

# Assoziation und Aggregation

---

Prof. Dr. Christian Böhm

in Zusammenarbeit mit  
Michael Eckert und Gefei Zhang

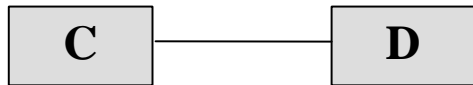
<http://www.dbs.ifi.lmu.de/Lehre/NFInfoWS>

# Ziele

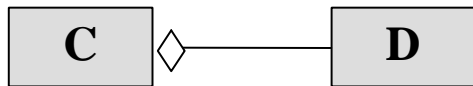
- Verstehen der Begriffe Assoziation und Aggregation
- Implementierung von Assoziationen in Java schreiben lernen

# Assoziation und Aggregation

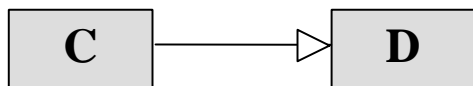
Assoziation ist eine Beziehung zwischen zwei oder mehr Klassen



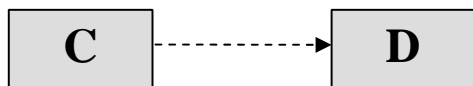
**Assoziation:** Die Klassen C und D stehen in Beziehung



**Aggregation:** Jedes Objekt von C enthält Objekte von D



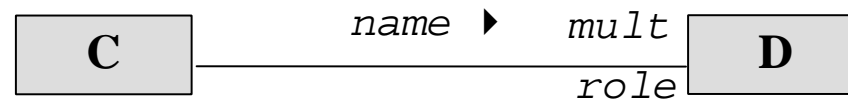
**Vererbung:** Die Klasse C ist Erbe der Klasse D



**Abhängigkeit:** Die Klasse C benützt Elemente der Klasse D (i.a. Methoden)

## Assoziation

Die Assoziation ist die allgemeinste Art einer Beziehung



Das Diagramm drückt aus, daß jedes Objekt  $o$  von C mit „*mult*“-vielen Objekten von D im Beziehung steht, die die Rolle „*role*“ für  $o$  spielen.

Dabei ist *role* ein Name und *mult* entweder eine natürliche Zahl, ein Stern „\*“ für beliebig viele Objekte oder ein Intervall der Form  $a..b$  mit  $a \in \mathbf{N}$ ,  $b \in \mathbf{N} \cup \{*\}$

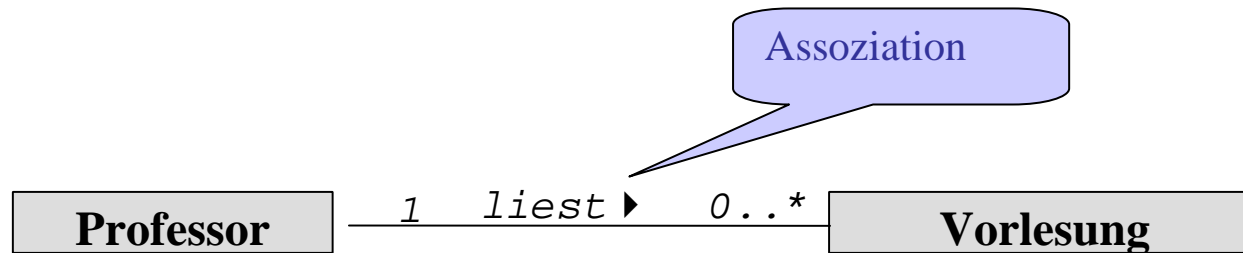
Der Name *name* gibt den Namen der Assoziation an.

Das ausgefüllte Dreieck  $\blacktriangleright$  bezeichnet die Leserichtung.

