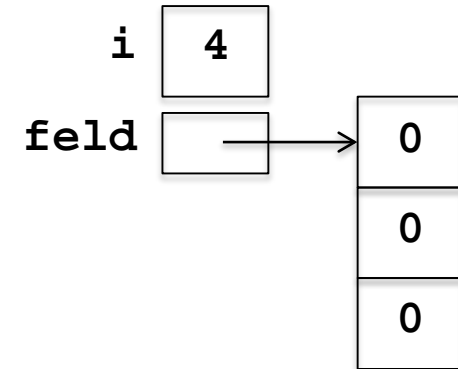

```
int i = 0;  
int[] feld = {0,0,0};  
feld[++i + i++] = i++ + ++i;  
System.out.println(feld[0]);  
System.out.println(feld[1]);  
System.out.println(feld[2]);  
System.out.println(i);
```



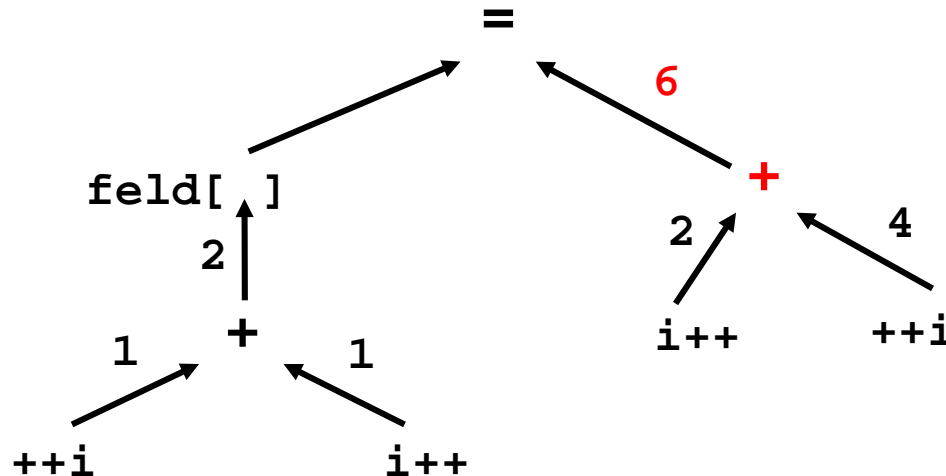
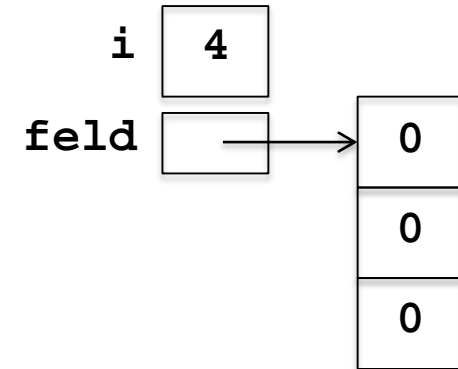
```
int i = 0;  
int[] feld = {0,0,0};  
feld[++i + i++] = i++ + ++i;  
System.out.println(feld[0]);  
System.out.println(feld[1]);  
System.out.println(feld[2]);  
System.out.println(i);
```



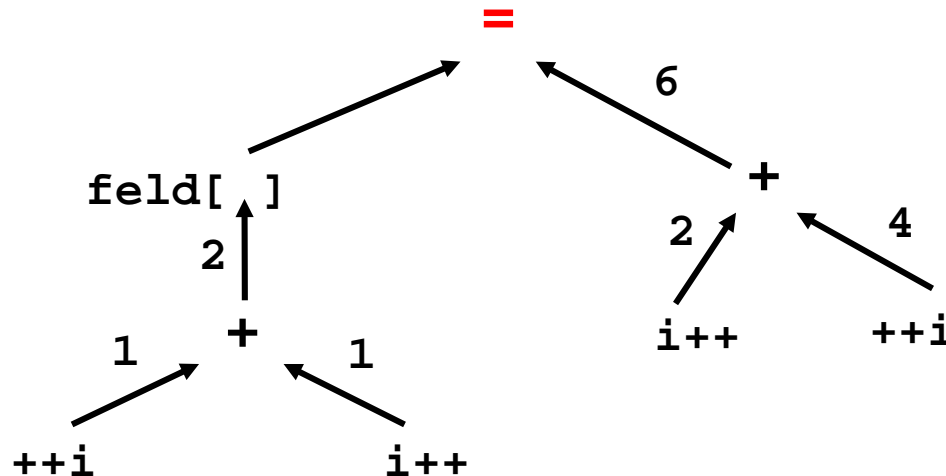
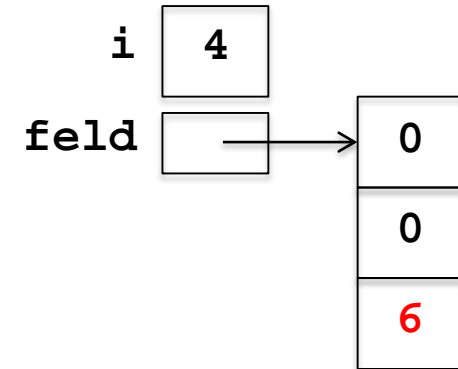
```
int i = 0;
int[] feld = {0,0,0};
feld[++i + i++] = i++ + ++i;
System.out.println(feld[0]);
System.out.println(feld[1]);
System.out.println(feld[2]);
System.out.println(i);
```



```
int i = 0;
int[] feld = {0,0,0};
feld[++i + i++] = i++ + ++i;
System.out.println(feld[0]);
System.out.println(feld[1]);
System.out.println(feld[2]);
System.out.println(i);
```



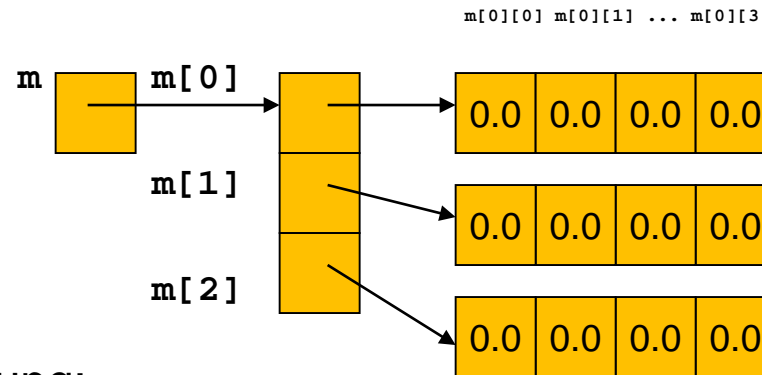
```
int i = 0;
int[] feld = {0,0,0};
feld[++i + i++] = i++ + ++i;
System.out.println(feld[0]);
System.out.println(feld[1]);
System.out.println(feld[2]);
System.out.println(i);
```



- Beispiel "zweidimensionales rechteckiges Feld"

```
double[][] m = new double[3][4];
```

"Feld mit Komponententyp **Feld mit Komponententyp double**"



Alternative Erzeugung:

```
double[][] m;  
m = new double[3][];  
for (int i=0; i<3; i++)  
    m[i] = new double[4];
```

Wichtig: In beiden Fällen gilt

`m.length` ist 3

`m[i].length` ist 4

Initialisierung bei der Deklaration

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

Allgemeine Form:

```
<Typ>[ ][ ]...[ ] <Feldname> = new <Typ>[a1]...[ak][ ][ ]...[ ];
```

<Typ> - beliebiger Typ

<Feldname> - Bezeichner

a₁, ..., a_k - ganzzahlige Ausdrücke, k > 0

links und rechts muss die gleiche Anzahl eckiger Klammern stehen

Zweite Variante:

```
<Typ>[ ][ ]...[ ] <Feldname>;
```

```
<Feldname> = new <Typ>[a1]...[ak][ ][ ]...[ ];
```

Beachten Sie:

Bei einem mehrdimensionalen Feld müssen immer
zuerst die ersten Dimensionen initialisiert werden.

