

## **Vorlesung an der Berufsakademie Oldenburg**

### **Unterrichtseinheit 19**

#### **Verbunde**



**Dr. Dietrich Boles**

- Verbunde
  - Motivation
  - Verbundtypen
  - Verbundvariablen
  - Verbunderzeugung
  - Verbundattribute
  - Felder mit Verbunden
  - Arrays und Verbunde als Attribute
  - und Funktionen
  
- Beispiele
  
- „Abstrakter Datentyp“

## Definition:

Datenstruktur/-typ zur Zusammenfassung von mehreren u.U. **Elementen unterschiedlichen Typs** zu einer Einheit. Der Wertebereich eines Verbundes ist das kartesische Produkt der Wertebereiche seiner einzelnen Elemente (auch **Attribute** genannt)

## Synonyme:

Records, Strukturen, Datensätze

## Anmerkungen:

- Verbunde gibt es in Java nicht; man kann sie aber mit Hilfe von Klassen bzw. Objekten nachbilden
- Verbundtypen sind Referenzdatentypen!

### Beispiel: Personaldatei

#### Mitarbeiter:

Name: String name;  
Alter: short alter;  
Geschlecht: boolean istMann;  
Gehalt: float gehalt;

### Java:

Definition des Verbundtyps:

```
class Mitarbeiter {  
    String name;  
    short alter;  
    boolean maennlich;  
    float gehalt = 2000.0F;  
}
```

Definition einer Verbundvariablen:

```
Mitarbeiter dibo;
```

Nutzung der Variablen:

```
dibo.alter = 42;
```

## Verbundtypen / Definition

```
<Verbundtyp> ::= "class" <Bezeichner>  
                "{" { <Variablendefinition> } "}"
```

### Anmerkungen:

- der Bezeichner legt den Namen für den neuen Typ fest
- der neue Typ kann überall dort stehen, wo auch Standarddatentypen stehen dürfen
- die Variablen nennt man auch Elemente oder Attribute

### Beispiel: Datei Personalverwaltung.java

```
class Mitarbeiter {  
    String name;  
    short alter;  
    boolean maennlich;  
    float gehalt = 2000.0F;  
}  
class Personalverwaltung {...}
```

neuer Typ

Attribute

**<Verbundvar-Def> ::= <Verbundtyp> <Bezeichner>  
{ "," <Bezeichner> } ";"**

## Anmerkungen:

- <Verbundtyp> muss gültiger Verbundtyp sein
- Reservierung von Speicherplatz für Adressen (Referenzen auf Verbünde)
- <Bezeichner> ist der Name der Verbundvariablen
- kann Referenzen (Adressen) auf Verbunde speichern

## Beispiele:

**Mitarbeiter dibo;**

**Mitarbeiter hans, hubert;**

**Mitarbeiter heike = null; // expl. Initialisierung**

`<Verbund-Erz> ::= "new" <Verbundtyp> "(" " ")"`

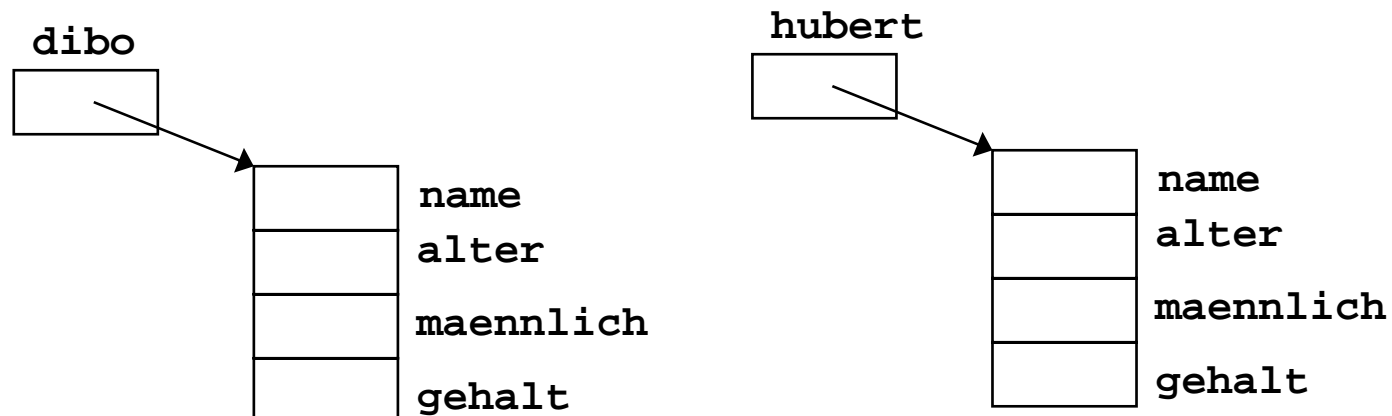
## Anmerkungen:

- `<Verbundtyp>` muss gültiger Verbundtyp sein
- Reservierung von Speicherplatz für Verbundattribute (auf dem Heap)
- liefert Referenz (Adresse) des Speicherbereiches

## Beispiele:

```
Mitarbeiter dibo = new Mitarbeiter();
```

```
Mitarbeiter hubert = new Mitarbeiter();
```



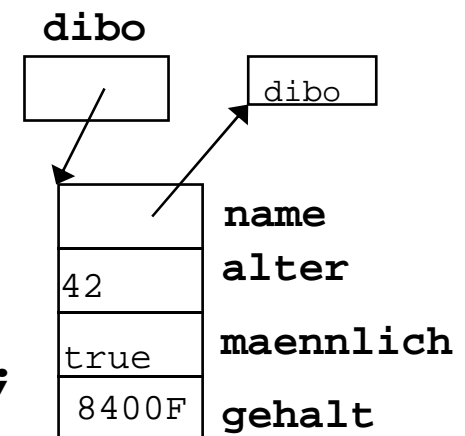
**<Verbund-Attr-Zugr> ::= <verbundvar-bezeichner>  
"."  
<verbund-attr>**

## Anmerkungen:

- <verbundvar-bezeichner> muss gültiger Name einer Verbundvariablen sein
- <verbund-attr> muss Attribut eines Verbundes vom Typ der Verbundvariablen sein
- Verbund muss vorher erzeugt worden sein (-> Laufzeitfehler!)
- können überall dort stehen, wo auch Variablennamen stehen dürfen

## Beispiele:

```
Mitarbeiter dibo = new Mitarbeiter();  
dibo.name = "dibo";  
dibo.alter = 42;  
dibo.maennlich = true;  
dibo.gehalt = 2000.0F * (dibo.alter / 10.0F);
```

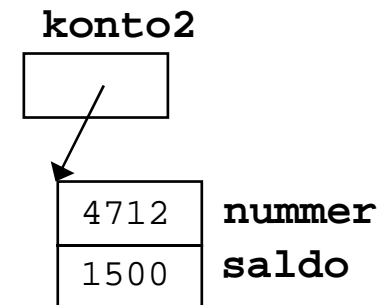
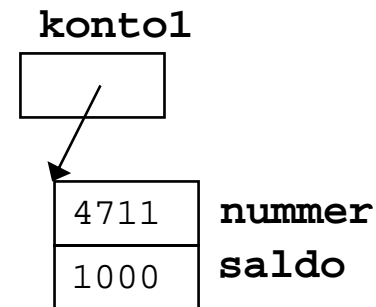




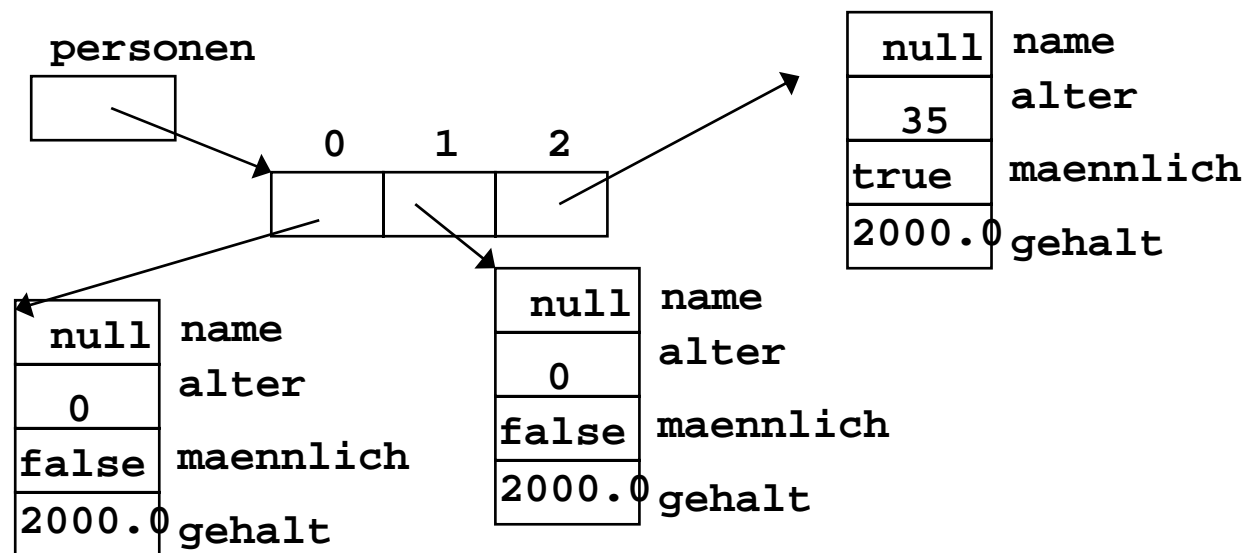
- bei der Definition des Verbundtyps (allgemeingültig)
- durch Default-Werte
- explizit für jedes Attribut