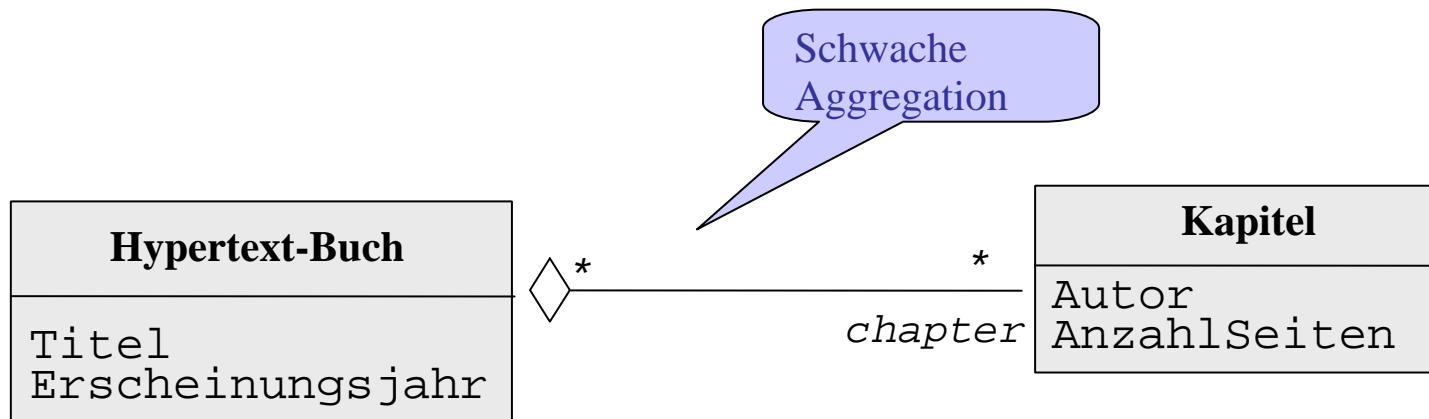
# Assoziation und Aggregation

## Beispiele:

1.



2.

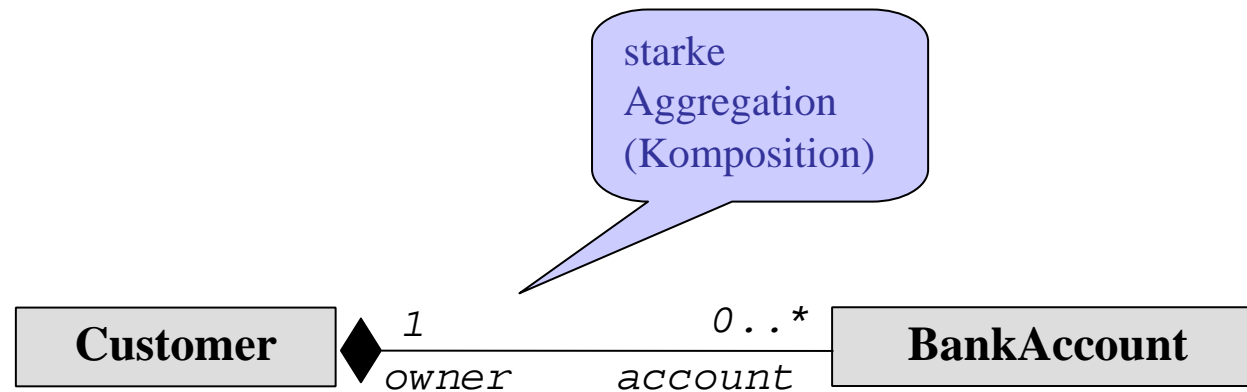


Jedes Hypertext-Buch ist gekennzeichnet durch seine Kapitel, d.h. Kapitel spielt die Rolle "*chapter*" für Hypertext-Buch.