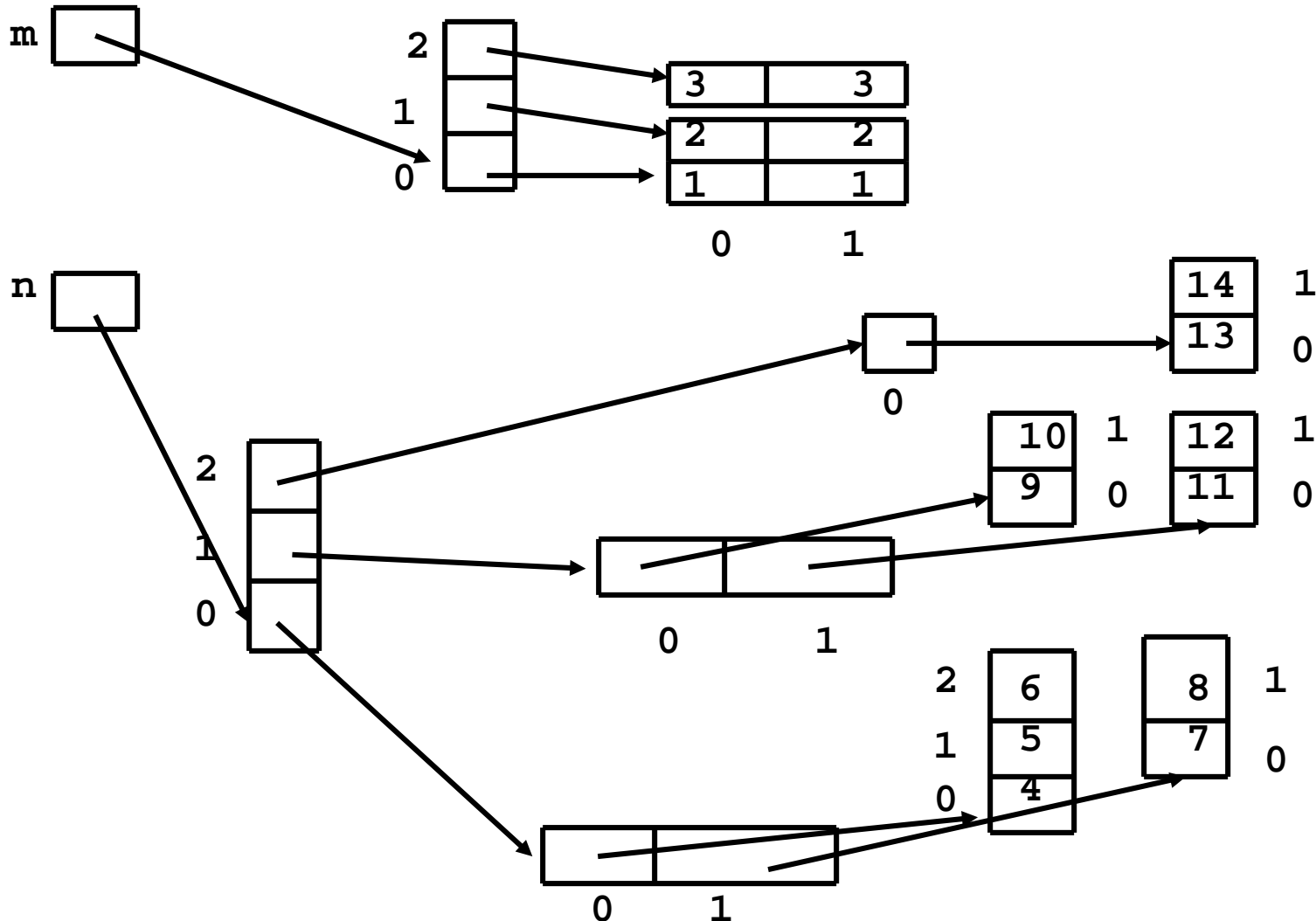
```
int[][][] big = new int[][][2]; // Fehler!
```



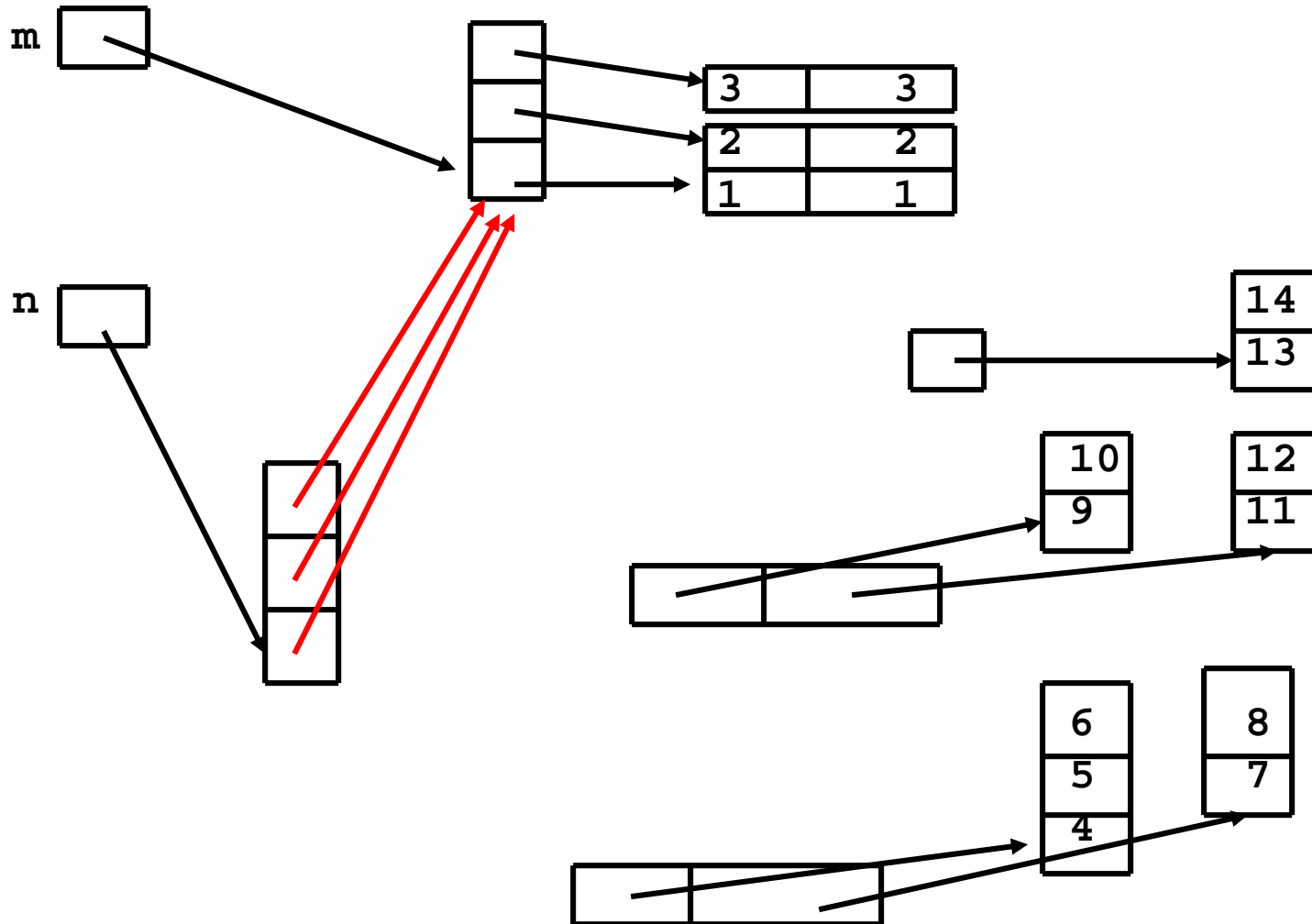
```
int[][] m = {{1,1},{2,2},{3,3}};  
int[][][] n = {{{4,5,6},{7,8}}, {{9,10},{11,12}},  
               {{13,14}}};
```

```
n[0] = m;  
n[1] = m;  
n[2] = m;  
// int[] k = {1,2,3};  
// k = m; nicht erlaubt: incompatible types  
m[0] = m[2];  
m[1] = m[2];  
n[2][1][1] = 17;
```

