

# ***Objektorientierte Modellierung***

*Prof. Dr. Roland Dietrich*

## **3. Statische Modellierung mit UML**

Objekte und Klassen

Assoziationen

Generalisierung

Pakete

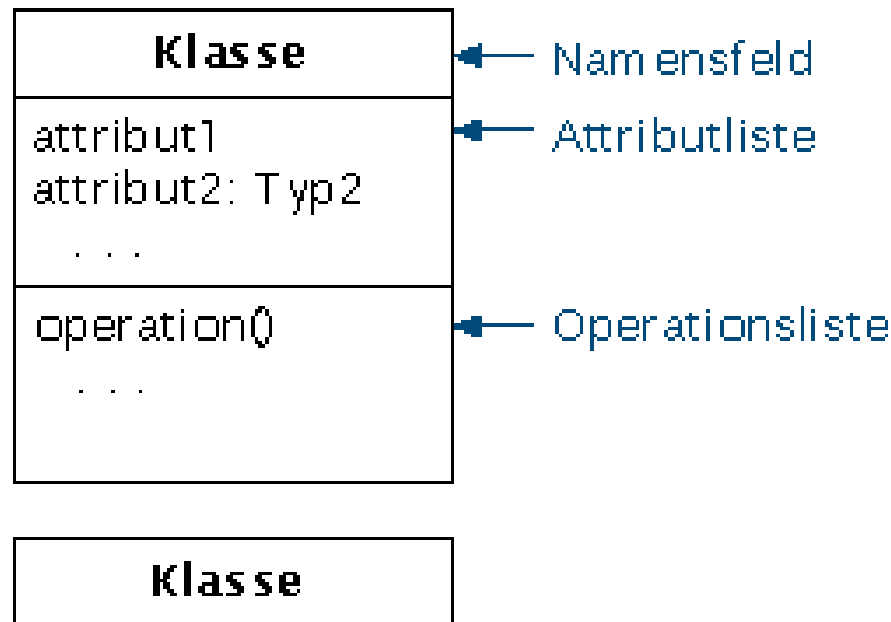
- Ein **Objekt**
  - ist ein „Gegenstand des Interesses“
    - Dinge (Fahrrad, Auto, Büro, Buch,...)
    - Personen (Kunden, Mitarbeiter,...)
    - Begriffe (Konto, Graph, Krankheit,...)
- Merkmale von Objekten
  - besitzt einen bestimmten **Zustand**
    - Wie ist die innere Struktur von Objekten, welche Daten/Informationen stecken in Objekten?  
→ **Attribute**
  - reagiert mit einem definierten **Verhalten** auf seine Umgebung
    - Was können Objekte, was kann man mit Objekten tun?  
→ **Operationen**
  - besitzt eine Identität, die es von allen anderen Objekten unterscheidet
  - kann ein oder mehrere andere Objekte kennen (**Objektbeziehungen**)
  - Objekte arbeiten zusammen, um bestimmte Probleme zu lösen

- Eine **Klasse**
  - definiert für eine Kollektion von Objekten deren
    - Struktur (Attribute)
    - Verhalten (Operationen) und
    - Beziehungen (*relationships*)
  - jedes Objekt gehört zu genau einer Klasse (ist **Instanz** einer Klasse)
  - Beispiele
    - Zur Klasse *Person* gehören die Objekte *Hans Müller, geb. am 1.1.1970* und *Carolin Maier, geb. am 2.2.1980*
      - Alle Personen haben die gleiche innere Struktur: sie haben einen Namen und ein Geburtsdatum (→ **Attribute**)
      - Jedes einzelne Objekt hat individuelle **Attributwerte** für diese Attribute
      - Alle Personen haben ähnliche Fähigkeiten (**Operationen**), sie können z.B. Autos kaufen
      - Alle Personen haben ähnliche Beziehungen, sie besitzen z.B. Autos
    - Zur Klasse *Auto* gehört das Objekt *grüner Opel mit Kennzeichen AA XY 99* und der *rote Ford mit Kennzeichen HDH Z 2010*

- **Attribute**
  - beschreiben die Daten, die von den Objekten einer Klasse angenommen werden können (→ **Zustand**)
  - Jedes Attribut ist von einem bestimmten **Typ**
  - Alle Objekte einer Klasse besitzen dieselben Attribute, jedoch unterschiedliche Attributwerte
- **Operationen**
  - beschreiben Tätigkeiten (Aktivität, Verhalten), die von Objekten einer Klasse ausgeführt werden können
  - Alle Objekte einer Klasse verwenden dieselben Operationen
  - Operationen haben eine **Signatur** (Parameterliste, Resultattyp)
- **Beziehungen**
  - **Assoziationen**
  - **Vererbung** (Verallgemeinerung/Spezialisierung)  
*später mehr!*

# Klassen und Objekte

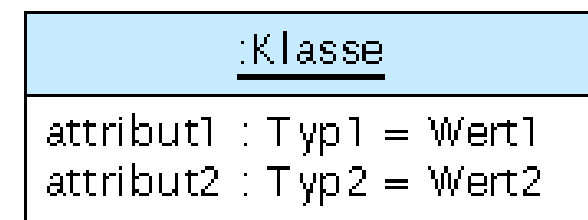
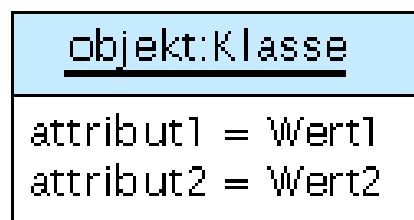
- Notation von Klassen in UML



Quelle: [Balzert 05], Abb. 2.2-2

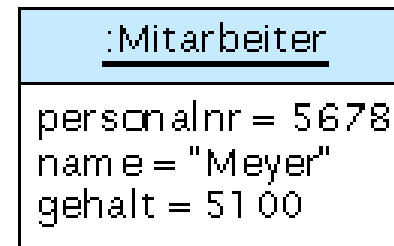
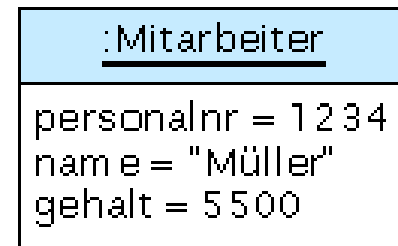


- Notation von Objekten in UML

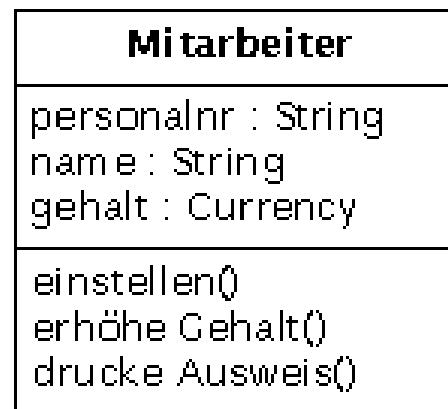


Quelle: [Balzert 05], Abb. 2.1-1

- Beispiel



Objekte (Instanzen)  
der Klasse



Klasse

Quelle: [Balzert 05], Abb. 2.2-1

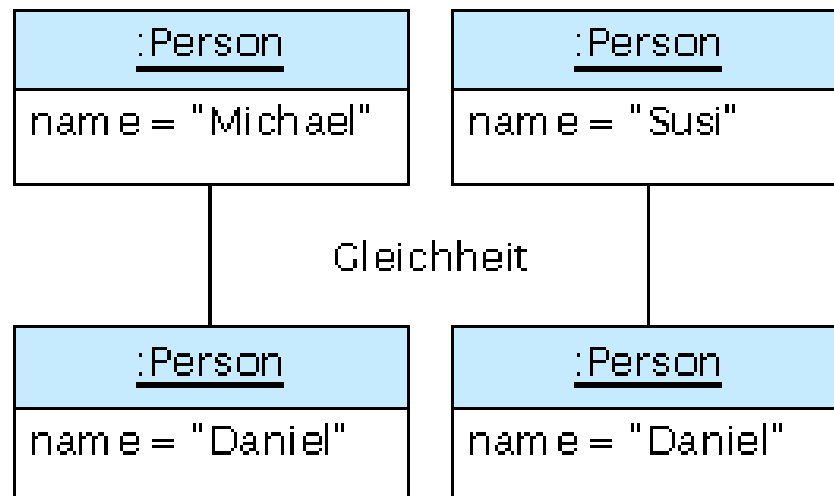
- UML-Namenskonventionen:
  - Attribut- und Operationsnamen klein schreiben!
  - Klassen- und Typnamen groß schreiben!

- **Objektdiagramm** (*object diagram*)
  - Momentaufnahme bzw. „Schnappschuss“ des Systems
  - Beschreibt Objekte, Attributwerte und Objekt-Beziehungen zu einem bestimmten Zeitpunkt
- **Klassendiagramm** (*class diagram*)
  - Enthält Klassen und Beziehungen zwischen Klassen
  - Beschreibung des statischen Modells des Systems
  - Bei großen Systemen mehrere Klassendiagramme erstellen
- **Objektorientierte Softwareentwicklung**
  - Objekte der "realen Welt" werden abgebildet in Software-Objekte
- **Objektorientierte Modellierung**
  - Darstellung der relevanten Objekte der realen Welt und ihrer Merkmale

- **Objekt-Identität**

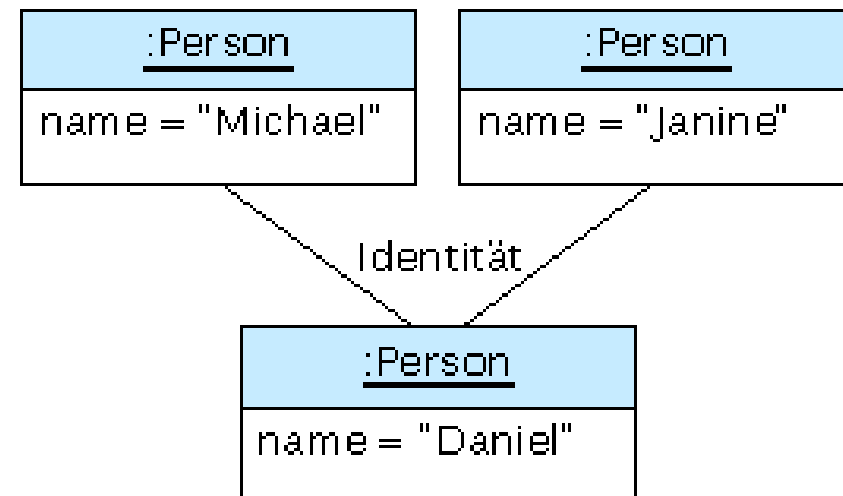
- Eigenschaft, die ein Objekt von allen anderen Objekten unterscheidet
- Die Identität eines Objekts kann sich nicht ändern
- Keine zwei Objekte besitzen dieselbe Identität

- **Gleichheit vs. Identität**



Diese beiden Objekte sind zwar gleich, aber nicht identisch.  
– Jedes Person-Objekt hat seine eigene Identität

Quelle: [Balzert 05], Abb. 2.1-5



Hier sind Michael und Janine mit demselben Person-Objekt verknüpft

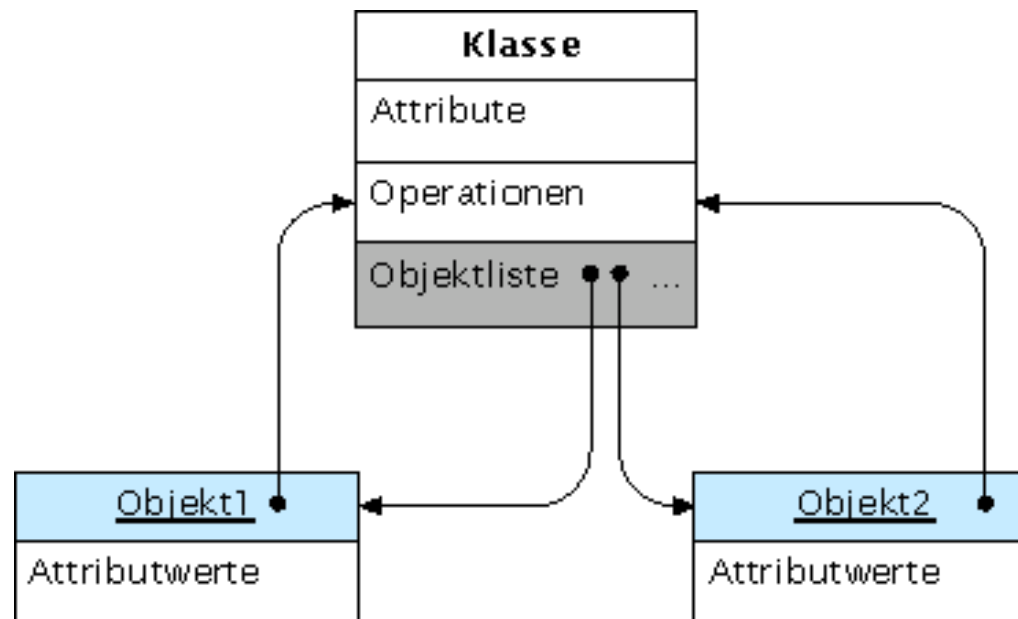


- **Objektverwaltung**

- Ein Objekt „kennt“ seine Klasse
  - findet z.B. dort die Operationen
- Eine Klasse „weiß“ **nicht**, welche Objekte sie „besitzt“ bzw. welche Objekte von ihr erzeugt wurden
- In der Analyse ist es praktisch, davon auszugehen, dass eine Klasse ihre Objekte kennt
- Objektverwaltung bedeutet, die Klasse „führt Buch“ über das Erzeugen und Löschen ihrer Objekte
- Die Klasse erhält die Möglichkeit, Anfragen und Manipulationen auf der Menge der Objekte einer Klasse durchzuführen (*class extension, object warehouse*)
- Die Objektverwaltung muss bei Bedarf im Entwurf und in der Implementierung realisiert werden!