## Beispiele:

```
class Konto {  
    int nummer;  
    int saldo = 1000;  
} ...  
public static void main(String[] a){  
    Konto konto1 = new Konto();  
    konto1.nummer = 4711;  
    Konto konto2 = new Konto();  
    konto2.nummer = 4712;  
    konto2.saldo += 500;  
}
```



```
Mitarbeiter[] personen = new Mitarbeiter[3];  
for (int i=0; i<3; i++) {  
    personen[i] = new Mitarbeiter();  
}  
personen[2].alter = 35;  
personen[2].maennlich = true;
```



# Arrays und Verbunde als Attribute

```
class Produkt {
    String bez;
    float  preis;
}

class Kunde {
    int    nummer;
    String name;
}

class Warenkorb {
    Kunde    kaeufer;
    Produkt[] produkte;
}

public class Shop {

    Warenkorb korb = new Warenkorb();

    korb.kaeufer = new Kunde();
    korb.kaeufer.nummer = 4711;
    korb.kaeufer.name = "dibo";

    korb.produkte = new Produkt[3];

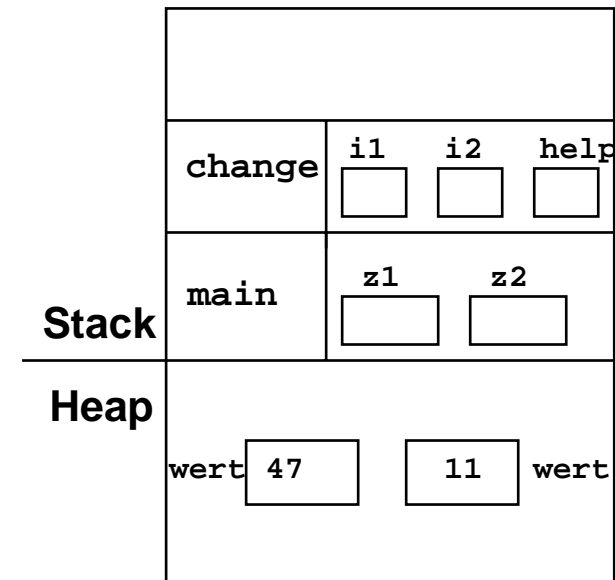
    korb.produkte[0] = new Produkt();
    korb.produkte[0].bez = "Hamster-Buch";
    korb.produkte[0].preis = 21.90F;
    ...
}}
```

# Verbunde und Funktionen (1)

```
class Integer {
    int wert;
}
```

```
public static void change(Integer i1, Integer i2) {
    Integer help = i1;
    i1 = i2;
    i2 = help;
}
```

```
public static void main(String[] a) {
    Integer z1 = new Integer();
    z1.wert = 47;
    Integer z2 = new Integer();
    z2.wert = 11;
    change(z1, z2);
    // z1.wert == 47
    // z2.wert == 11;
}
```