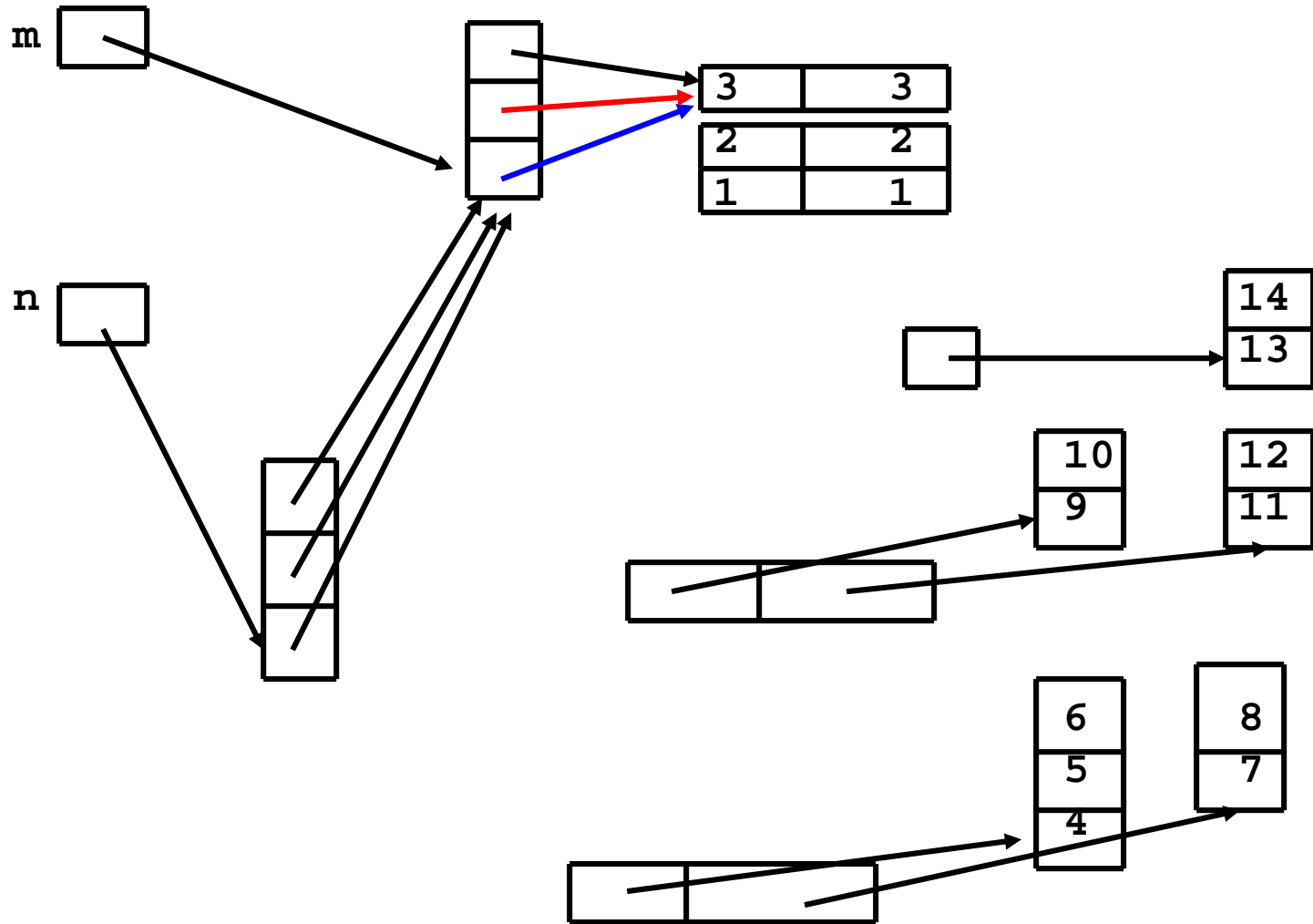
```
int[][] m = {{1,1},{2,2},{3,3}};
int[][][] n = {{{4,5,6},{7,8}},{{9,10},{11,12}},{{13,14}}};
```