- **Objektverwaltung**

- Wir gehen in der Anforderungsanalyse davon aus, dass eine Klasse auf die Liste aller ihrer Objekte zugreifen kann
- Operationen können sich auf diese Objektliste beziehen



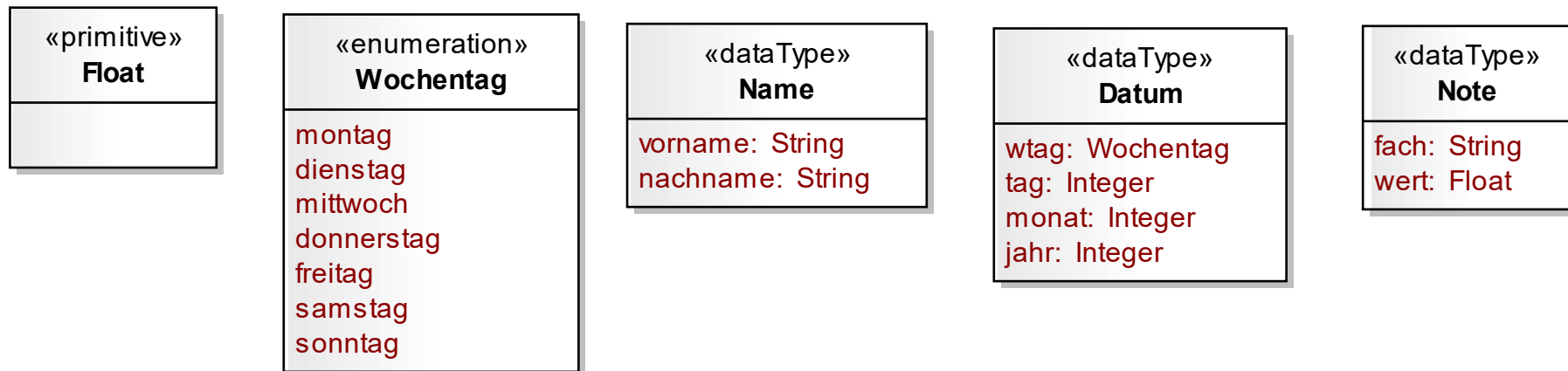
Quelle: [Balzert 05], Abb. 2.2-3

- **Aufgabe 1**

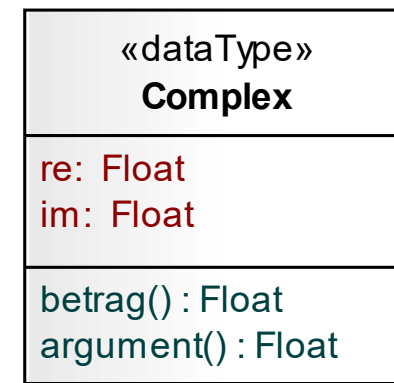
- Erstellen Sie ein Objektdiagramm, welches die folgende Momentaufnahme einer Situation in einer Bibliothek wiedergibt:
  - In der Bibliothek sind u.a. folgende Bücher vorhanden:
    - Ken Follet, Die Säulen der Erde, 1990,
    - Noah Gordon, Der Medicus, 1987 und
    - Nicholas Evans, Der Pferdeflüsterer, 1995
  - Für jeden Leser werden Name, Adresse und Geburtsdatum und dessen Nummer gespeichert. Hans Müller, geb. am 1.3.1975 aus Bochum leiht sich »Die Säulen der Erde« aus, das er spätestens am 12.5.1998 zurückgeben muss. Dieses Datum wird ins Buch eingetragen. Else Wallersee aus Dortmund, geb. am 26.3.1975 leiht sich »Der Medicus« und »Der Pferdeflüsterer«. Beide Bücher muß sie am 14.5.1998 zurückgeben.
- Erstellen Sie ein Klassendiagramm, welches die Klassen der identifizierten Objekte mit ihren Attributen zeigt (zunächst noch ohne Beziehungen).

- **Datentypen**

- Vordefinierte UML-Datentypen (*Primitive Datentypen*)  
Boolean, String, Integer, UnlimitedNatural
- Weitere **primitive Datentypen** können definiert werden als Klasse mit Stereotyp <<primitive>>
- **Strukturierte Datentypen** können definiert werden als Klassen mit Stereotyp <<data type>>
- **Aufzählungstypen** können definiert werden als Klasse mit Stereotyp <<enumeration>>



- Datentypen
  - Werden verwendet als Typen von Attributen
  - Werden verwendet als Typen von Parametern von Operationen
  - Instanzen von Datentypen besitzen keine eigenen Identität
    - Die Instanzen werden nicht als Objekte, sondern als Werte aufgefasst
    - Zwei Instanzen sind identisch, wenn Sie identische Attributwerte besitzen
  - Können neben Attributen auch Operationen besitzen
    - **Beispiel:** Datentyp für Komplexe Zahlen
      - Real- und Imaginärteil als Attribute
      - Umrechnung in Polarkoordinaten durch Operationen



- **Multiplizität (*multiplicity*)** für Attribute
  - definiert, wie viele Werte ein Attribut in einem Objekt besitzen kann
  - wird in eckigen Klammern angegeben
  - Beispiel:
    - `noten: Note [0..10]`  
→ Der Wert von `noten` enthält 0...10 Werte des Typs `Note`
  - Es gilt:
    - `[5..5] = [5]` : genau 5 Werte
    - `[0..*] = [*]` : null bis beliebig viele Werte
    - `[1..1] = [1]` : genau ein Wert
      - **Voreinstellung**, wenn keine Multiplizität angegeben wird
    - `[0..1]` : null oder ein Wert

Student
matrikelnummer: String name: Name geburtsdatum: Datum immatrikulation: Datum vorprüfung: Datum [0..1] noten: Note [0..*]

- **Anfangswert** für Attribute

- Der **Anfangswert** (*initial value*) definiert den Wert, den ein Attribut beim Erzeugen des zugehörigen Objekts erhält
- Kann später geändert werden
- Beispiel:

```
rechnungsdatum: Date = heute
```

- **Einschränkung**

- Eine **Einschränkung** (*constraint*) ist eine Invariante bzw. eine Zusicherung, die immer wahr sein muss.
- Werden in geschweiften Klammern angegeben
- Können sich auf ein oder mehrere Attribute beziehen
- Beispiele:

```
{geburtsdatum <= aktuelles Datum}  
{vordiplom > immatrikulation > geburtsdatum}  
{verkaufspreis >= 1.5 * einkaufspreis}
```

- **Eigenschaftswerte** für Attribute
  - **Eigenschaftswerte** (*property values*) spezifizieren zusätzliche Eigenschaften von Attributen
  - Werden in geschweiften Klammern angegeben
  - Mehrere Werte werden durch Kommata getrennt
  - Beispiele :
    - `{readOnly}`: das Attribut darf nicht verändert werden  
z.B.: `pi: Float = 3.1415 {readOnly}`
    - `{key}` : der Wert des Attributs identifiziert das sein Objekt eindeutig („Schlüsselattribut“)  
z.B.: `matrNr: String {key}`
    - `{ordered}`: die Werte des Attributs sind geordnet
    - `{unique}`: die Werte des Attributs sind duplikatfrei
      - Bei Attributen mit Multiplizität >1
  - Mehrere Eigenschaftswerte können kombiniert werden  
z.B.: `matrNr: String {key, readOnly}`



- **Attributspezifikationen**

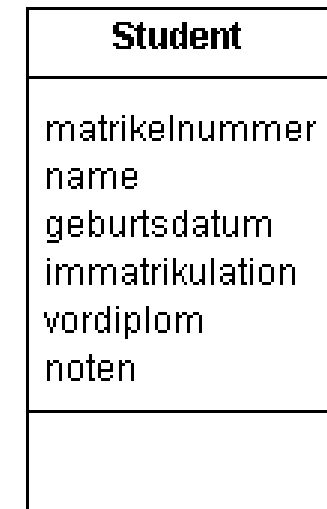
- Genaue Spezifikation aller Eigenschaften von Attributen
- Nicht immer notwendig bei der Anforderungsanalyse, aber spätestens im Entwurf
- Kann auch außerhalb des Klassendiagramms erfolgen
- Beispiel: Spezifikation der Attribute der Klasse `Student`

- **Attribute**

```
matrikelnummer: String {readOnly, key}  
name: Name  
geburtsdatum: Datum  
immatrikulation: Datum  
vordiplom: Date [0..1]  
noten: Note [0..*]
```

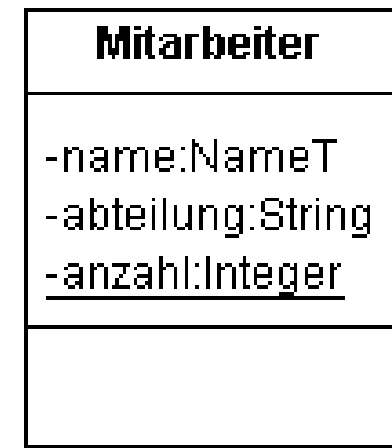
- **Restriktionen:**

```
{geburtsdatum < immatrikulation < vordiplom}
```



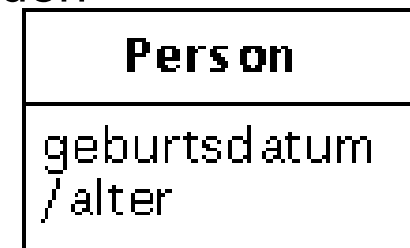
- **Klassenattribute**

- Für ein Klassenattribut existiert nur ein Attributwert für alle Objekte einer Klasse
- Klassenattribute existieren auch, wenn es zu einer Klasse (noch) keine Objekte gibt
- Kennzeichnung durch Unterstreichen
  - klassenattribut
- Beispiel:
  - Die Anzahl der Objekte einer Klasse:



- **Abgeleitete Attribute**

- Der Wert von **abgeleiteten Attributen** (*derived attributes*) kann jederzeit aus anderen Attributwerten berechnet werden
- Kennzeichnung mit dem Präfix “/”



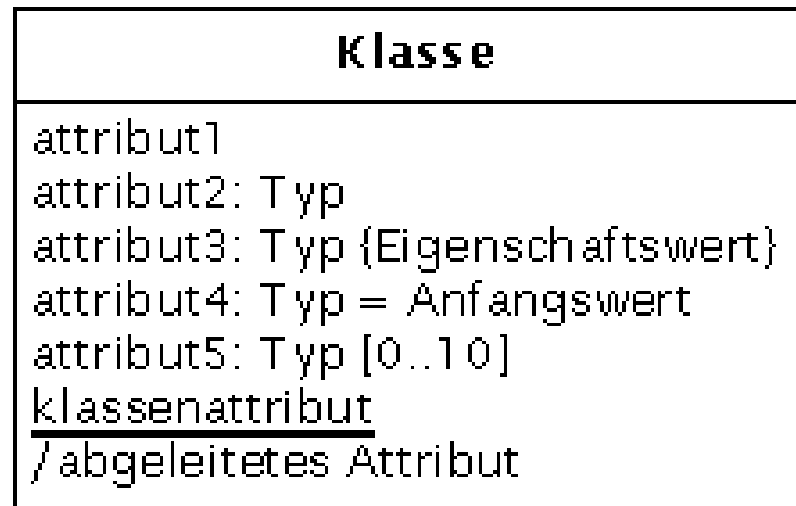
- **Aufgabe 2**

- Erstellen Sie ein Klassendiagramm, welches Klassen mit vollständigen Attributspezifikationen enthält, die folgenden Sachverhalt modellieren:
  - An einer Hochschule sind studentische Hilfskräfte und Angestellte zu verwalten. Für alle Personen sind der Name, bestehend aus Vor- und Nachname, und die Adresse, bestehend aus PLZ, Ort und Straße, zu speichern. Für studentische Hilfskräfte sind außer der Matrikelnummer auch Beginn und Ende aller Arbeitsverträge sowie die jeweilige wöchentliche Stundenzahl einzutragen. Alle studentischen Hilfskräfte erhalten den gleichen Stundenlohn. Für jeden Angestellten wird das Eintrittsdatum gespeichert.

- **Aufgabe 3**

- Definieren Sie die Klasse Videofilm mit Attributen, so dass folgende Sachverhalte modelliert sind:
  - In einem Videoverleih werden für Videofilme folgende Informationen festgehalten: Titel des Films, Laufzeit und Erscheinungsjahr. Jeder Videofilm besitzt eine individuelle Ausleihgebühr. Wird ein Film beschädigt zurückgegeben, so ist eine fixe Entschädigungsgebühr zu entrichten (für alle Filme gleich). Außerdem soll die Anzahl aller Videofilme der Videothek festgehalten werden.