# Assoziation und Aggregation

## Beispiele:

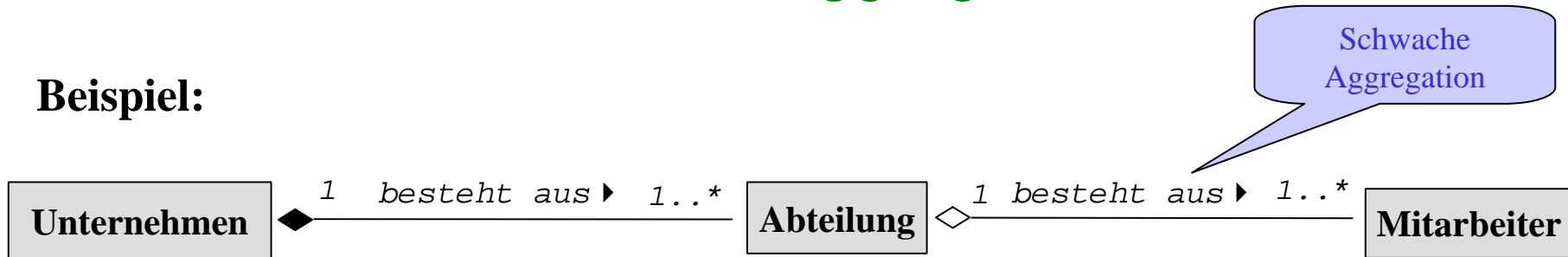
3.



- Jeder Bankkunde besitzt 0 oder mehrere Konten, die die Rolle "*account*" spielen.
- Jedes Bankkonto hat genau einen Besitzer ("*owner*").
- Die Lebensdauer eines Bankkontos ist durch die des Kunden beschränkt.

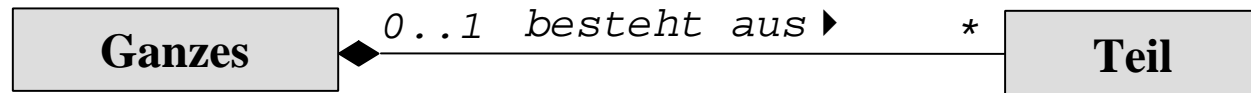
## Schwache Aggregation

### Beispiel:



- Schwache Aggregation ist eine „Teile-Ganzes“ Relation, die Lebensdauer der Teile hängt aber **nicht** von der Lebensdauer des Ganzen ab.
- Es gilt aber, dass die Objekte der schwachen Aggregation einen gerichteten azyklischen Graphen bilden. (Wenn B Teil von A, dann ist A nicht Teil von B).
- Ein Teil darf jedoch auch einem anderen Ganzen zugeordnet werden.