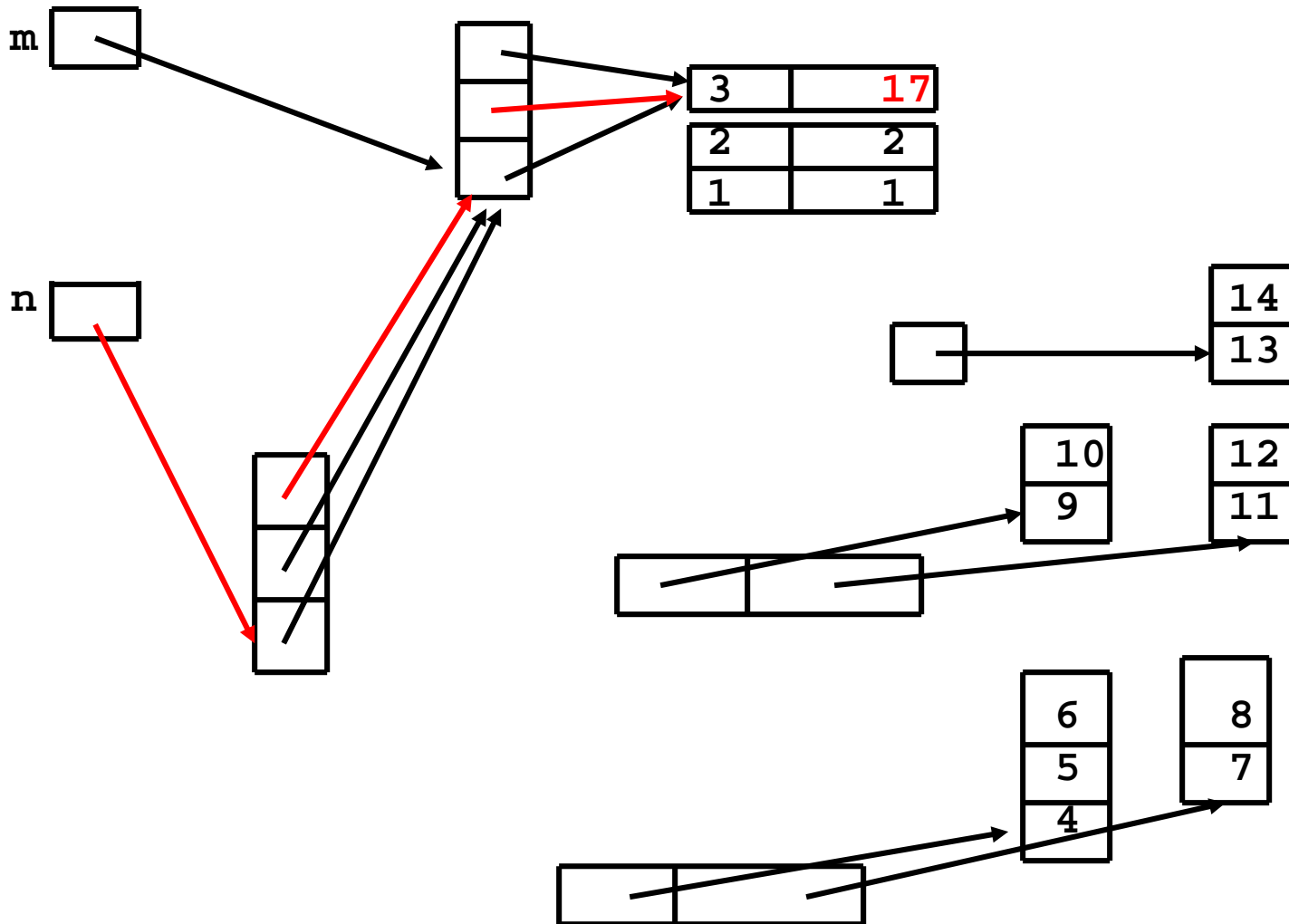
`n[0] = m; n[1] = m; n[2] = m;`



`m[0] = m[2]; m[1] = m[2];`



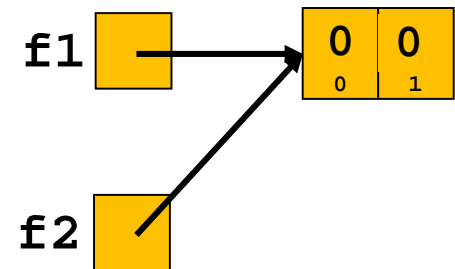
`n[2][1][1] = 17;`



➔ **Vorsicht beim Kopieren von Referenztypen!**

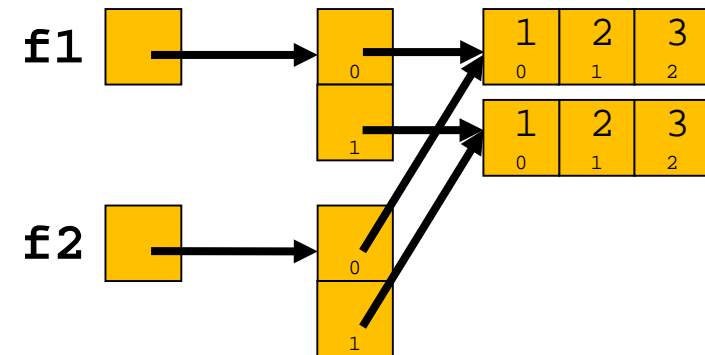
- Es werden stets nur die Referenzen kopiert. Werden andere Kopien erwünscht, z.B. **vollständige (tiefe)** Kopien (siehe nächste Folie), so muss **Cloning** benutzt (und eventuell / je nach Anwendungsfall selbstständig implementiert) werden!

- Man unterscheidet
 - **Referenzkopie**: Nur die Referenz wird kopiert (Kopieren des „Werts“ einer Referenzvariable in eine andere)
 - **Flache Kopie**: Lediglich die erste Ebene einer Struktur wird dupliziert
 - **Tiefe Kopie**: Alle Ebenen einer Struktur werden dupliziert. Dies nennt man auch **Clonen** (später)
- Anwendung auf Felder
 - **Referenzkopie**: durch die Anweisung von $\text{f2} = \text{f1}$ wird lediglich eine Kopie des „Wertes“ (Pfeil) von f1 erzeugt; eine Referenz auf dieselbe Struktur
 - Eine Veränderung von f2 bewirkt eine Veränderung von f1



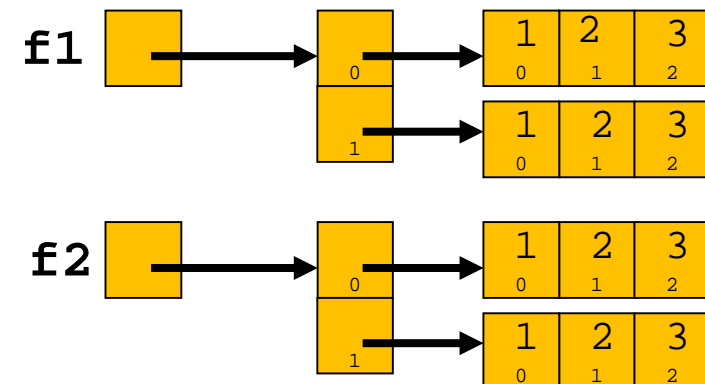
Flache Kopie

- Es wird lediglich ein Duplikat für die erste Ebene erstellt.
Alle weiteren Ebenen sind identisch
- Änderungen von der ersten Ebene von **f2** wirken sich nicht auf **f1** aus
- **Übung:** Wie muss der zugehörige Quelltext dazu aussehen?



Tiefe Kopie

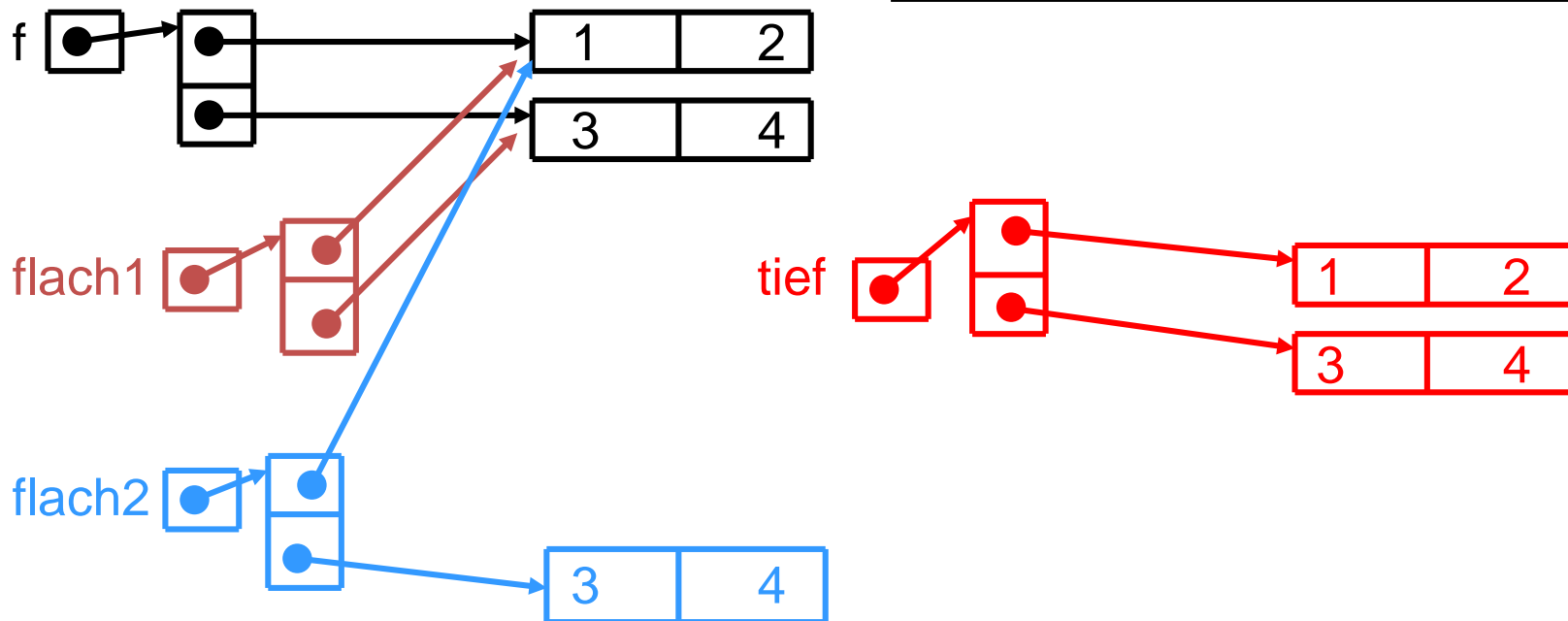
- Die komplette Struktur wird dupliziert.
f2 ist völlig unabhängig von **f1**
- **Übung:** Wie muss der zugehörige Quelltext dazu aussehen?