- Zusammenfassung: **Notation** von Attributen in UML
  - Alle Bestandteile optional, außer Attributname



Quelle: [Balzert 05], Abb. 2.3-2

- Anmerkung:
  - Den Attributen können zusätzlich **Sichtbarkeiten** vorangestellt werden:
    - public (+), private (-), protected (#), package(~)
  - Diese spielen erst in der Entwurfsphase eine Rolle

- **Operationen**

- Eine **Operation** beschreibt eine Tätigkeit (Aktivität, Verhalten), die von Objekten einer Klasse ausgeführt werden kann
- Alle Objekte einer Klasse verwenden dieselben Operationen
- Jede Operation kann auf alle Attribute eines Objekts dieser Klasse direkt zugreifen
- Die Menge aller Operationen wird als das Verhalten der Klasse oder als die **Schnittstelle** der Klasse bezeichnet
- Der Operationsname beginnt mit einem Kleinbuchstaben und darf beliebige Zeichen enthalten

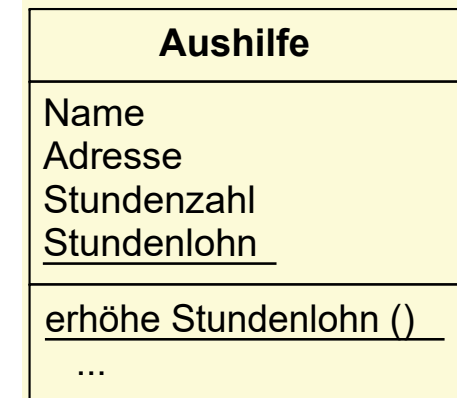
- Operationen
  - Beispiel:

Student
matrikelnummer name geburtsdatum immatrikulation vordiplom noten <u>anzahl</u>
immatrikulieren() exmatrikulieren() drucke Studienbescheinigung() notiere Noten() berechne Durchschnitt() <u>drucke Vordiplomliste()</u> anmelde Praktikum() drucke Prakt.bescheinigung()

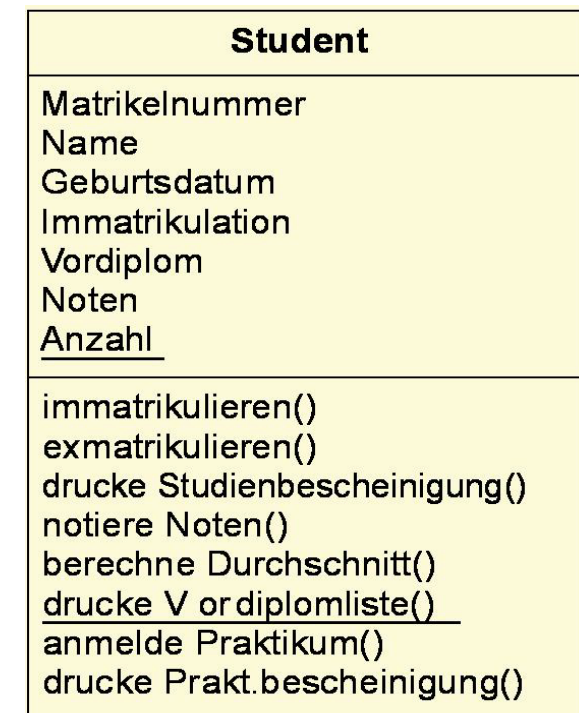
Firma
name ort anzahl Mitarbeiter branche

Quelle: [Balzert 05], Abb. 2.4-1

- Grundsätzliche Operationsarten
  - Objektoperation bzw. Operation
    - `drucke Studienbescheinigung()`
    - `exmatrikulieren()`
    - `immatrikulieren()`
      - Zugriff auf ein Objekt der Klasse
  - Klassenoperation
    - `drucke Vordiplomliste()`
      - Alle Objekte einer Klasse (Objektverwaltung)
    - `erhöhe Stundenlohn()`
      - Zugriff auf Klassenattribute



Quelle: [Balzert 05], Abb 2.4-3



Quelle: [Balzert 05], Abb 2.4-1



- Externe und interne Operationen
  - Externe Operation ...
    - wird direkt vom Benutzer aktiviert
    - kann weitere – interne – Operationen aufrufen
  - Interne Operation ...
    - wird von anderen Operationen aktiviert
    - wird nur dann in das Klassendiagramm eingetragen, wenn es für das Verständnis notwendig ist
  - Ziel der Systemanalyse ist es, alle externen Operationen zu ermitteln

- **Notation UML**

- Operation:

- `operationsname (Parameterliste) : ErgebnisTyp`

- Parameterliste

- `p-1: Typ-1, ..., p-n:Typ-n`

- Vor jedem Parameter kann zusätzlich eine **Parameterrichtung** angegeben werden
      - `in`, `out`, `inout`
    - Den Operationen können zusätzlich **Sichtbarkeiten** vorangestellt werden:
      - `public (+)`, `private (-)`, `protected (#)`, `package (~)`
    - Parameterlisten/Ergebnistyp werden in der Anforderungsanalyse häufig noch nicht präzisiert (→ Entwurf)
    - Klassenoperationen werden unterstrichen dargestellt

- Klassifikation von Operationen

- *accessor operation*

- lesender Zugriff auf Objektattribute
      - `drucke Studienbescheinigung()`

- *update operation*

- schreibender Zugriff auf Objektattribute
      - `notiere Note()`

- Durchführung von Berechnungen

- `berechne Durchschnitt()`

- *constructor operation / destructor operation*

- Objekte erzeugen bzw. initialisieren / Objekte löschen
      - `immatrikulieren()`
      - `exmatrikulieren()`

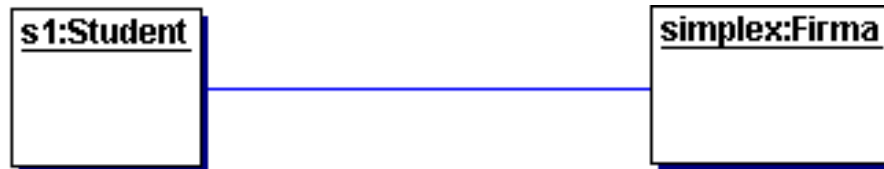
Alternativ: *create operation / destroy operation*

Student
Matrikelnummer Name Geburtsdatum Immatrikulation Vordiplom Noten <u>Anzahl</u>
immatrikulieren() exmatrikulieren() drucke Studienbescheinigung() notiere Noten() berechne Durchschnitt() <u>drucke V or diplomliste()</u> anmelde Praktikum() drucke Prakt.bescheinigung()

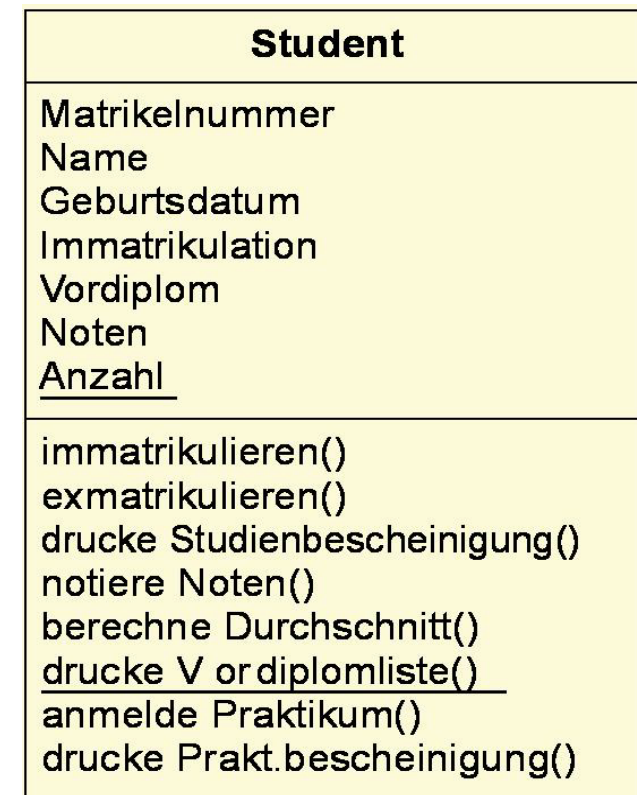
Quelle: [Balzert 05], Abb 2.4-1

- Klassifikation von Operationen
  - *query operation* bzw. *select operation*
    - Objekte einer Klasse nach bestimmten Kriterien auswählen
      - `drucke Vordiplomliste()`
    - → Klassenoperation!

- *connect operation*
  - Verbindung herstellen zu anderen Objekten



- Operationen, die Operationen anderer Klassen aktivieren
  - `drucke Praktikumsbescheinigung()`



Quelle: [Balzert 05], Abb 2.4-1

- Stereotype zum Gruppieren von Operationen



- *Stereotyp-Bezeichnungen sind frei wählbar!*
- *UML definiert Standard-Stereotypen*

- Verwaltungsoperationen
  - **new()** : Erzeugen eines neuen Objekts
  - **delete()** : Löschen eines Objekts
  - **setAttribute()** : Attributwert setzen
  - **getAttribute()** : Attributwert lesen
  - **link()** : Aufbauen einer Beziehung zu einem anderen Objekt
  - **unlink()** : Abbauen einer Beziehung zu einem anderen Objekt
  - **getlink()** : Zugriff auf ein verbundenes Objekt

## Anmerkungen

- In der Analysephase werden Verwaltungsoperationen nicht ins Klassendiagramm eingetragen
- Können dennoch in anderen Diagrammen (z.B. in Interaktionsdiagrammen) verwendet werden

- **Aufgabe 4**

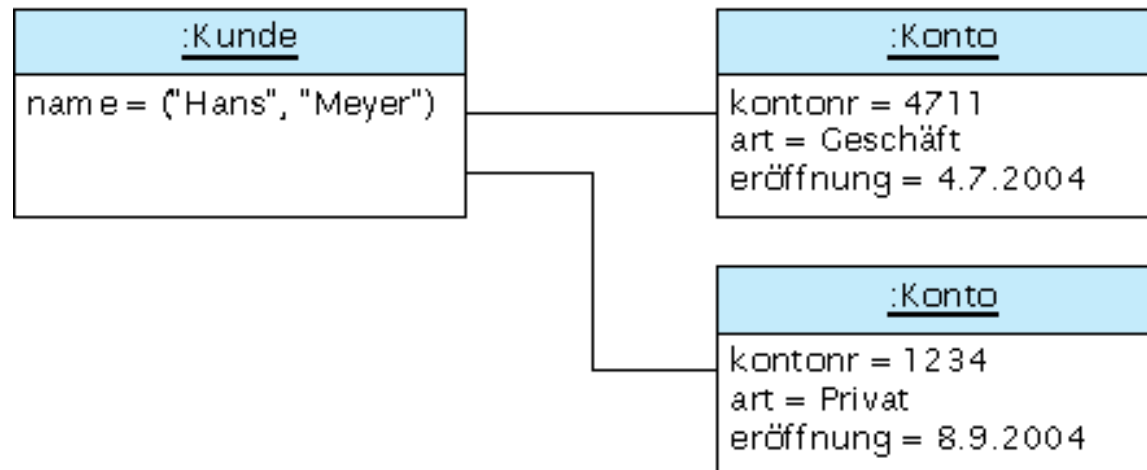
- Definieren Sie eine Klasse Artikel mit Attributen und Operationen, so dass die folgenden Sachverhalte modelliert sind:
  - Eine Artikelverwaltung ist für ein Warenhaus ist mit EDV zu unterstützen. Jeder Artikel besitzt eine eindeutige Nummer, eine Bezeichnung, einen Einkaufs- und einen Verkaufspreis. Neue Artikel müssen erfasst und bei vorhandenen Artikeln die Preise geändert werden. Artikelzu- und -abgänge müssen gebucht werden können. Ist der Mindestbestand von Artikeln unterschritten, so muss für alle betreffenden Artikel ein Bestellvorschlag gedruckt werden, der jeden Artikel bis zum Maximalbestand auffüllt. Außerdem soll eine Liste aller Artikel erstellt werden.