## Starke Aggregation

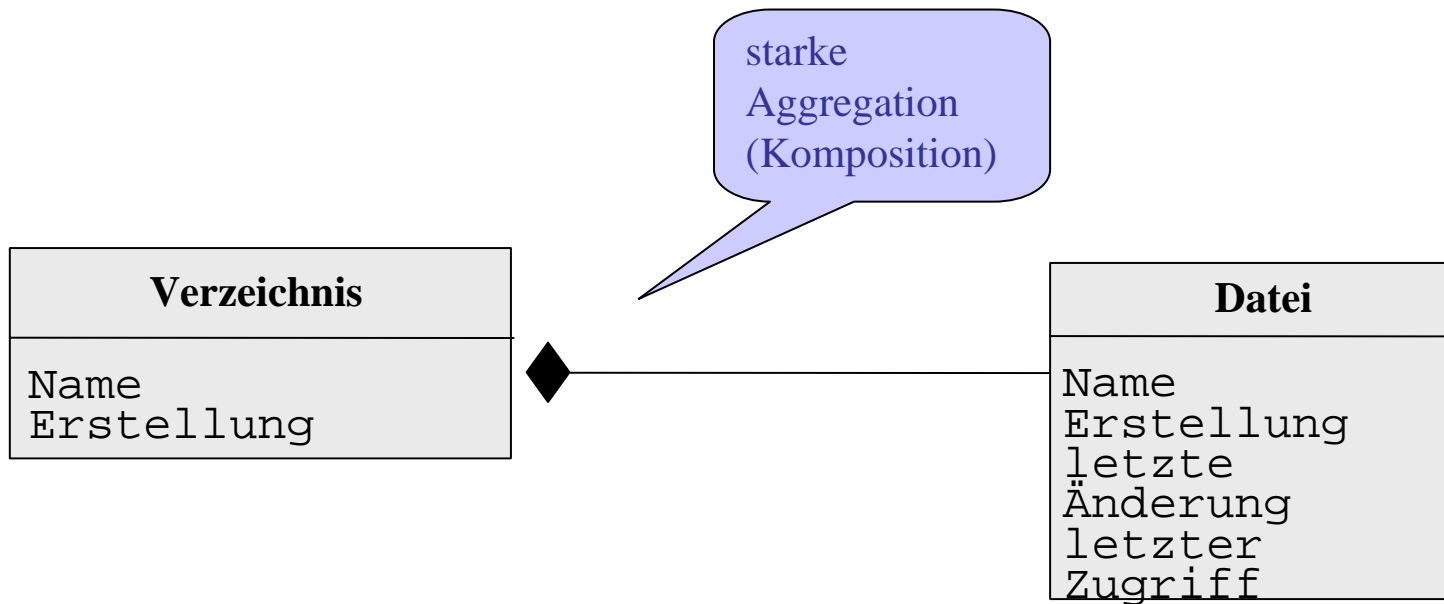


- Die Teile existieren nur innerhalb des Aggregats “Ganzes”; d.h. die Lebensdauer eines “Teil”-Objekts wird durch die Lebensdauer des zugehörigen Aggregats beschränkt.
- Objekte der Aggregation bilden einen Baum.  
(Wenn A ein Objekt B enthält, dann enthält B nicht A.)
- Jedes Objekt der Teilklasse kann - zu einem Zeitpunkt - nur Komponente eines einzigen Objekts der Aggregatklasse sein, d.h., die bei der Aggregatklasse angetragene Kardinalität darf nicht größer als eins sein (*unshared aggregation, strong ownership*).
- Die dynamische Semantik des Ganzen gilt auch für seine Teile (*propagation semantics*). Wird beispielweise das Ganze kopiert, so werden auch seine Teile kopiert.

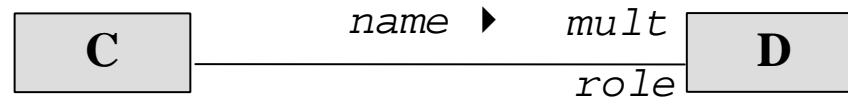


# Starke Aggregation

## Beispiel:

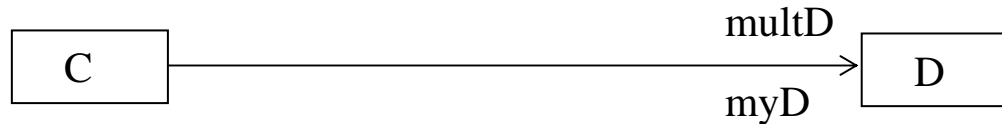


## Java Implementierung von Assoziationen



- In Java repräsentiert man Assoziationen durch Attribute.
- Jede Rolle *role* vom Typ *D* wird als Attribut von *C* implementiert.
- Ist die Multiplizität 0, 1 oder 0..1 erhält man ein Attribut *D role* .
- Ist die Multiplizität >1 oder \* verwendet man eine Reihung oder eine ArrayList von D-Objekten.

