gds2 Gottlieb-Daimler-Schule 2
Technisches Schulzentrum Sindelfingen
mit Abteilung Akademie für Datenverarbeitung

# Inhaltsverzeichnis

# 1 Assoziation

## 1.1 Begriffsbestimmung

Eine Assoziation modelliert Verbindungen zwischen Objekten einer oder mehrerer Klassen.
In der UML wird eine Assoziation durch eine Linie zwischen den beteiligten Klassen dargestellt.
Obwohl die Assoziation eigentlich zwischen Objekten besteht, spricht man dennoch von einer Assoziation zwischen Klassen.

Jede Assoziation hat einen ***Namen***. Dieser ist ein Verb und wird i.d.R. unter die Linie geschrieben und mit einem ***Pfeil als Leserichtung*** versehen. Der Name kann fehlen, wenn die Bedeutung der Assoziation offensichtlich ist. Er wird kursiv geschrieben.

Die Verbindungslinie der Assoziation wird ergänzt durch die Angabe des Mengenverhältnisses der beteiligten Objekte, der sog. ***Kardinalität*** oder ***Multiplizität***.

Beispiel:


Ein Kunde besitzt 0 bis viele Konten, ein Konto gehört genau einem Kunden.
Assoziationen sind, wenn nicht anders gekennzeichnet ***biderektional (ungerichtet)***, d.h. ein Kunde kennt seine Konten und ein Konto kennt seinen Inhaber.

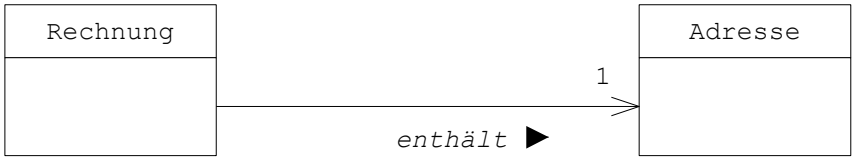
Folgende Kardinalitäten sind möglich:

| | |
|---|---|
| 1 | genau 1 |
| 0..1 | 0 bis 1 |
| * | 0 bis viele |
| 3..* | 3 bis viele |
| 2,4,6 | 2, 4 oder 6 |
| 1..5,7..* | nicht 6 |

Dementsprechend gibt es Kann- und Muss-Assoziationen: eine Kann-Assoziation hat als Untergrenze 0. Kardinalitäten werden über die Linie geschrieben.
Zur besseren Beschreibung der Bedeutung eines Objekts in der Assoziation kann eine **Rolle** angegeben werden. Die Angabe ist Pflicht, wenn mehrere Assoziationen zwischen zwei Klassen bestehen oder bei reflexiven Assoziationen (s.u.).

## 1.2 Gerichtete Assoziation

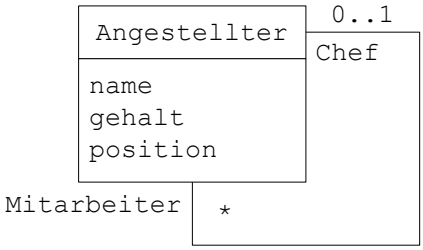
Gerichtete Assoziationen sind Beziehungen, die nur in eine Richtung navigierbar sind.
Dargestellt wird die Navigation durch eine offene Pfeilspitze, die die zugelassene Navigationsrichtung angibt.




Eine Rechnung enthält genau eine Adresse. Die umgekehrte Richtung ist nicht relevant.