- Eine **Assoziation** modelliert Verbindungen (*links*) zwischen Objekten einer oder mehrerer Klassen
  - Beispiele
    - Eine Person besitzt Autos
    - Eine Person fährt ein Auto
    - Ein Mitarbeiter ist bei einer Firma angestellt
    - Ein Leser hat ein Buch ausgeliehen
    - Ein Kunde besitzt ein Konto
    - Ein Steuergerät überwacht Sensoren
- Eine **reflexive Assoziation** besteht zwischen Objekten derselben Klasse
  - Beispiele
    - Eine Person ist Verwandter einer anderen Person
    - Eine Person ist Vater einer anderen Person
    - Ein Mitarbeiter ist Vorgesetzter eines anderen Mitarbeiters



- Beispiel

Objektdiagramm



Klassendiagramm



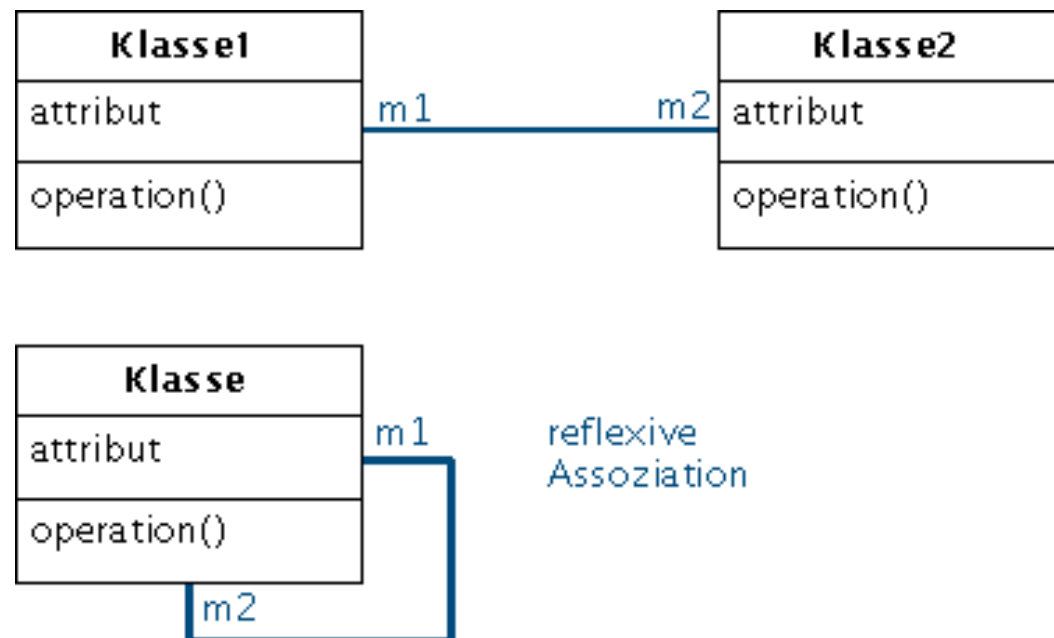
Quelle: [Balzert 05], Abb 2.5-1

- Notation UML
  - Binäre Assoziation
    - Linie zwischen einer oder zwei Klassen
    - An jedem Ende der Linie steht die **Multiplizität** (*multiplicity*)

## Multiplizität

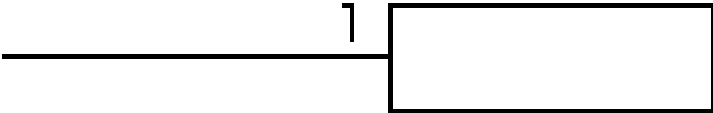
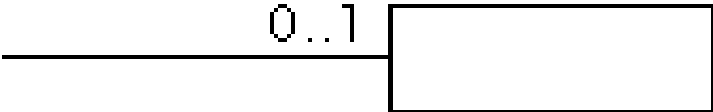
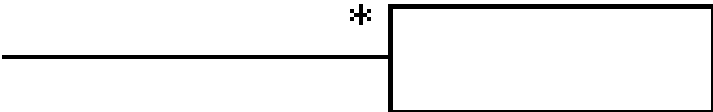
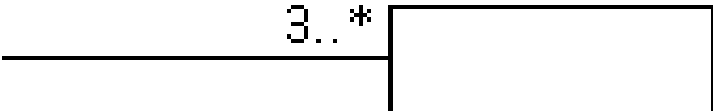
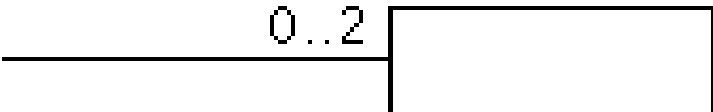
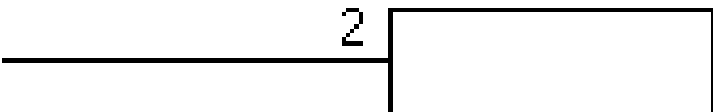
Mit einem Objekt der Klasse **Klasse1** können **m2** Objekte der Klasse Klasse2 verknüpft sein.

Mit einem Objekt der Klasse **Klasse2** können **m1** Objekte der Klasse **Klasse1** verknüpft sein.



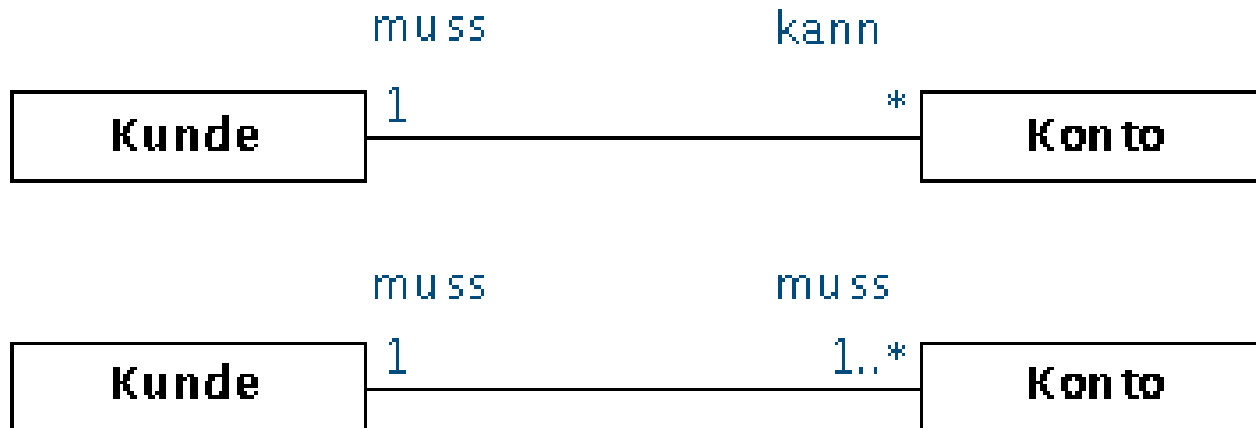
Quelle: [Balzert 05], Abb 2.5-2

- Notation Multiplizität

1		genau 1
0..1		0 bis 1
*		0 bis viele
3..*		3 bis viele
0..2		0 bis 2
2		genau 2

Quelle: [Balzert 05], Abb 2.5-3

- Muss- und Kann-Assoziation
  - Kann-Assoziation
    - Untergrenze: Multiplizität 0
  - Muss-Assoziation
    - Untergrenze: Multiplizität 1 oder größer



Quelle: [Balzert 05], Abb 2.5-4

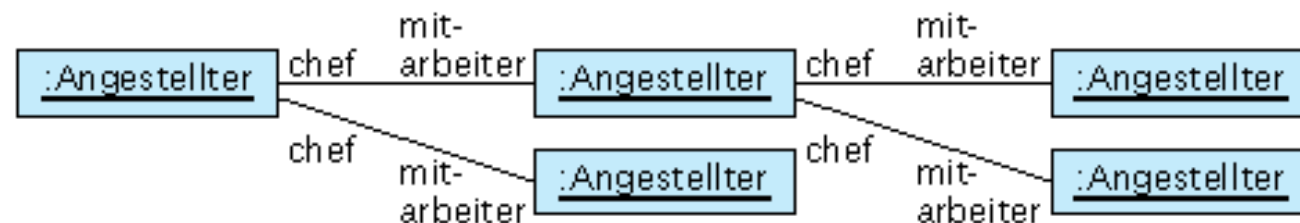
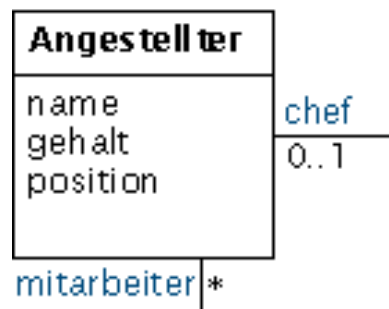
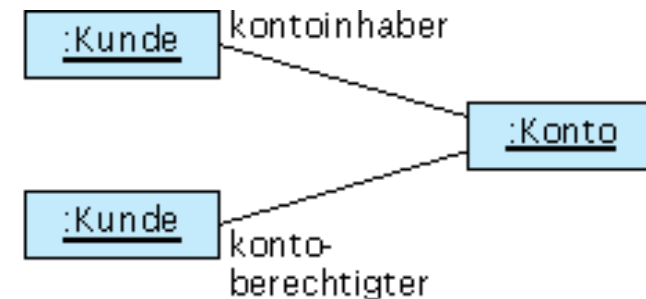
- **Name** einer Assoziation
  - Beschreibt meistens nur eine Richtung der Assoziation
  - Ein schwarzes Dreieck gibt die Leserichtung an
  - Name darf fehlen, wenn die Bedeutung der Assoziation offensichtlich ist



Quelle: [Balzert 05], Abb 2.5-1

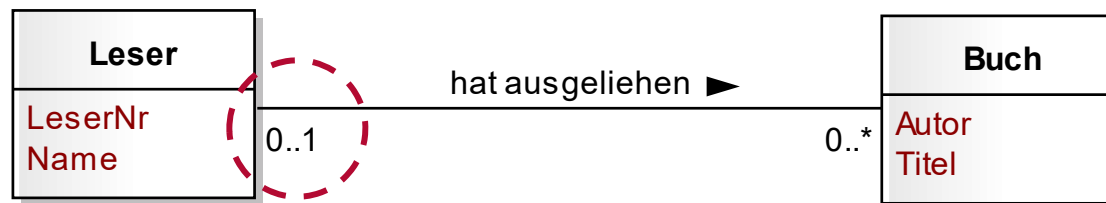
- **Rollenname**

- Beschreibt Bedeutung eines Objekt in einer Assoziation
- Binäre Assoziationen besitzen maximal zwei Rollen
- Er wird an das Ende der Assoziation geschrieben bei der Klasse, deren Bedeutung in der Assoziation sie beschreibt
- Tragen zur Verständlichkeit des Modells mehr bei als der Name der Assoziation

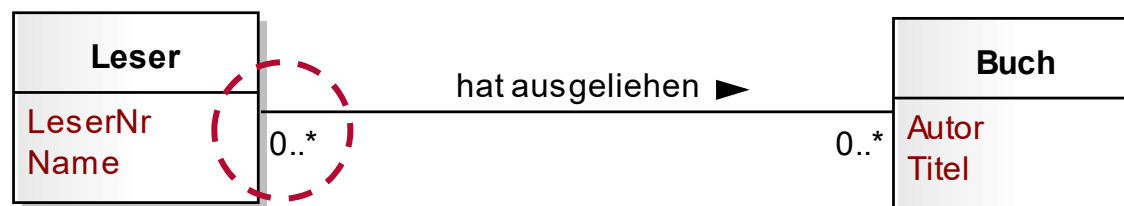


Quelle: [Balzert 05], Abb 2.5-5

- „Schnappschuss“ oder Historie?
  - **Schnappschuss**: das System gibt Auskunft über die aktuelle Situationen/Vorgänge
    - Welcher Leser hat gerade ein Buch ausgeliehen?



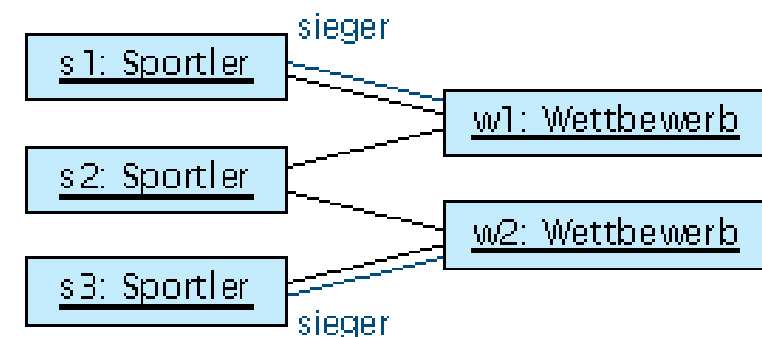
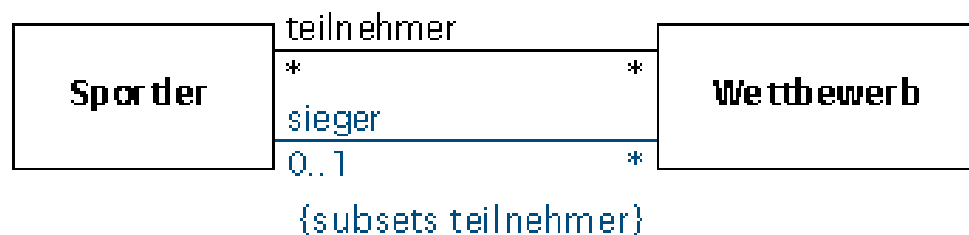
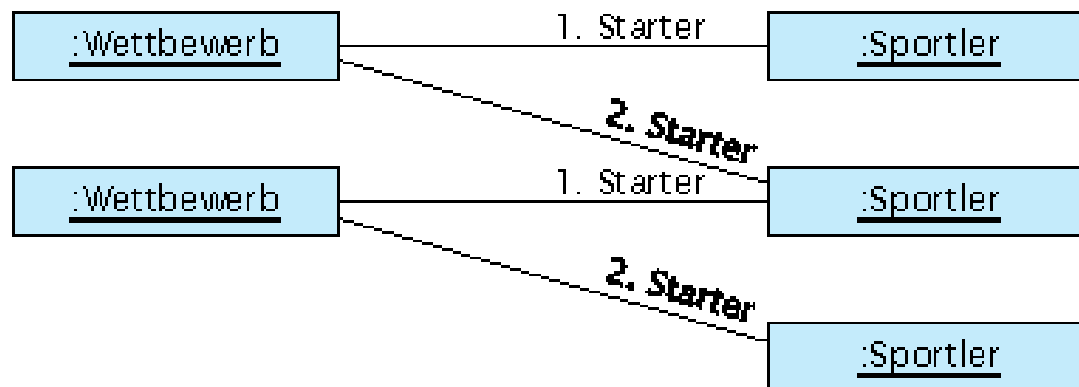
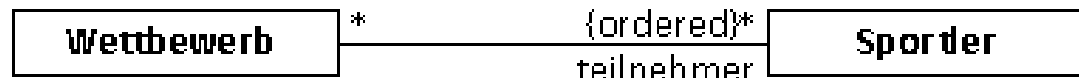
- **Historie**: das System gibt auch Auskunft über vergangene Situationen/Vorgänge
  - Von welchen Lesern wurde ein Buch bereits ausgeliehen?