## Java Implementierung von Assoziationen



- **multB = 0..1** induziert das Schema:

```
class C
{
    D myD; ...
}
```

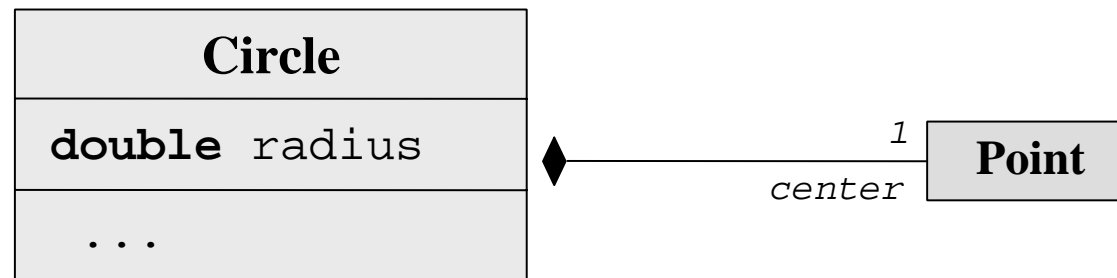
- **multB = \*** induziert das Schema:

```
class C
{
    ArrayList<D> myD; ...
}
```

## Implementierung von starker Aggregation

- Erzeugung der Komponenten durch Konstruktor des Aggregats

### Beispiel



```
class Circle
{
    private double radius;
    private Point center;

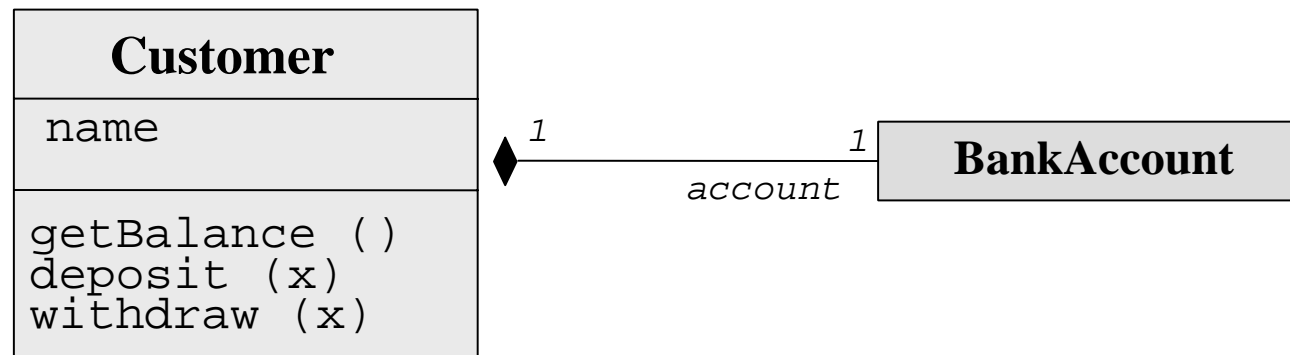
    public Circle(double rad, double x, double y)
    {
        radius = rad;
        center = new Point(x,y);
    }
    ...
}
```

Das Point Objekt wird durch den Konstruktor von Circle erzeugt und sollte keine Beziehungen zu "äußeren" Objekten besitzen.