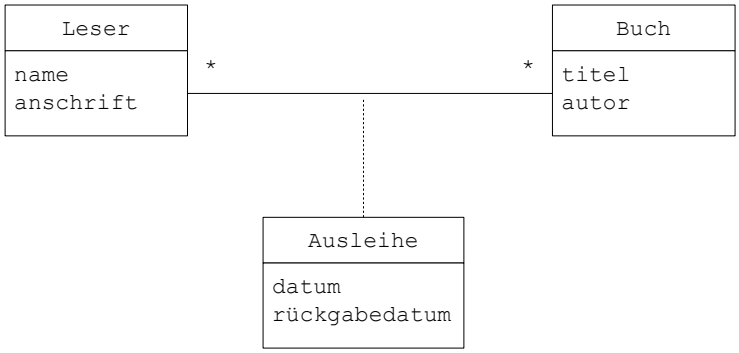
### 1.3 Reflexive Assoziation

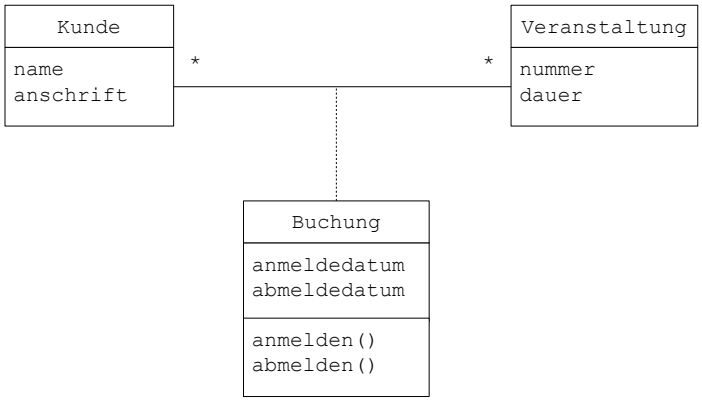
Eine Assoziation zwischen Objekten der selben Klasse heißt *reflexiv*. Hier müssen zum besseren Verständnis Rollen angegeben werden.



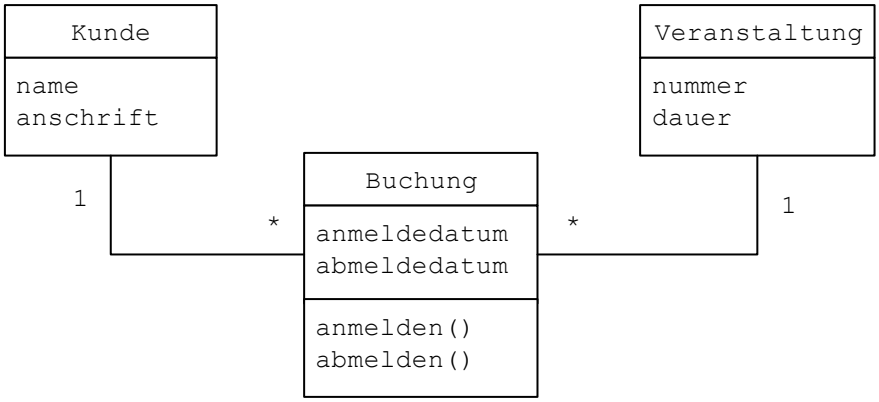

### 1.4 Assoziative Klasse

Eine Assoziation kann zusätzlich die Eigenschaften einer Klasse besitzen, d.h. sie hat Attribute, Operationen sowie Assoziationen zu anderen Klassen. Sie wird dargestellt wie eine Klasse, mit gestrichelter Linie zur Assoziation. Beispiele:



Realisierung der assoziativen Klasse durch Auflösung in zwei Assoziationen:
Hier:
Ein Kunde bucht mehrere Veranstaltungen,
eine Veranstaltung kann von mehreren Kunden gebucht werden.
Jede Buchung hat dann z.B. ein An- und ein Abmeldedatum




## 2 Aggregation

Bei einer Aggregation besteht zwischen den beteiligten Klassen eine "is-part-of"-Beziehung (= "Ist-Teil-Von"). Zu einem Zeitpunkt kann ein Teilobjekt mehreren Aggregatobjekten zugeordnet sein ("shared aggregation"):


Eine Aggregation wird durch eine offene Raute dargestellt, die auf der Seite des "Ganzen" (=Aggregatklasse) angebracht wird.

## 3 Komposition

Eine Komposition ist eine starke Form der Aggregation. Zu einem Zeitpunkt kann ein Teilobjekt nur einem einzigen Aggregatobjekt zugeordnet sein ("unshared aggregation"):


Eine Komposition wird durch eine geschlossene Raute dargestellt, die auf der Seite des "Ganzen" (=Aggregatklasse) angebracht wird.