- **Eigenschaftswert** (*property string*)
  - An ein Assoziationsende kann ein Eigenschaftswert (*property string*) angetragen werden
  - Beispiele:
    - {subsets Eigenschaft}: beschreibt eine Teilmenge von Objektbeziehungen
    - {ordered}: definiert eine Ordnung auf der Menge der Objektbeziehungen (Multiplizität > 1)
      - Keine Aussage über die Definition der Ordnung (z.B. zeitlich, alphabetisch)

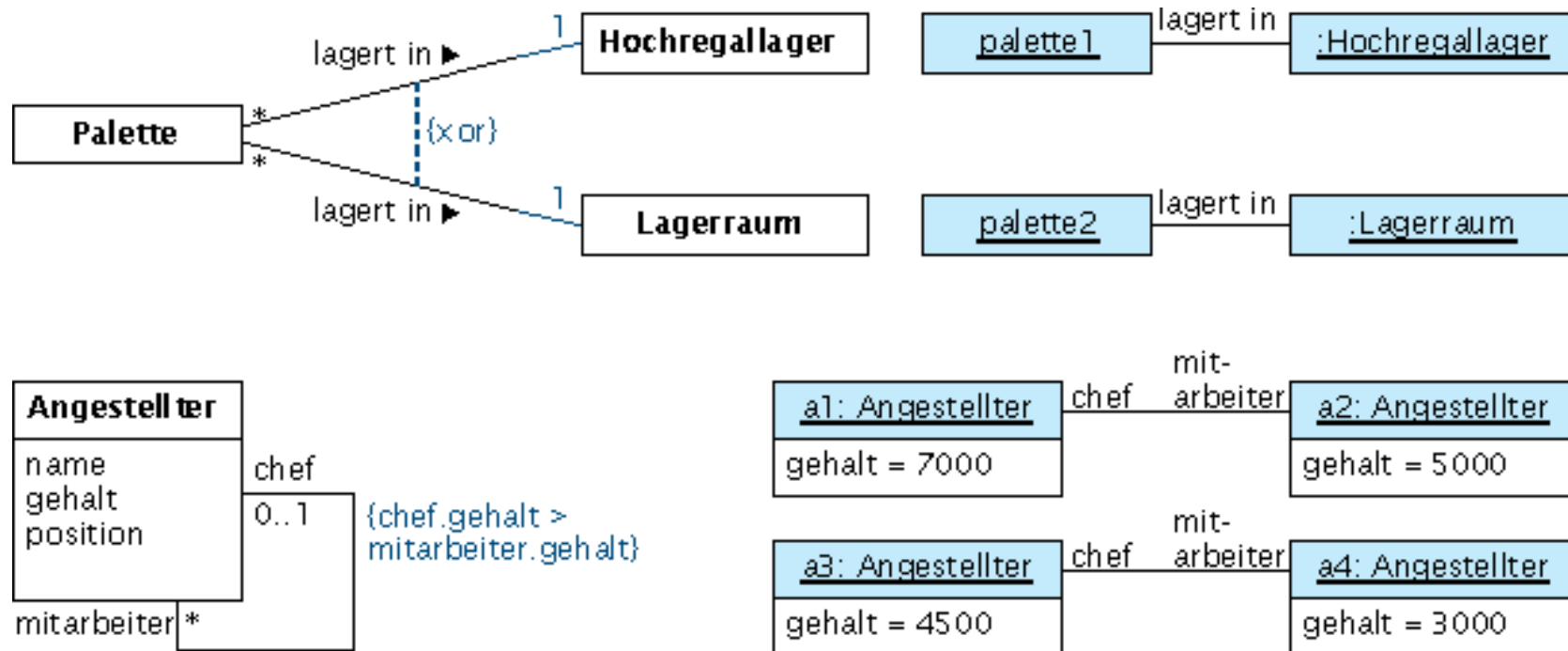


- Beispiele für Eigenschaftswerte



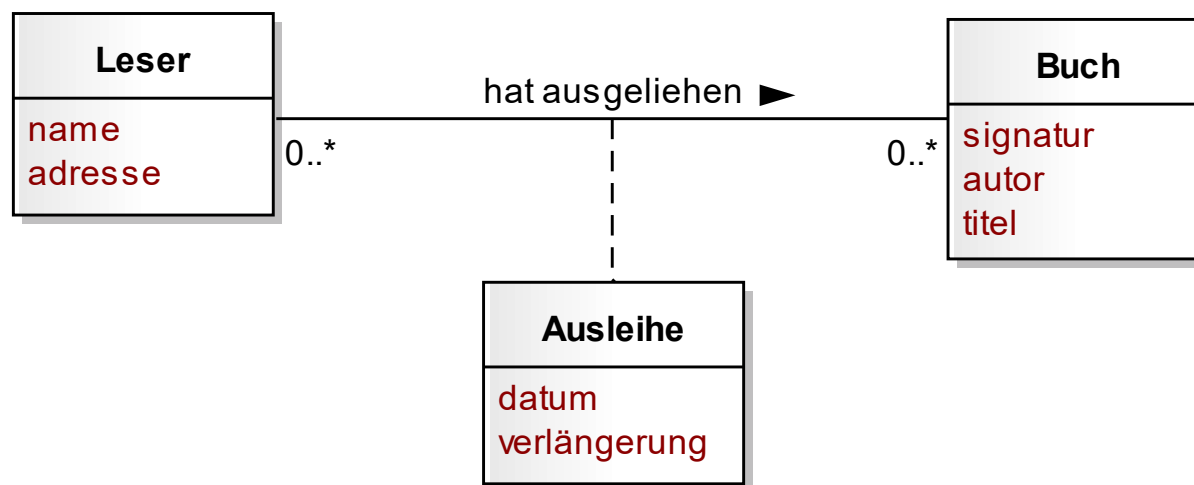
Quelle: [Balzert 05], Abb 2.5-6

- **Einschränkung (constraint)**
  - Eine Einschränkung (constraint) ist eine Zusicherung, die immer wahr sein muss
    - {xor}: sagt aus, dass zu einem Zeitpunkt genau eine der gekennzeichneten Assoziationen gilt
    - Andere Einschränkungen können durch beliebige Ausdrücke formuliert werden

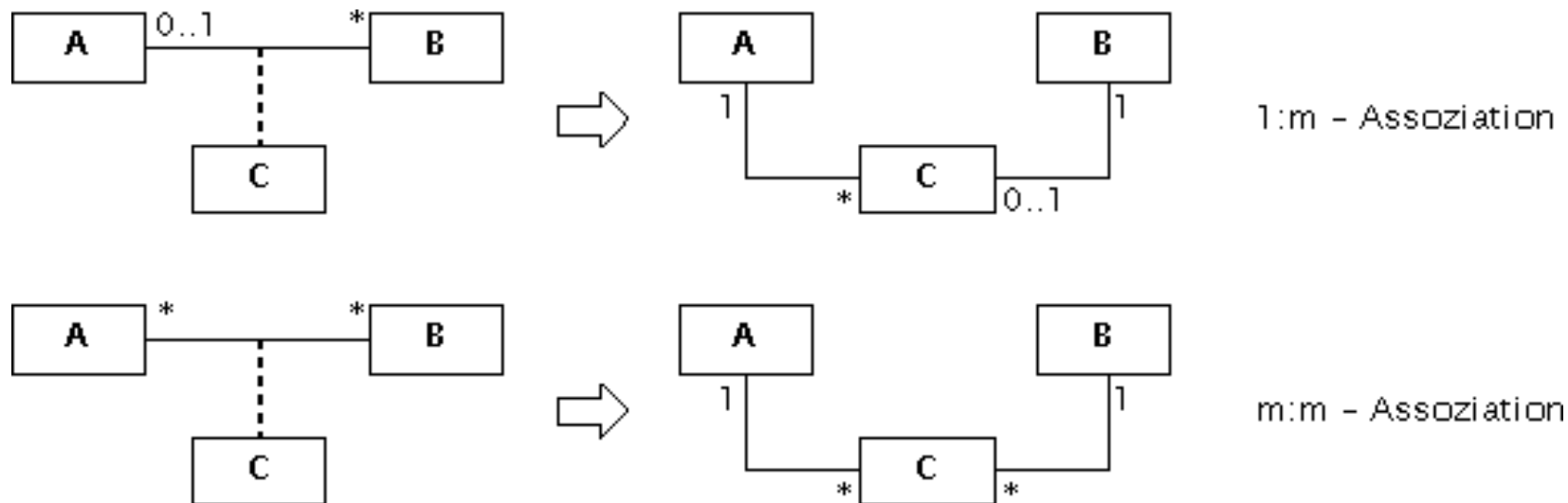


Quelle: [Balzert 05], Abb 2.5-7

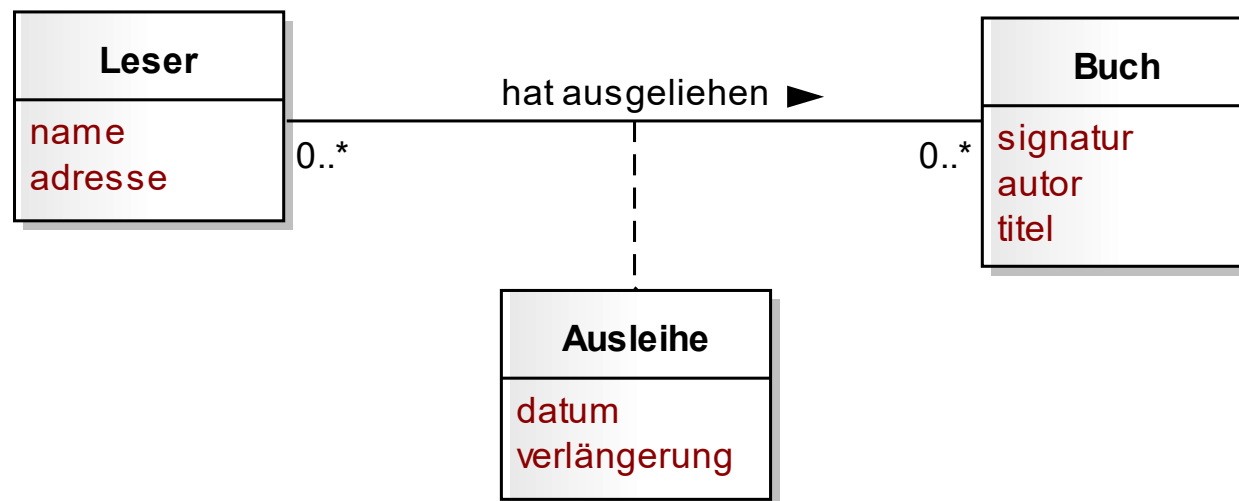
- **Assoziationsklasse** (*association class*)
  - Assoziationen können zusätzlich die Eigenschaften einer Klasse besitzen (Attribute, Operationen)
  - Insbesondere bei der Modellierung von Historien
    - Für jede erstellte Verbindung zwischen zwei Objekten werden spezielle Eigenschaften gespeichert, die weder dem einen noch dem anderen Objekt zugeordnet werden können
      - z.B. der Zeitpunkt, wann die Verbindung erstellt wurde



- Assoziationsklasse (*association class*)
  - Eine Assoziationsklasse kann in eine normale Klasse transformiert werden.
    - Die Assoziationsklasse (C) *koordiniert* die Beziehung der beiden an der Assoziation beteiligten Klassen (A,B),
    - Die Klasse C wird auch **Koordinator-Klasse** genannt



- Assoziationsklasse vs. Koordinator-Klasse – Beispiel
  - Mit Assoziationsklasse



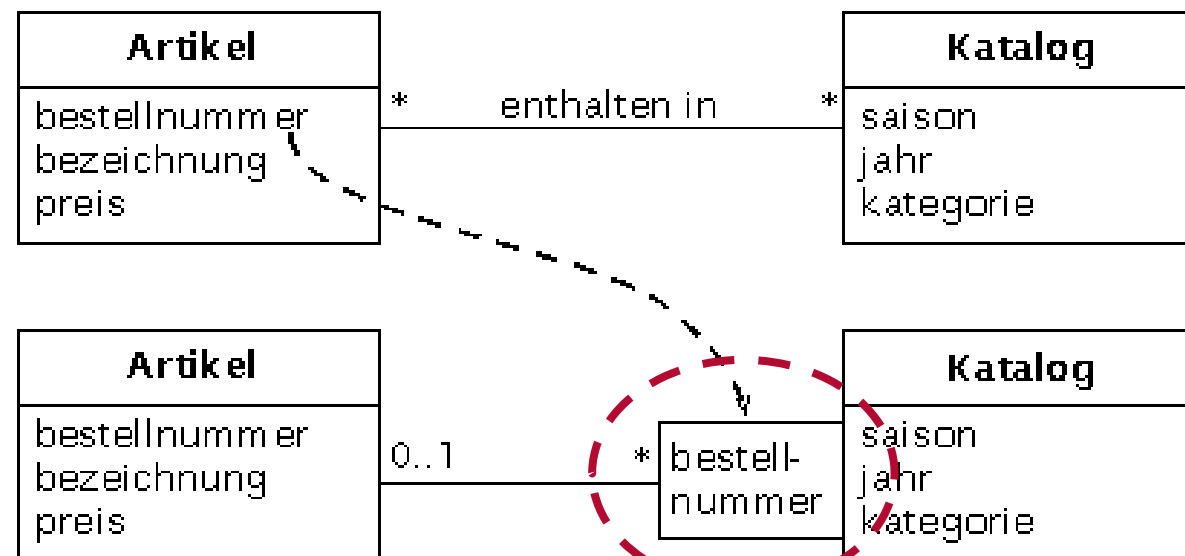
- Mit Koordinator-Klasse



- **Qualifizierte Assoziation** (*qualifier*)