# Implementierung von starker Aggregation

**Beispiel Assoziation mit Multiplizität 1:**

**Kunde, der genau ein Bankkonto besitzen muss.**



# Implementierung von starker Aggregation

```
class Customer
{
    private String name;
    private BankAccount account;

    public Customer(String cname, double initialBalance)
    {
        name = cname;
        account = new BankAccount(initialBalance);
    }

    public String getName()
    {
        return name;
    }

    public double getBalance()
    {
        return account.getBalance();
    }

    public void transferTo(BankAccount other, double amount)
    {
        account.transferTo(other, amount);
    }

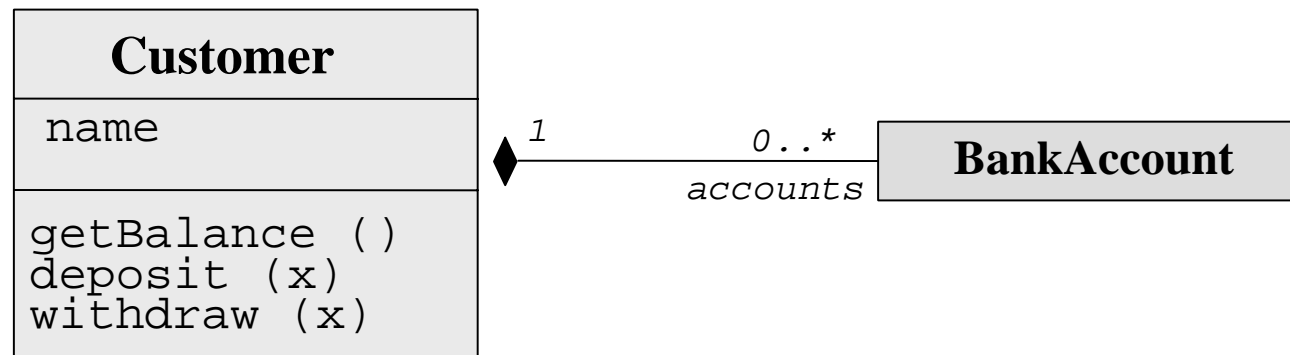
    public void withdraw(double x)
    {
        account.withdraw(x);
    }

    public void deposit(double x)
    {
        account.deposit(x);
    }
}
```

# Implementierung von Assoziationen mit Multiplizität \*

**Beispiel:**

**Kunden mit 0 oder mehreren Bankkonten**



## Die Klasse ArrayList

```
class ArrayList<T>
{
    public ArrayList()
        //Constructs an empty list with an initial capacity of ten.

    public ArrayList(int initialCapacity)
        //Constructs an empty list with the specified initial capacity.

    public T get(int index)
        //Returns the element at the specified position in this list.

    public T set(int index, T element)
        //Replaces the element at the specified position
        //in this list with the specified element.

    public boolean add(T element)
        //Appends the specified element to the end of this list.

    public int size()
        //Returns the number of elements in this list.
}
```



# Implementierung von Assoziationen mit Multiplizität \*

```
import java.util.ArrayList;

class Customer
{
    private String name;
    private ArrayList<BankAccount> accounts;

    public Customer(String cname)
    {
        name = cname;
        accounts = new ArrayList<BankAccount>();
    }

    public void createAccount(double initialBalance)
    {
        BankAccount a = new BankAccount(initialBalance);
        accounts.add(a);
    }

    public String getName()
    {
        return name;
    }
}
```



fügt neues Element an das Ende der ArrayList an.

## Implementierung von starker Aggregation

. . .

```
public double getBalance(int i)
{
    BankAccount a = accounts.get(i);
    return a.getBalance();
}

public void transferTo(int i, BankAccount other, double amount)
{
    BankAccount a = accounts.get(i);
    a.transferTo(other, amount);
}

public void withdraw(int i, double x)
{
    BankAccount a = accounts.get(i);
    a.withdraw(x);
}

public void deposit(int i, double x)
{
    BankAccount a = accounts.get(i);
    a.deposit(x);
}
}
```

## Zusammenfassung

- Eine Assoziation ist eine Relation (Beziehung) zwischen Klassen.
- Eine Aggregation ist eine Teile-Ganzes-Beziehung und damit eine spezielle Assoziation. Bei einer starken Aggregation sind die Teile existenzabhängig vom Ganzen. Insbesondere kann jedes Teil nur zu einem Aggregat gehören.
- Eine Rolle myB (vom Typ B) mit Multiplizität 1 oder 0..1 wird durch ein Attribut myB vom Typ B implementiert, bei Multiplizität \* bekommt es den Typ ArrayList.
- Bei der starken Aggregation werden die abhängigen Objekte im Konstruktor des Aggregats erzeugt, um die Lebensdauerabhängigkeit zu garantieren.