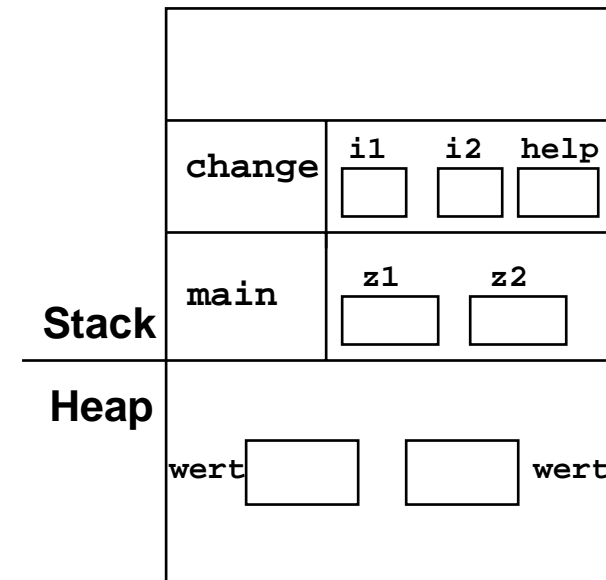


# Verbunde und Funktionen (2)

```
class Integer {  
    int wert;  
}
```

```
public static void change(Integer i1, Integer i2) {  
    int help = i1.wert;  
    i1.wert = i2.wert;  
    i2.wert = help;  
}
```

```
public static void main(String[] a) {  
    Integer z1 = new Integer();  
    z1.wert = 47;  
    Integer z2 = new Integer();  
    z2.wert = 11;  
    change(z1, z2);  
    // z1.wert == 11  
    // z2.wert == 47;  
}
```



# Beispiel 1 (1)

```
class Position { int zeile; int spalte; }

public class Schiebespiel {
    public static void main(String[] args) {
        int[][] matrix = { {2, 14, 5, 7},
                           {3, 4, 15, 13},
                           {6, 1, 12, 11},
                           {3, 8, 9, 0} };

        print(matrix);
        while (!korrekt(matrix)) {
            Position position = posEingabe(matrix);
            verschieben(matrix, position);
            print(matrix);
        }
    }
}
```

2	14	5	7
3	4	15	13
6	1	12	11
3	8	9	



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

<http://www.mazeworks.com/sliders/index.htm>

## Beispiel 1 (2)

---

```
static void print(int[][] matrix) {  
    for (int i=0; i<matrix.length; i++) {  
        for (int j=0; j<matrix[i].length; j++) {  
            if (matrix[i][j] <= 10)  
                IO.print(" ");  
            IO.print(matrix[i][j] + " ");  
        }  
        IO.println();  
    }  
    IO.println();  
}
```

## Beispiel 1 (3)

```
static boolean korrekt(int[][] matrix) {  
    int vgl = 0;  
    for (int i=0; i<matrix.length; i++) {  
        for (int j=0; j<matrix[i].length; j++) {  
            if (vgl != matrix[i][j]) return false;  
            vgl++;  
        }  
    }  
    return true;  
}
```

```
static Position posEingabe(int[][] matrix) {  
    // Achtung: erwartet eine korrekte Eingabe!  
    Position pos = new Position();  
    pos.zeile = IO.readInt("korrekte Zeile eingeben: ");  
    pos.spalte = IO.readInt("korrekte Spalte eingeben: ");  
    return pos;  
}
```



## Beispiel 1 (4)

```
static void verschieben(int[][] m, Position pos) {
    Position frei = freiesFeld(m);
    m[frei.zeile][frei.spalte] = m[pos.zeile][pos.spalte];
    m[pos.zeile][pos.spalte] = 0;
}

static Position freiesFeld(int[][] matrix) {
    for (int i=0; i<matrix.length; i++) {
        for (int j=0; j<matrix[i].length; j++) {
            if (matrix[i][j] == 0) {
                Position pos = new Position();
                pos.zeile = i; pos.spalte = j;
                return pos;
            } } }
    return null; // sollte eigentlich nicht vorkommen!
}

} // end class
```

## Beispiel 2 (1)

```
class Bruch {  
    int zaehler = 1;  
    int nenner = 1;  
}
```

Abstrakter Datentyp (ADT) =

Datenstruktur

+

Funktionen auf der Datenstruktur

```
class BruchRechnung {
```

```
    public static void init(Bruch bruch, int z, int n) {  
        bruch.zaehler = z;  
        bruch.nenner = n;  
        kuerzen(bruch);  
    }
```

## Beispiel 2 (2)

---

```
static void multTo(Bruch b1, Bruch b2) {  
    b1.zaehler *= b2.zaehler;  
    b1.nenner *= b2.nenner;  
    kuerzen(b1);  
}
```

```
static void addTo(Bruch b1, Bruch b2) {  
    b1.zaehler =  
        b2.nenner * b1.zaehler +  
        b1.nenner * b2.zaehler;  
    b1.nenner = b1.nenner * b2.nenner;  
    kuerzen(b1);  
}
```