- Neben einer flachen Kopie (**flach1**) und einer tiefen Kopie (**tief**) existieren noch verschiedene Mischformen, hier z.B. **flach2**

Übung:

Überlegen Sie sich entsprechende Implementierungen zur Erzeugung **flacher** und **tiefer** Kopien.



■ Felder (arrays)

- Deklaration
- Erzeugung und Initialisierung
- Mehrdimensionale Felder
- Beispiele

■ Klassen (classes)

- Deklaration und Instanziierung
- Objekte
- Klassen als Referenzdatentyp
- Geschachtelte Klassen
- Elementklassen
- Klassenvariablen

- **Klassen** sind selbstdefinierte Mustervorlagen/Baupläne für einen Datentyp zur Erzeugung von **Objekten** mit mehreren **Komponenten unterschiedlichen** Typs
- Die **Komponenten** eines Objekts haben *eigene* Bezeichner, man greift über den *Objektnamen* (die Referenz) und einen **.** auf sie zu

- **Motivation**

Beispiel: Ein Programm soll die Geburtstage mehrerer Personen handhaben können. Ein **Geburtsdatum** soll aus:

dem Tag	als byte
dem Monat	als string
dem Jahr	als short

bestehen.

```
public class Family {  
    public static void main(String[] args) {  
        byte papaTag;  
        byte mamaTag;  
        byte kindTag;  
        String papaMonat;  
        String mamaMonat;  
        String kindMonat;  
        short papaJahr;  
        short mamaJahr;  
        short kindJahr;  
        papaTag = 5;  
        papaMonat = "November";  
        papaJahr = 1933;  
        mamaTag = 3;  
        mamaMonat = "Januar";  
        mamaJahr = 1945;  
        kindTag = 25;  
        kindMonat = "Mai";  
        kindJahr = 1970;  
    }  
}
```

```
public class Datum {  
    public byte tag;  
    public String monat;  
    public short jahr;  
}
```

```
public class Familie {  
    public static void main(String[] args){  
        Datum papa, mama, kind;  
        papa = new Datum();  
        papa.tag = 5;  
        papa.monat = "November";  
        papa.jahr = 1933;  
        mama = new Datum();  
        mama.tag = 3;  
        mama.monat = "Januar";  
        mama.jahr = 1945;  
        kind = new Datum();  
        kind.tag = 25;  
        kind.monat = "Mai";  
        kind.jahr = 1970;  
    }  
}
```

- **Klassen sind Referenzdatentypen!**
(Man erinnere sich an die entsprechende Diskussion bei Feldern)
- **Klassen können wie jeder andere Typ verwendet werden**
 - Man kann Felder von Klassen anlegen
 - Jeder Feldkomponente wird dabei ein **eigenes Objekt** zugewiesen

Durch Festlegung des Klassen-Namens und Angabe der Klassen-Komponenten (Attribute, Elemente)

Syntax:

Modifizierer-Folge
(kann entfallen)

class

Bezeichner {

Modifizierer-Folge
(kann entfallen)

⋮

Modifizierer-Folge
(kann entfallen)

Variablen-, Methoden- oder Konstruktor-
Deklaration

⋮

Variablen-, Methoden- oder Konstruktor-
Deklaration

}

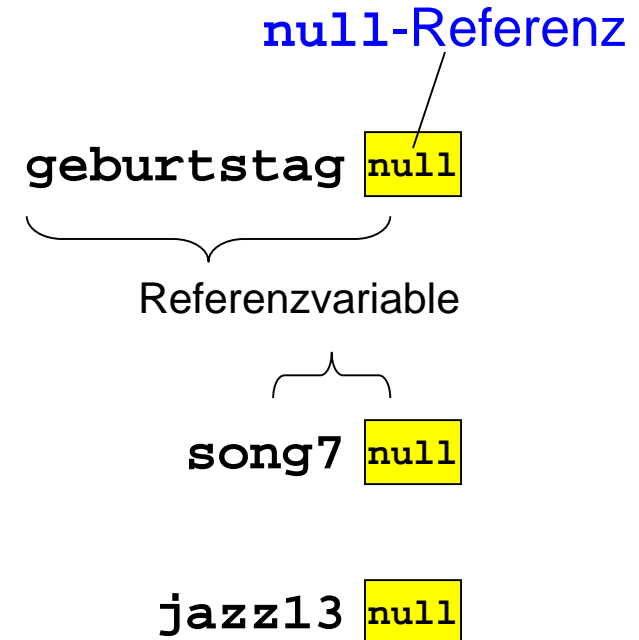
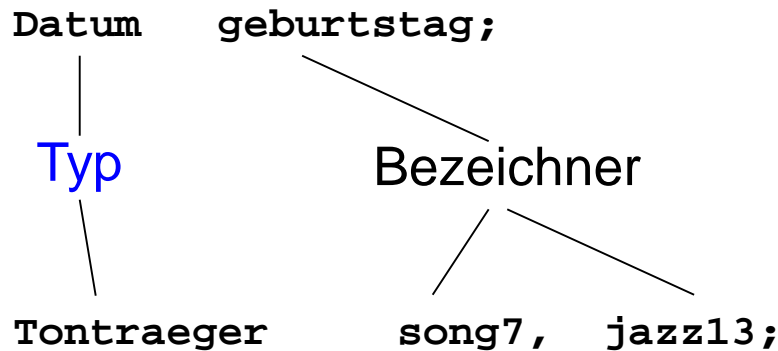
Beispiele:

```
public class Datum {  
    public byte tag;  
    public String monat;  
    public short jahr;  
}
```

```
public class Tontraeger {  
    public String medium,      // z.B. CD, DVD  
                stil,         // z.B. Klassik, Pop, Jazz  
                interpret,  
                titel;  
    public short  jahr;        // Erscheinungsjahr  
    public byte   bewertung;   // Noten von 1 bis 6  
}
```