- Spezielles Attribut, dessen Wert ein oder mehrere Objekte auf der anderen Seite selektiert
- Qualifizierung zerlegt die Objekte der gegenüberliegenden Klasse in Partitionen
- Qualifikationsangaben verändern die Multiplizität
  - Sie gibt an, wie groß die Partitionen sein können

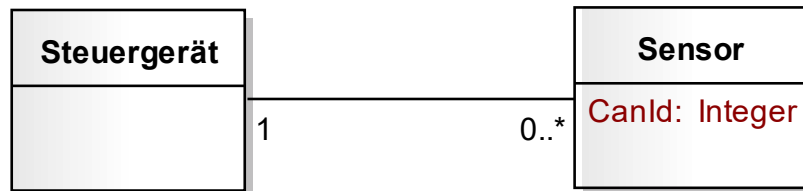
– Beispiel



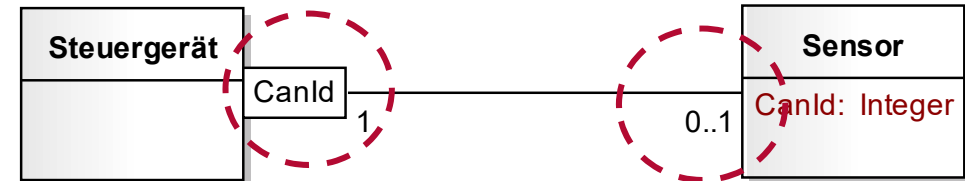
**Qualifikationsangabe**

Quelle: [Balzert 05], Abb 2.5-10

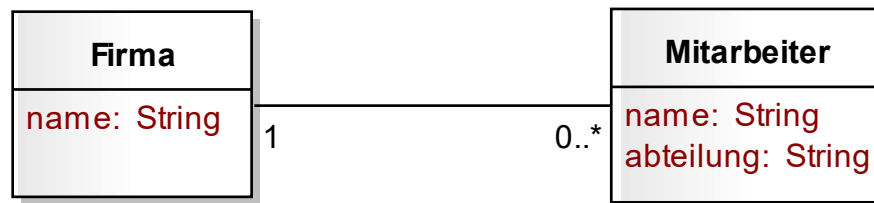
- Qualifizierte Assoziation - Beispiele



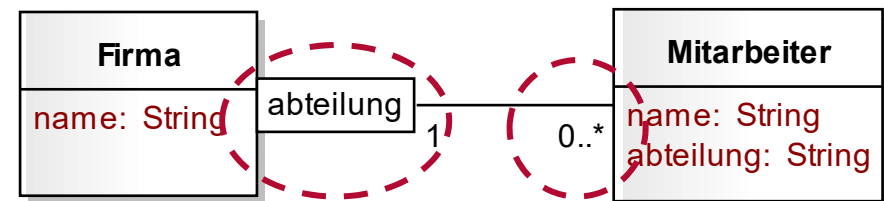
Ohne Qualifikationsangabe



**Mit Qualifikationsangabe:** die Multiplizität reduziert sich zu 0..1.



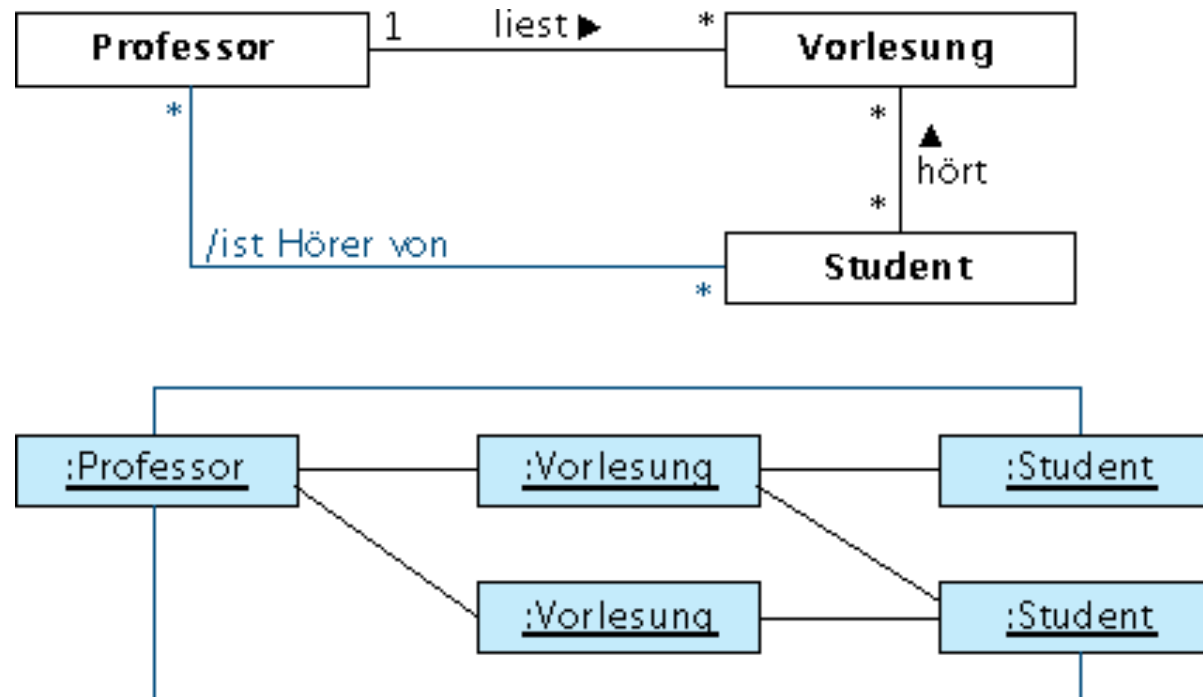
Ohne Qualifikationsangabe



**Mit Qualifikationsangabe:** die Multiplizität reduziert sich zwar, ist aber immer noch „viele“.

- **Abgeleitete Assoziation**

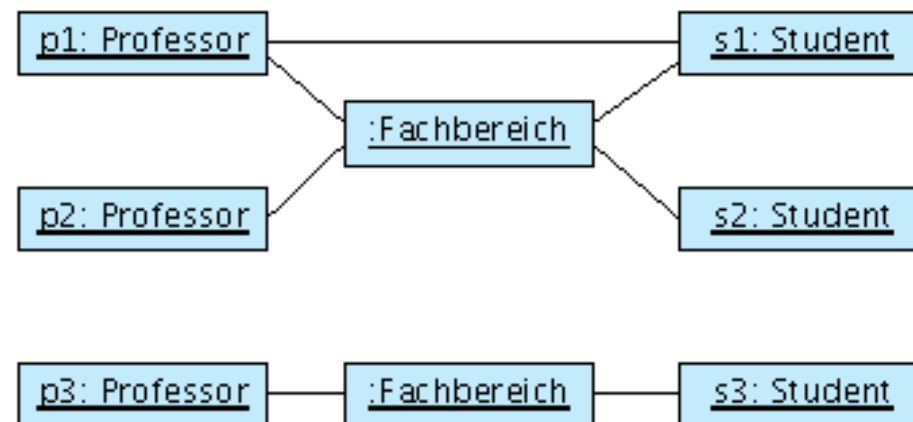
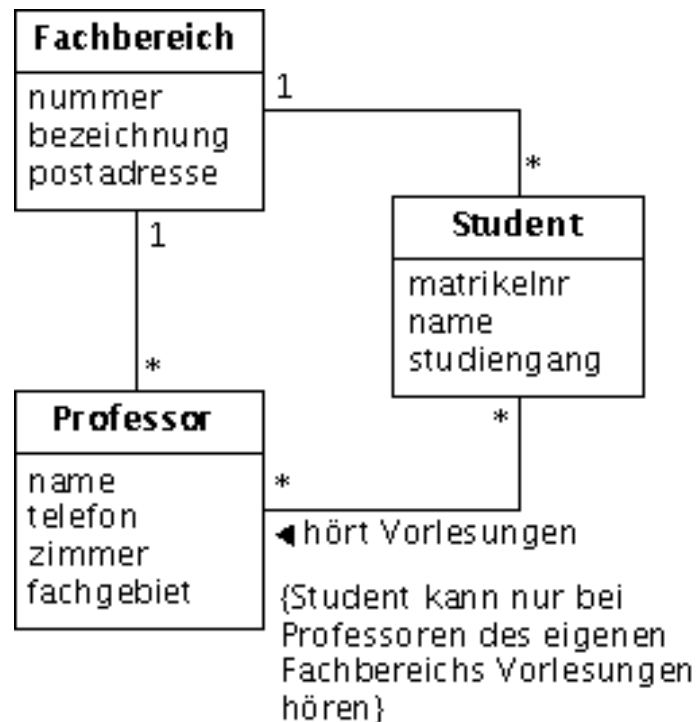
- Bei einer abgeleiteten Assoziation (*derived association*) werden die Abhängigkeiten bereits durch andere Assoziationen beschrieben



Quelle: [Balzert 05], Abb 2.5-11

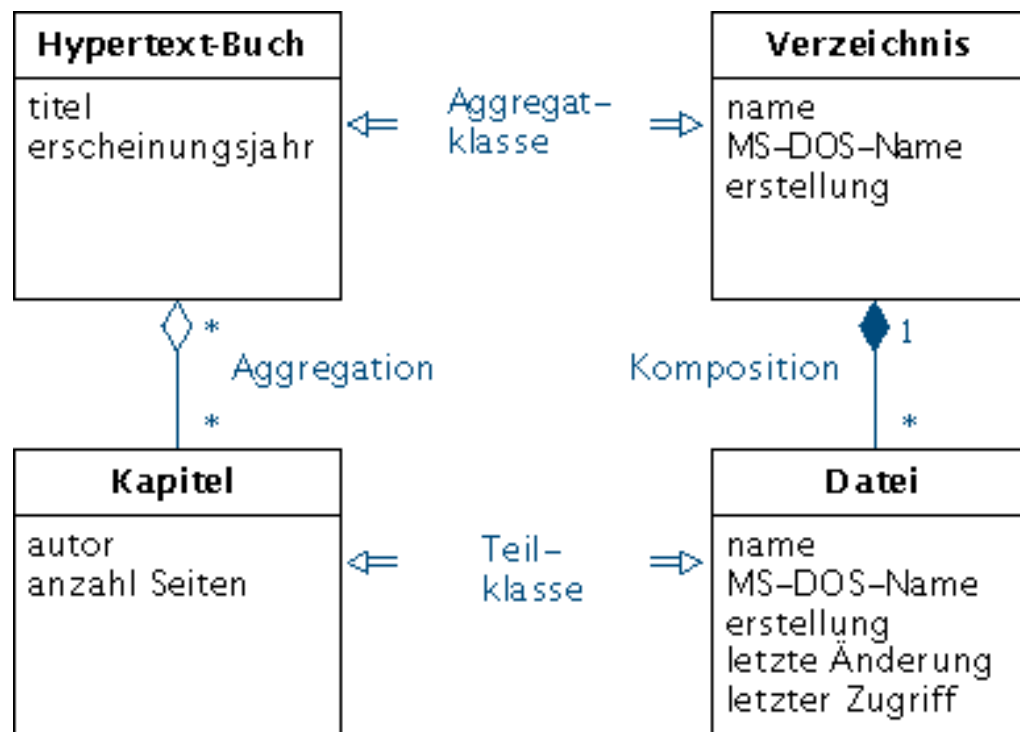


- Abgeleitete Assoziation
  - **Keine** abgeleitete Assoziation:



Quelle: [Balzert 05], Abb 4.5-2

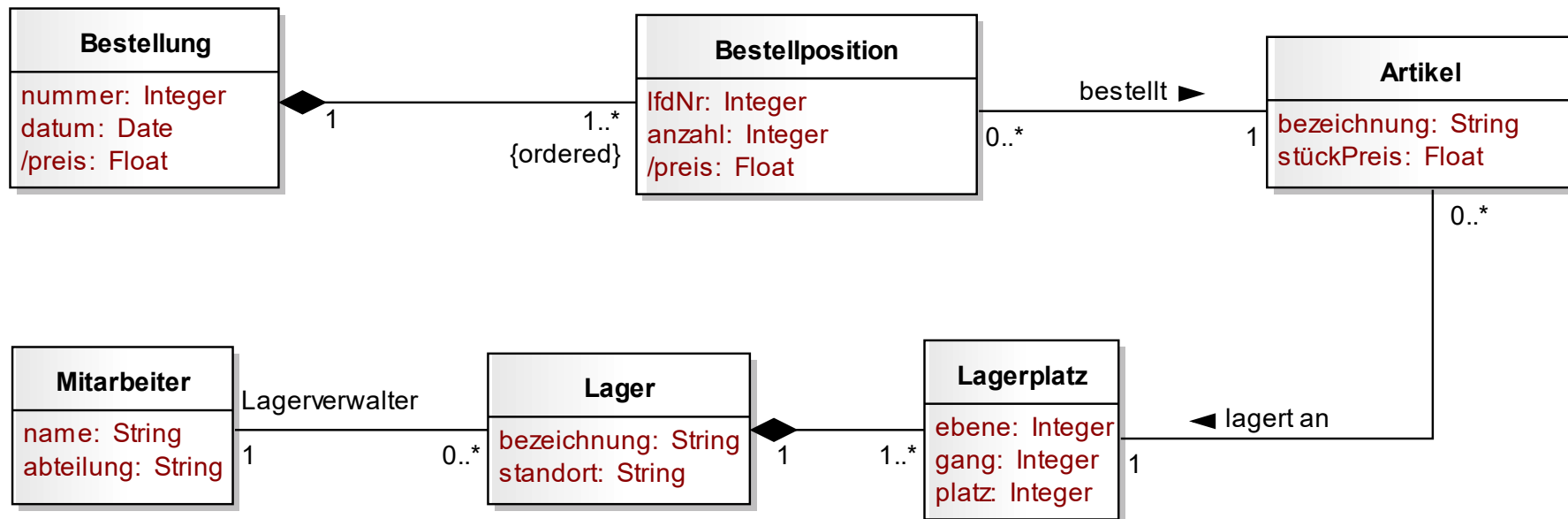
- 3 Arten von Assoziationen in der UML
  - Einfache Assoziation (*ordinary association*)
  - **Aggregation** (*aggregation*)
  - **Komposition** (*composite aggregation*)