## Beispiel 2 (3)

---

```
static String toString(Bruch b) {  
    return b.zaehler + "/" + b.nenner;  
}
```

```
static void kuerzen(Bruch b) {  
    int ggt = ggT(b.zaehler, b.nenner);  
    b.zaehler /= ggt;  
    b.nenner /= ggt;  
}
```

```
static int ggT(int z1, int z2) {  
    if (z2 == 0) {  
        return z1;  
    } else {  
        return ggT(z2, z1 % z2);  
    }  
}
```

## Beispiel 2 (4)

```
public static void main(String[] args) {
    int zaehler = IO.readInt("Bruch 1 (Zaehler): ");
    int nenner = IO.readInt("Bruch 1 (Nenner): ");
    Bruch b1 = new Bruch();
    init(b1, zaehler, nenner);
    IO.println("Bruch 1: " + toString(b1));

    zaehler = IO.readInt("Bruch 2 (Zaehler): ");
    nenner = IO.readInt("Bruch 2 (Nenner): ");
    Bruch b2 = new Bruch();
    init(b2, zaehler, nenner);
    IO.println("Bruch 2: " + toString(b2));

    multTo(b1, b2);
    IO.println("Bruch 1: " + toString(b1));
    IO.println("Bruch 2: " + toString(b2));

    addTo(b1, b2);
    IO.println("Bruch 1: " + toString(b1));
    IO.println("Bruch 2: " + toString(b2));
} }
```

Das so genannte Normalgewicht berechnet sich nach der Formel "Körpergröße (in cm) minus 100". Das Idealgewicht beträgt bei Männern 90% und bei Frauen 85% des Normalgewichts.

Schreiben Sie ein Java-Programm, welches nach Eingabe von Größe, Gewicht und Geschlecht ausgibt, ob ein Mensch zu dick oder zu dünn ist, oder ob er/sie zwischen Ideal- und Normalgewicht liegt.

Verwenden Sie in Ihrem Programm den folgenden Verbund:

```
class Mensch {  
    float gewicht;  
    char geschlecht;  
    int groesse;  
}
```

## Aufgabe 2 (1)

- Das „Game-of-Life“ wird auf einem schachbrettartigen Feld gespielt, das eine „Bevölkerung“ von „toten“ und „lebenden“ Zellen darstellt. Jede Zelle kann „überleben“, „sterben“ oder „geboren“ werden. Die schrittweise Entwicklung von einem Stellungsbild zum nächsten erfolgt gemäß einiger Regeln, die berücksichtigen, wie viele lebende Nachbarzellen eine Zelle hat. Eine Zelle  $x$ , die nicht am Spielfeldrand liegt, hat 8 Nachbarzellen, Zellen am Spielfeldrand entsprechend weniger.
- Die Regeln, nach denen sich die Population von einer Stellung zur nächsten entwickelt, sind:
  - 1. Für eine Zelle  $x$ , die gerade tot ist, gilt: Wenn  $x$  genau 3 lebende Nachbarzellen hat, wird  $x$  neu geboren; sonst bleibt  $x$  tot.
  - 2. Für eine Zelle  $x$ , die gerade lebendig ist, gilt: Wenn  $x$  weniger als 2 lebende Nachbarn hat, stirbt  $x$  an Vereinsamung; wenn  $x$  2 oder 3 lebende Nachbarzellen hat, bleibt  $x$  in der nächsten Stellung lebendig. In allen anderen Fällen stirbt  $x$  an Überbevölkerung.

## Aufgabe 2 (2)

- Alle Veränderungen gemäß dieser Regeln geschehen gleichzeitig. Die Simulation beginnt mit einer bestimmten eingelesenen Verteilung von lebenden und toten Zellen.
- Beispiel:

	*	*	
	*		
*			

Population 1

	*	*	
*	*	*	

Population 2

*		*	
*		*	
	*		

Population 3

- Schreiben Sie ein Programm, das zunächst die Größe des Feldes, eine Startpopulation und die Anzahl der Simulationszyklen einliest und anschließend die Simulation startet. Nach jedem Zyklus soll der aktuelle Zustand auf dem Bildschirm ausgegeben werden.
- siehe auch <http://www.ibiblio.org/lifepatterns/>