Durch Angabe der **Klasse** (des **Datentyps**) gefolgt von einem Variablennamen

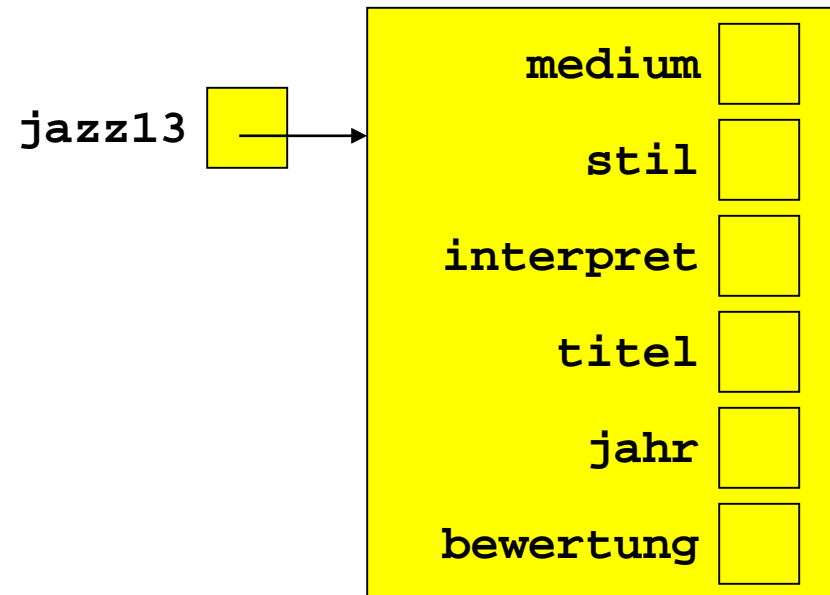
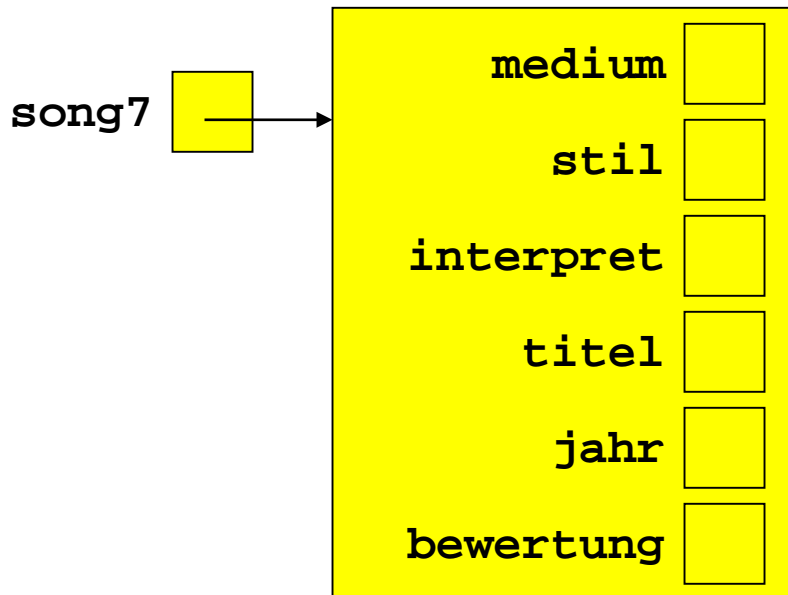
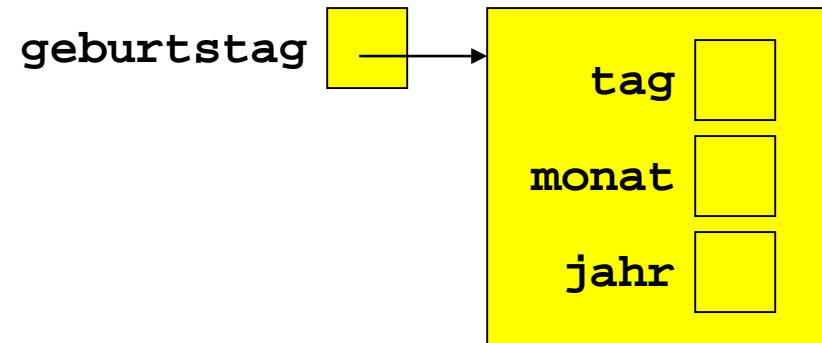
Beispiele:



- Durch den **new**-Operator gefolgt von einem **Konstruktoraufruf** (mehr dazu später) unter Verwendung des Klassennamens

- Beispiele:

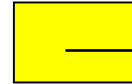
```
geburtstag = new Datum();
song7 = new Tontraeger();
jazz13 = new Tontraeger();
```



Verwendung der Punkt-Notation

Beispiele:

geburtstag



```
geburtstag.tag = 4;  
geburtstag.monat = "Dezember";  
geburtstag.jahr = 1960;
```

```
song7.medium = "CD";  
song7.stil = „Song“;  
song7.interpret = „Katie Melua“;  
song7.titel = „The closest thing to crazy“;  
song7.jahr = 2003;  
song7.bewertung = 3;
```

```
jazz13.medium = "CD";  
jazz13.stil = "Jazz";  
jazz13.interpret = „Brook Benton“;  
jazz13.titel = „Rainy Night in Georgia“;  
jazz13.jahr = 1995;  
jazz13.bewertung = 2;
```

- Klassen können deklariert werden
 - innerhalb einer anderen Klasse (**geschachtelte / innere Klasse**)
 - außerhalb einer Klasse, aber in eigenständiger Datei (**bevorzugt!**)
 - außerhalb einer Klasse, aber in der gleichen Datei
- Die eigenständigen Klassen/Dateien können im aktuellen Arbeitsverzeichnis oder in anderen Verzeichnissen liegen
- Der Compiler und der Interpreter suchen nach Bytecode-Dateien
 - im aktuellen Verzeichnis und
 - in den zip- und jar-Archiven der JRE (Java Runtime Environment)
- Weitere Suchpfade können über den Klassenpfad angegeben werden
 - durch Definition einer Umgebungsvariable CLASSPATH
 - durch Compilieren und Interpretieren mit der Option `-classpath` z.B.

```
javac -classpath .;D:\JavaTools MeineKlasse.java  
java -classpath .;D:\JavaTools MeineKlasse
```

```
public class <MAINCLASS> {  
  
    // Geschachtelte Klasse  
    public static class <KLASSENNAME> {  
        // hier die einzubindenden Variablen angeben  
    }  
  
    // Das Hauptprogramm kann geschachtelte Klassen nutzen  
  
    public static void main(String[] args) {  
        // hier steht das eigentliche Hauptprogramm  
    }  
  
}
```

```
public class BeispielGeschachtelteKlasse {  
    public static class Person {  
        public String  vorname;  
        public String  nachname;  
        public int     alter;  
        public double   gehalt;  
        public String  abteilung;  
        public int     personalnummer;  
    }  
    public static void main(String[] args) {  
        Person otto;  
        otto = new Person();  
        otto.vorname = "Otto";  
        otto.nachname = "Meier";  
        otto.alter = 22;  
        otto.gehalt = 2345.0;  
        otto.abteilung = "F&E";  
        otto.personalnummer = 1234;  
        System.out.println(otto.alter);  
        System.out.println(otto.personalnummer);  
        // u.s.w.  
    }  
}
```

- Die Einbettung von Klassen als geschachtelte Klassen in ein Hauptprogramm birgt Nachteile bezüglich der Erstellung größerer Softwareprojekte
 - unübersichtliche Dateistruktur
 - in der Regel größerer Programmtext bei Wiederverwendung geschachtelter Klassen
 - Schwierigkeit der Korrektur