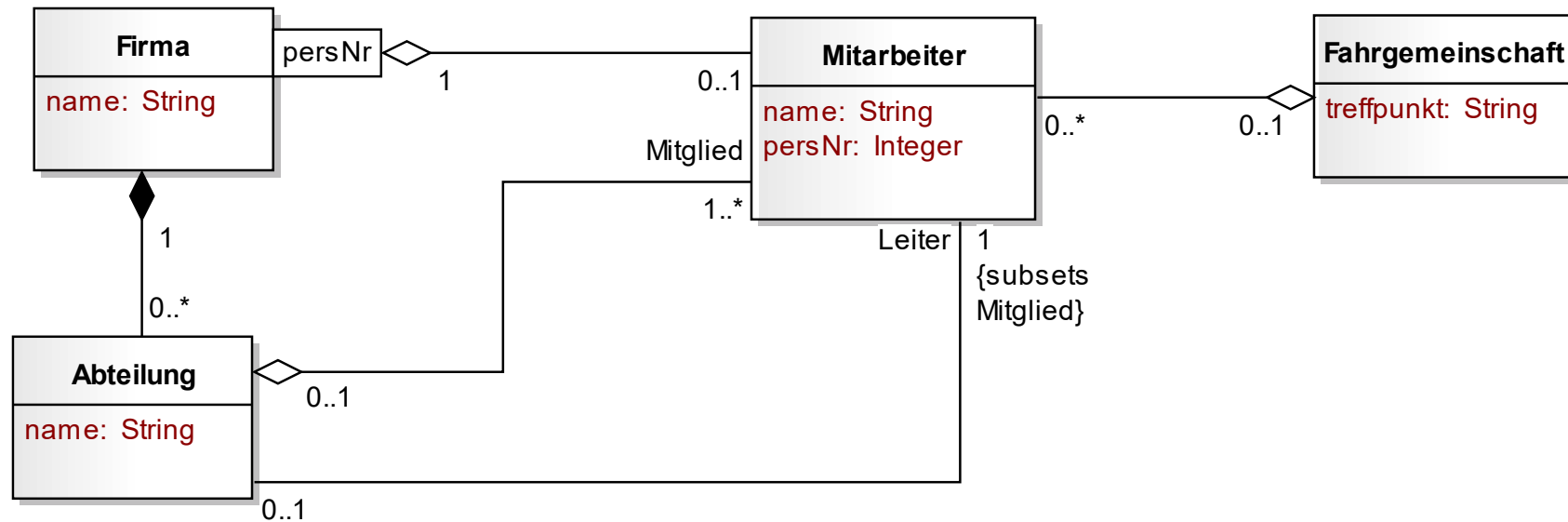
Quelle: [Balzert 05], Abb 2.5-12

- Aggregation
  - Läßt sich durch **ist Teil von** bzw. **besteht aus** beschreiben (Ganzes und Teile, *whole part*)
- Komposition
  - »starke« Aggregation
  - Multiplizität der Aggregatklasse  $\leq 1$
  - Das Ganze ist verantwortlich für das Erzeugen und Löschen seiner Teile
  - Wird das Ganze gelöscht, werden automatisch seine Teile gelöscht (*they live and die with it*)
  - Ein Teil darf auch anderem Ganzen zugeordnet werden

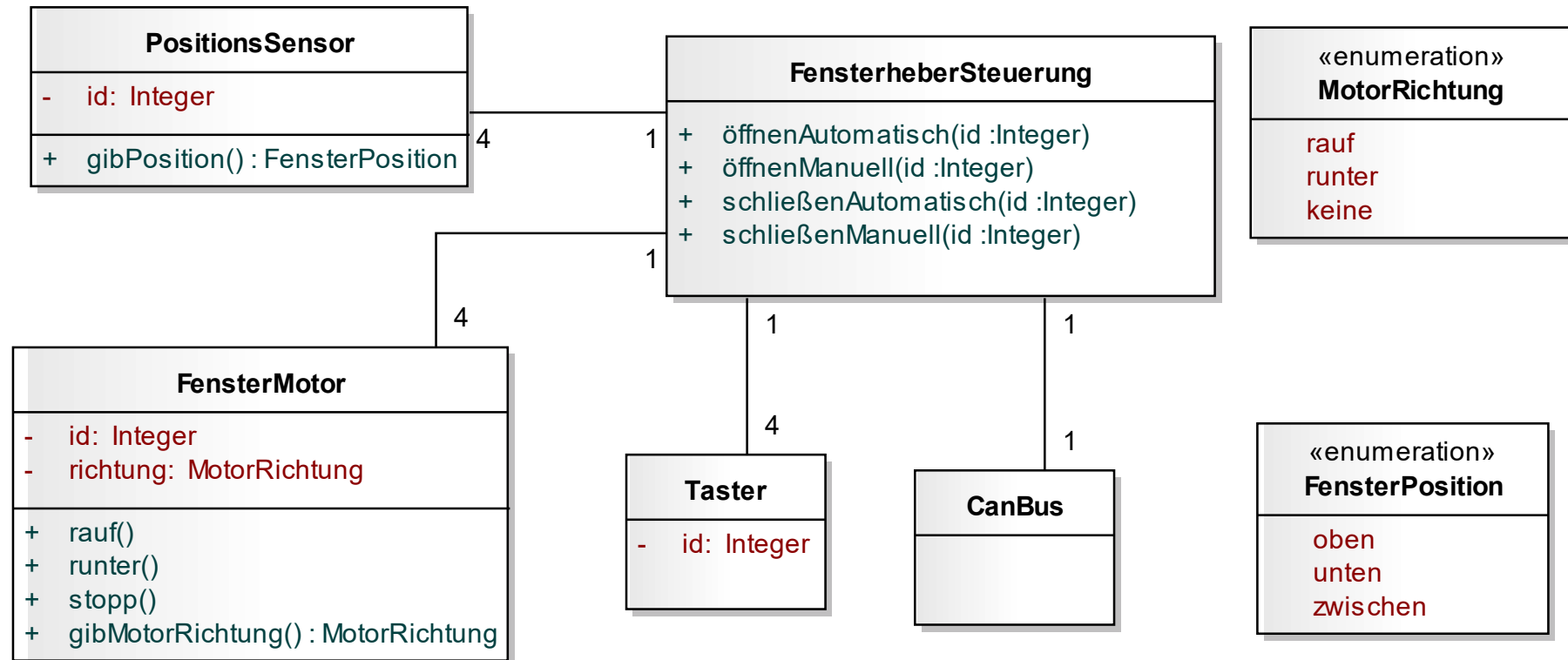
- Beispiel



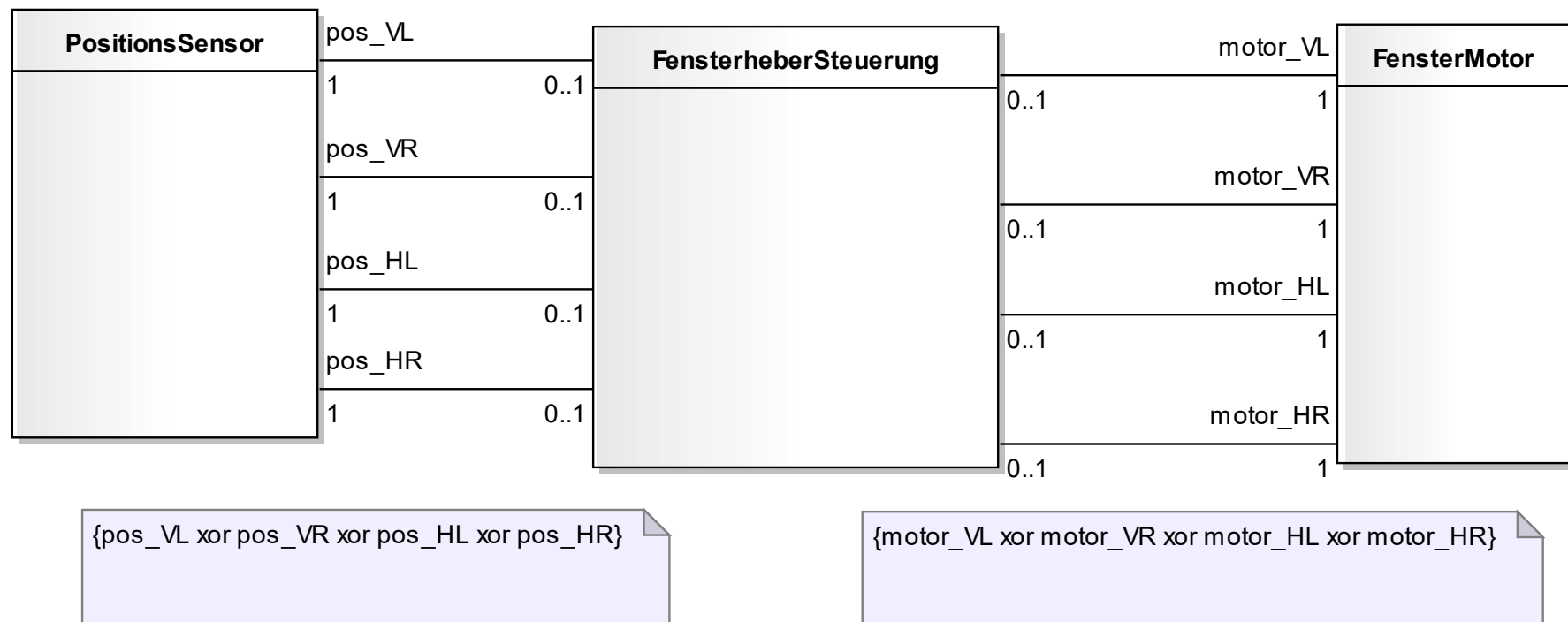
- Beispiel



- Beispiel: Fensterheber im Türsteuergerät (vgl. SWE4, S.25/40)



- Beispiel: Fensterheber im Türsteuergerät
  - Verwendung von Rollen zur Unterscheidung der verschiedenen Motoren/Taster/Sensoren



- **Aufgabe 5**

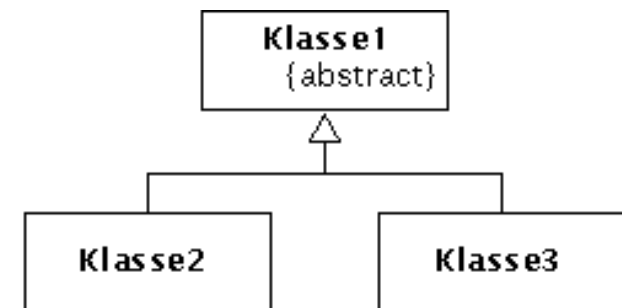
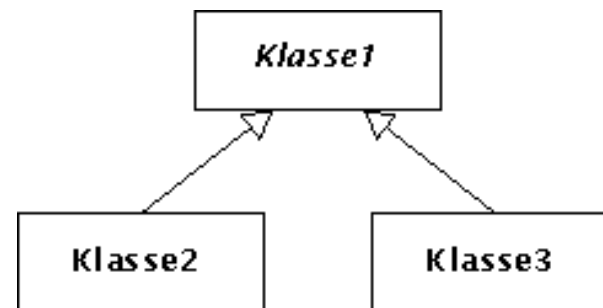
- Erstellen Sie ein Klassendiagramm, so dass die im folgenden beschriebenen Sachverhalte modelliert werden!
  - Eine Tagung (z.B. Softwaretechnik-Tagung in Hamburg) ist zu organisieren. Für jeden Teilnehmer der Tagung werden der Name, die Adresse und der Status (Student, Mitglied, Nichtmitglied) gespeichert.
  - Jeder Teilnehmer kann sich für ein oder mehrere halbtägige Tutorien, die zusätzlich zum normalen Tagungsprogramm angeboten werden, anmelden. Für jedes Tutorium werden dessen Nummer, die Bezeichnung sowie das Datum gespeichert. Alle Tutorien kosten gleich viel. Damit ein Tutorium stattfindet, müssen mindestens 10 Anmeldungen vorliegen. Jedes Tutorium wird von genau einem Referenten angeboten.



- **Aufgabe 5** (Forts.)
  - Für jeden Referenten werden dessen Name und Firma gespeichert. Ein Referent kann sich auch für ein oder mehrere Tutorien – anderer Referenten – anmelden und kann bei diesen kostenlos zuhören. Diese Anmeldungen zählen bei der Ermittlung der Mindestanmeldungen nicht mit. Ein Teilnehmer kann nicht gleichzeitig Referent sein. Ein Referent kann mehrere Tutorien anbieten. An einem Tutorium können mehrere Referenten kostenlos teilnehmen.
  - Ein Teilnehmer kann sich in der Tagungsanmeldung auch für einige Rahmenprogramme (z.B. Besuch eines Musicals) eintragen lassen. Für jedes Rahmenprogramm werden dessen Bezeichnung, das Datum, die Zeit, der Ort und die Kosten gespeichert.