# 4 Realisierung in Java

## 4.1 Beispiel für eine einfache Assoziation

```
public class Konto
{
  private int iKontoNr;
  private double dKontostand;
  private Kunde inhaber;

  public void setInhaber(Kunde derKunde)
  {
    this.inhaber = derKunde;
  }
}

public class Kunde
{
  private String sName;
  private ArrayList<Konto> konten = new ArrayList<Konto>();
  public void addKonto(Konto einKonto)
  {
    konten.add(einKonto);  // Dubletten möglich, durch
                           // Programmierung verhindern
  }
}
```

## 4.2 Beispiel für eine gerichtete Assoziation

Die Adresse „weiß“ nicht, auf welchen Rechnungen sie steht.

```
public class Rechnung
{
  private GregorianCalendar datum;
  private double dBetrag;
  private Adresse adresse;

  public void setAdresse(Adresse eineAdresse)
  {
    this.adresse = eineAdresse;
  }
}

public class Adresse
{
  private String sName;
  private String sAnschrift;
}
```

### 4.3 Reflexive Assoziation

```
import java.util.ArrayList;
public class Angestellter
{
  private String sName;
  private Angestellter chef;
  private ArrayList<Angestellter> mitarbeiter =
                                        new ArrayList<Angestellter>();
  public Angestellter(String derName)
  {
    this.sName = derName;
  }

  public static void macheZumChefVon(Angestellter derChef,
                                     Angestellter derMitarbeiter)
  {
    derMitarbeiter.chef = derChef;
    derChef.mitarbeiter.add(derMitarbeiter);
  }
}
```

### 4.4 Assoziative Klasse

```
import java.util.*;
import java.util.GregorianCalendar;

public class AssoziativeKlasseDemo
{
 public static void main(String [] x)
 {
   Veranstaltung v1 = new Veranstaltung();
   v1.setNummer(1);
   Kunde k1 = new Kunde();
   k1.setName("Maier");
   Buchung.anmelden(k1,v1);
   Buchung.anmelden(k1,v1);// schon angemeldet

 }
}
```

```
public class Veranstaltung
{
  private int nummer;
  private float dauer;
  private ArrayList<Buchung> buchungen = new ArrayList<Buchung>();

  public int getNummer()
  {
    return (nummer);
  }
  public void setNummer(int iNr)
  {
    nummer = iNr;
  }

  public ArrayList<Buchung> getBuchungen()
  {
    return (buchungen);
  }
}
public class Kunde
{
  private String name;
  private String adresse;
  private ArrayList<Buchung> buchungen = new ArrayList<Buchung>();

  boolean istSchonAngemeldet(Veranstaltung eineVeranstaltung)
  {
    boolean antwort = false;
    Iterator<Buchung> i=buchungen.iterator();
    while(i.hasNext() && !antwort)
    {
      antwort = (i.next().getVeranstaltung() == eineVeranstaltung);
    }
    return antwort;
  }
  public void setName(String sName)
  {
    name = sName;
  }
  public String getName()
  {
    return (name);
  }
  public ArrayList getBuchungen()
  {
    return (buchungen);
  }
}
```

```
public class Buchung
{
  private GregorianCalendar anmeldedatum, abmeldedatum;
  private Kunde kunde;
  private Veranstaltung veranstaltung;

  public static void anmelden(Kunde einKunde,
                              Veranstaltung eineVeranstaltung)
  {
    if(!einKunde.istSchonAngemeldet(eineVeranstaltung))
    {
      // Kunde darf nur einmal an einer Veranstaltung
      // angemeldet sein!
      Buchung neueBuchung = new Buchung();
      neueBuchung.anmeldedatum = new GregorianCalendar();
      neueBuchung.kunde = einKunde;
      neueBuchung.veranstaltung = eineVeranstaltung;
      einKunde.getBuchungen().add(neueBuchung);
      eineVeranstaltung.getBuchungen().add(neueBuchung);
      System.out.println("angemneldet");
    }
    else
      System.out.println("nicht angemeldet");
  }

  public Veranstaltung getVeranstaltung()
  {
    return (veranstaltung);
  }
}
```