Besserer Weg

- Klassen als **Elementklassen** bzw. **Top-Level-Klassen** definieren
- Diese werden in eigene Datei ausgelagert
- Diese Datei muss den Namen der zu definierenden Klasse tragen
- Außerdem muss das Schlüsselwort **static** (das bei geschachtelten Klassen verwendet wird) gestrichen werden

Vorgehen für Elementklassen

```
// eigene Datei benutzen
```

```
public class <KLASSENNAME> {  
    // hier einzubindende Variablen angeben  
}
```

Der Quelltext wird in einer Datei

```
<KLASSENNAME>.java
```

gespeichert

- Nach der Übersetzung erhält man eine Klassendatei namens

<KLASSENNAME>.class

- Diese Datei kann dann von anderen Programmen verwendet werden, ohne den Quelltext unserer Klasse zur Verfügung stellen zu müssen

Beispiel:

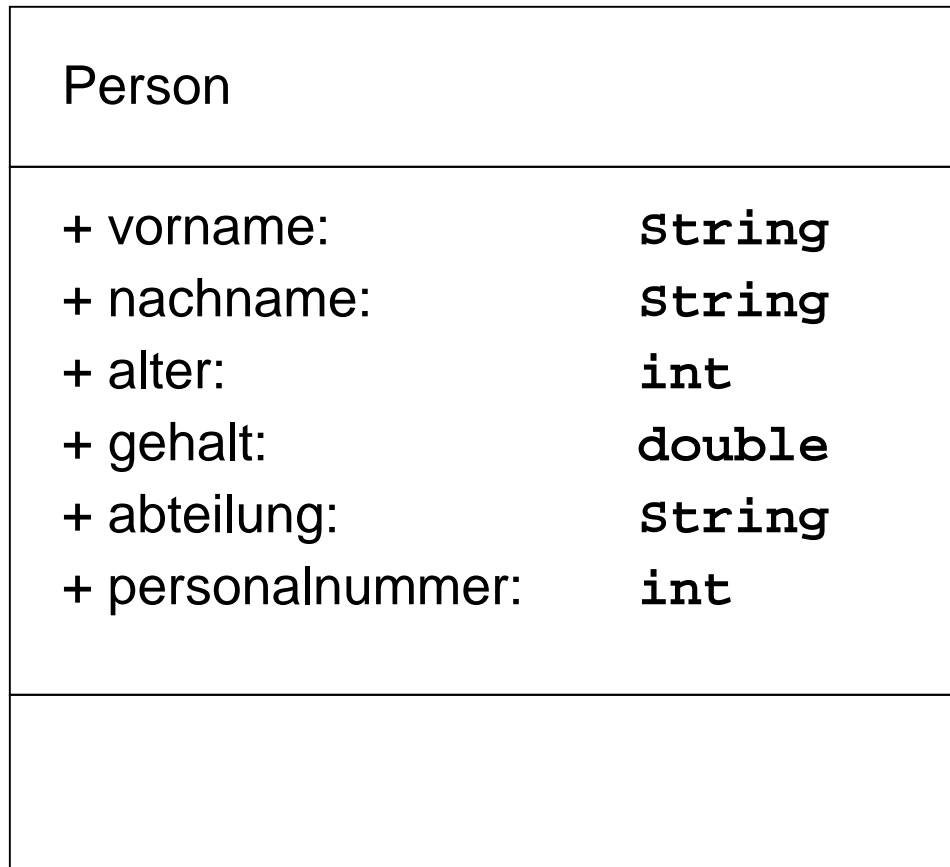
- Quelltext der Klasse Person (**ohne static**) in Datei **Person.java** speichern
- Übersetzen → **Person.class**
- Benutzen nach Bedarf


```
public class Person { // hier kein static!!!  
    public String      vorname1;  
    public String      nachname1;  
    public int         alter1;  
    public double      gehalt1;  
    public String      abteilung1;  
    public int         personalnummer1;  
}
```

```
import console.InputHelper;

public class TestPersonen {
    public static void main ( String[] args ) {
        Person otto;
        otto = new Person();
        otto.vorname1 = "Hans";
        // hier weitere Eingaben
        Person simone;
        simone = new Person();
        simone.vorname1 = "Luise";
        // hier weitere Eingaben
        // eine weitere Person
        Person susi;
        susi = new Person();
        simone.vorname1 = "Susi";
        // hier weitere Eingaben
    }
}
```

Klassendiagramm für Beispiel Person für eine Personaldatei



*Mehr dazu in der
Softwaretechnik-Vorlesung...*

- Der Zugriff von außen auf Klassen bzw. Objekte und deren Komponenten (Variablen und Methoden) wird geregelt durch sogenannte **Zugriffs-Modifizierer** (mehr später)
- Es existieren vier verschiedene Zugriffsberechtigungen
 - **public** (Klassendiagramm-Notation +)
in allen Klassen kann zugegriffen werden
 - **private** (Klassendiagramm-Notation -)
nur innerhalb der Klasse kann zugegriffen werden
 - **protected** (Klassendiagramm-Notation #)
in der Klasse selbst, in *allen* abgeleiteten Klassen und in anderen Klassen desselben Pakets kann zugegriffen werden (mehr später)
 - kein Modifizierer (default „*package-private*“)
in der Klasse selbst und in anderen Klassen desselben Pakets kann zugegriffen werden (Klassendiagramm-Notation ~)

- **Bisher:** Komponenten einer Klasse werden mit jedem neuen Objekt (jeder neuen Instanz) neu erzeugt: **Instanzvariablen**
- **Alternative:** Verwendung des Modifizierers **static** zur Kennzeichnung von Komponenten, die genau einmal erzeugt werden (beim Laden der Klasse) und für jedes Objekt der Klasse denselben Wert haben: **Klassenvariablen**
- Beispiel:

```
public class Mitspieler {  
    public static int gesamtzahl = 0;  
    public int nummer;  
    public String name;  
}
```

- **Klassenvariable**, d.h. als **static** deklarierte Variable gehören zur Klasse und nicht zu deren Instanzen
- Sie werden genau einmal pro Klasse angelegt und erhalten, wenn kein anderer Wert festgelegt wird, den entsprechenden **Default Wert** des ihnen zugeordneten Typs
- Auf Klassenvariable greift man über den Klassennamen (möglich ist auch der Zugriff über eine Objektreferenz) zu:

<KLASSENNAME> . <KLASSENVARIABLE>

- Die übrigen Variablen nennt man **Instanzvariablen**
Instanzvariablen werden für jedes Objekt neu erzeugt

```
public class TestSpieler {
    public static void main (String[] args) {
        Mitspieler ich, du, er;
        ich = new Mitspieler();
        ich.nummer = ++Mitspieler.gesamtzahl;
        ich.name = "Hugo";
        du = new Mitspieler();
        du.nummer = ++Mitspieler.gesamtzahl;
        du.name = "Hilde";
        er = new Mitspieler();
        er.nummer = ++Mitspieler.gesamtzahl;
        er.name = "Otto";
        System.out.println("Du bist Spieler Nr. "
                           + du.nummer + " von insgesamt "
                           + Mitspieler.gesamtzahl);
    }
}
```

Ausgabe:

```
Du bist der Spieler Nr. 2 von insgesamt 3
```

Danach geben die Anweisungen

```
System.out.println(er.gesamtzahl)  
System.out.println(du.gesamtzahl)  
System.out.println(ich.gesamtzahl)
```

allesamt den Wert 3 aus!


```
public class Punkt {    // Punkt in der Ebene
    public double x;      // x-Koordinate des Punktes
    public double y;      // y-Koordinate des Punktes
}

public class Rechteck { // Rechteck in der Ebene
    public double breite; // Breite des Rechtecks
    public double hoehe;  // Hoehe des Rechtecks
    public Punkt luEcke;   // Position der linken unteren
                          // Ecke des Rechtecks
}
```

```

public class TestRechteck {
    public static void main(String[] args){
        Punkt p = new Punkt();
        p.x = 3;
        p.y = 4;

        Rechteck r1 = new Rechteck();
        r1.breite = 7;
        r1.hoehe = 4;
        r1.luEcke = p;

        Rechteck r2 = new Rechteck();
        r2.breite = 1;
        r2.hoehe = 2;
        r2.luEcke = new Punkt();
        r2.luEcke.x = 5;
        r2.luEcke.y = 0;

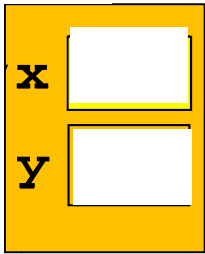
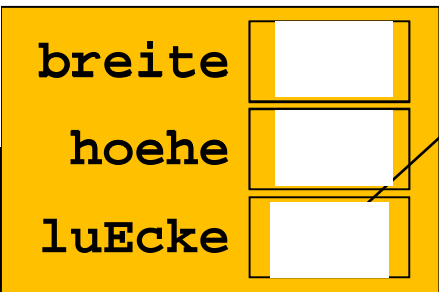
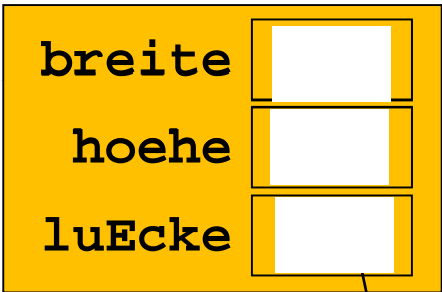
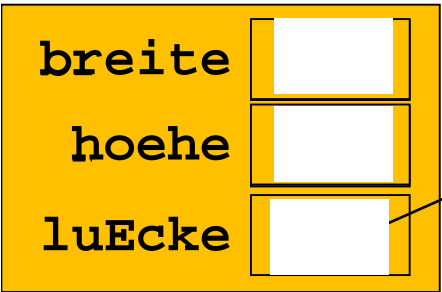
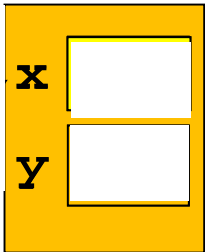
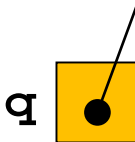
        Punkt q = p;
        q.x = 10;

        System.out.println(p.x);
        System.out.println(r1.luEcke.x);

        Rechteck s = new Rechteck();
        s.breite = r2.breite;
        s.hoehe = r2.hoehe;
        s.luEcke = r2.luEcke;
        r2.luEcke.x = 1;

        System.out.println(s.luEcke.x);
    }
}

```



```
public class TestRechteck {
    public static void main(String[] args){
        Punkt p = new Punkt();
        p.x = 3;
        p.y = 4;

        Rechteck r1 = new Rechteck();
        r1.breite = 7;
        r1.hoehe = 4;
        r1.luEcke = p;

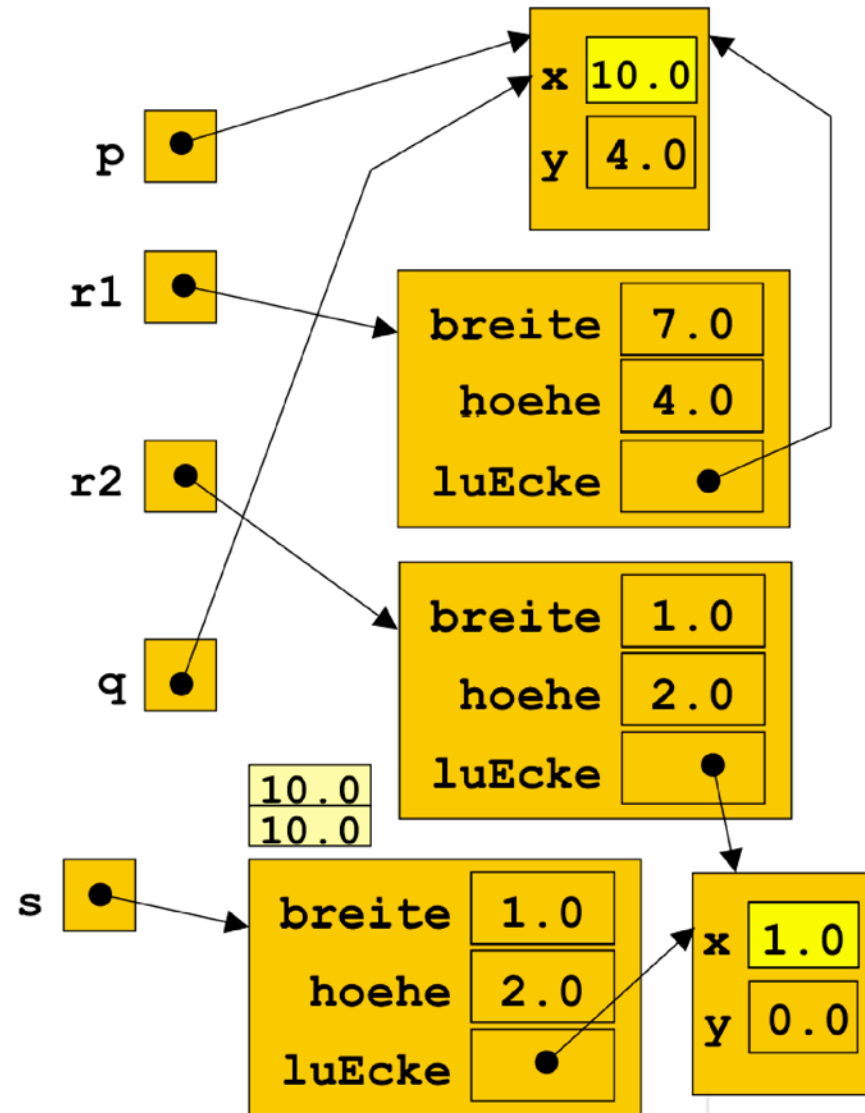
        Rechteck r2 = new Rechteck();
        r2.breite = 1;
        r2.hoehe = 2;
        r2.luEcke = new Punkt();
        r2.luEcke.x = 5;
        r2.luEcke.y = 0;

        Punkt q = p;
        q.x = 10;

        System.out.println(p.x);
        System.out.println(r1.luEcke.x);

        Rechteck s = new Rechteck();
        s.breite = r2.breite;
        s.hoehe = r2.hoehe;
        s.luEcke = r2.luEcke;
        r2.luEcke.x = 1;

        System.out.println(s.luEcke.x);
    }
}
```



- Instanzvariablen werden bei der Anlage eines Objektes mit ihren Standardwerten (Default-Werten) belegt

byte	(byte) 0
short	(short) 0
int	0
long	0L
char	(char) 0
boolean	false
float	0.0F
double	0.0D
Referenztyp	null

Fragen?