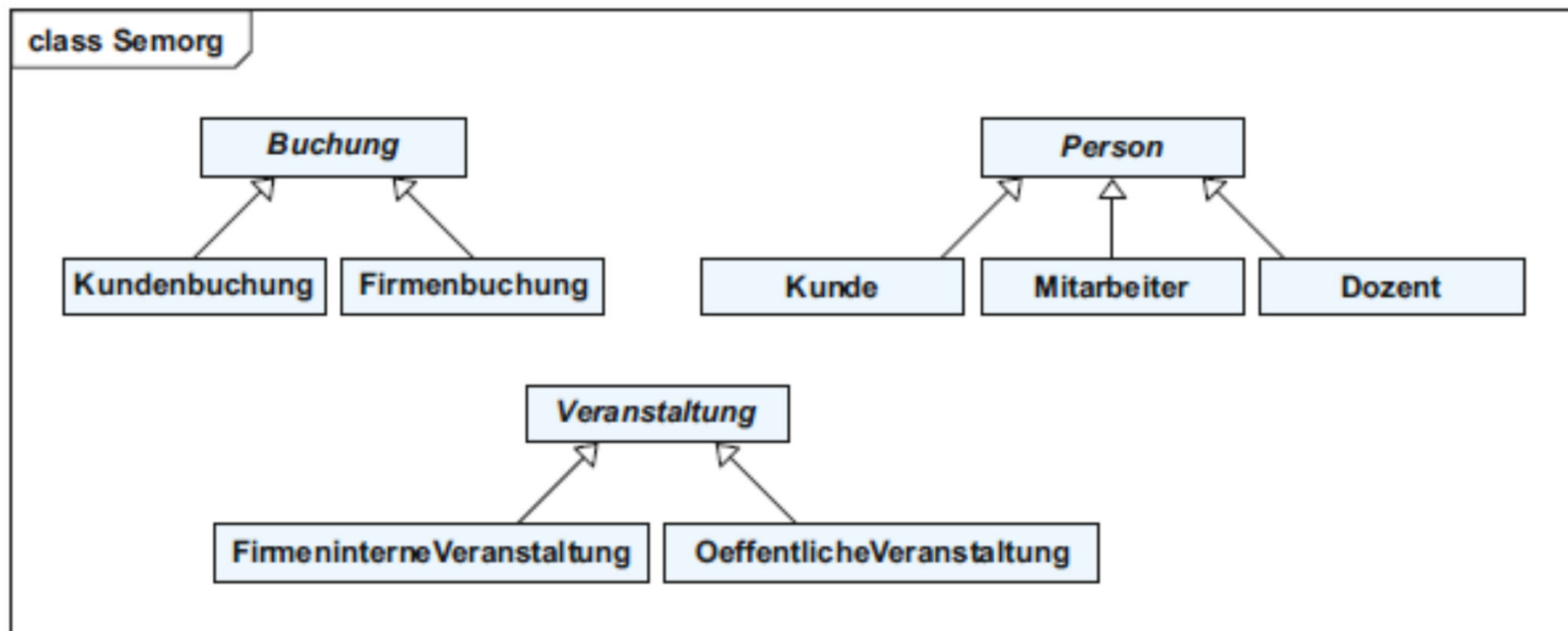
- Eine **Generalisierung** (*generalization*) ist eine Beziehung zwischen einer allgemeinen Klasse (**Basisklasse**) und einer spezialisierten Klasse
  - Allgemeine Klasse = **Oberklasse** (*super class*)
  - Spezialisierte Klasse = **Unterklasse** (*sub class*)
- Die spezialisierte Klasse ist vollständig konsistent mit der Basisklasse
  - Sie hat alle Merkmale der Basisklasse (→ **Vererbung**),
  - Sie enthält zusätzliche Merkmale

- Notation:



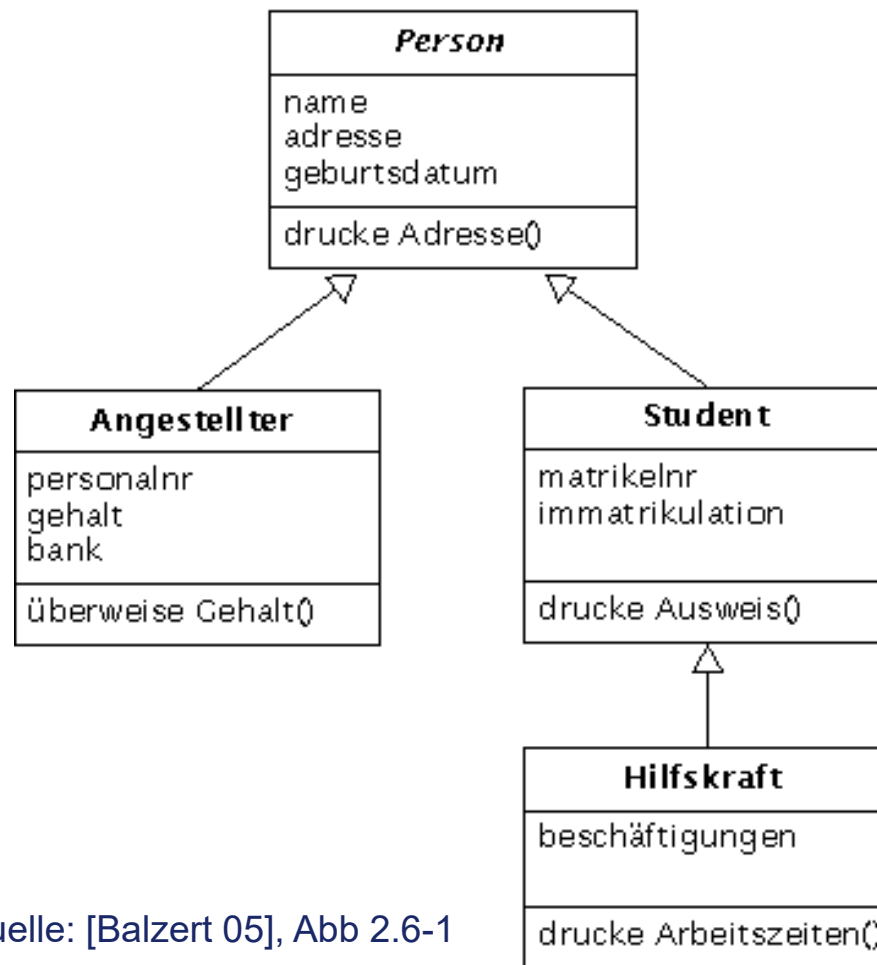
Quelle: [Balzert 05], Abb 2.6-2

- Beispiel ([Balzert 09], Abb. 9.3-10)
  - Seminarorganisation



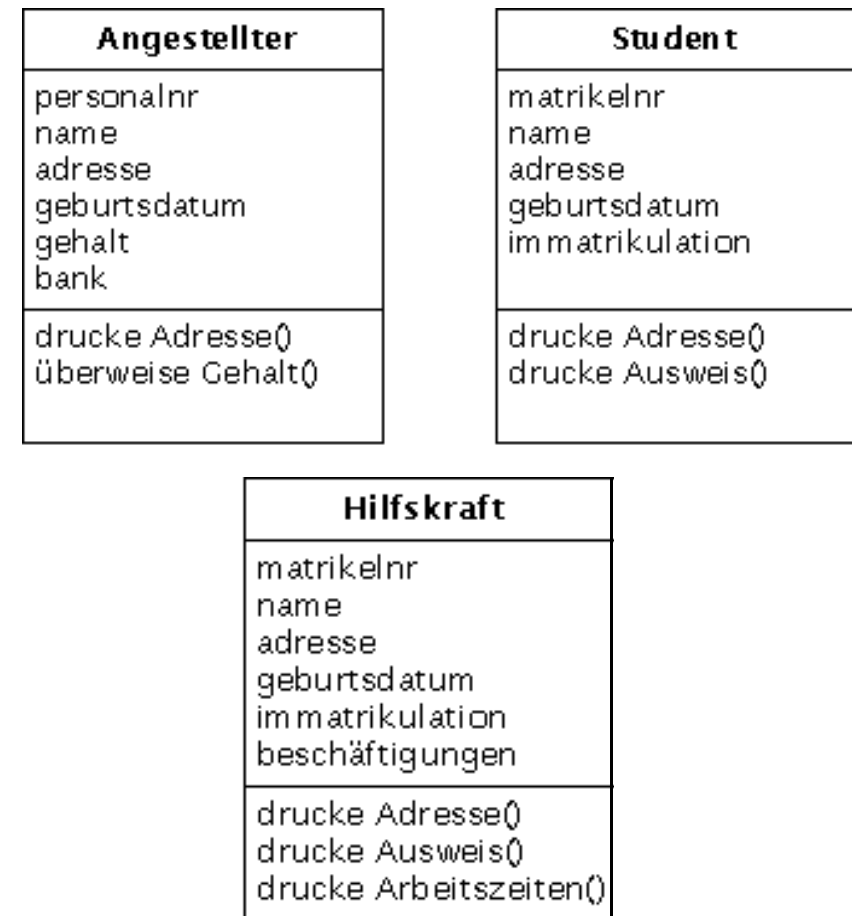
- Beispiel

*Mit Generalisierung*

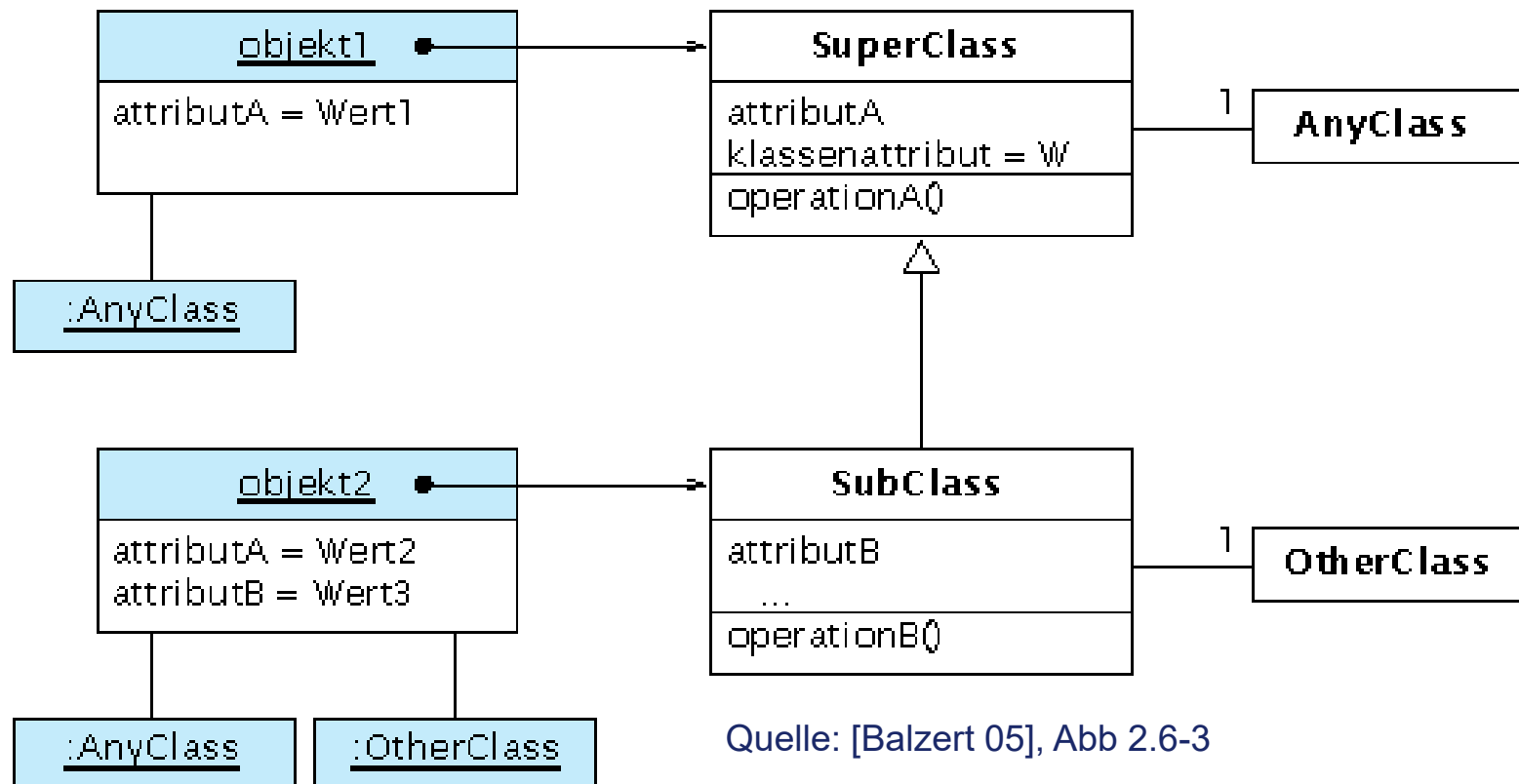


Quelle: [Balzert 05], Abb 2.6-1

*Ohne Generalisierung*



- Was wird vererbt?
  - Attribute
  - Operationen
  - Assoziationen

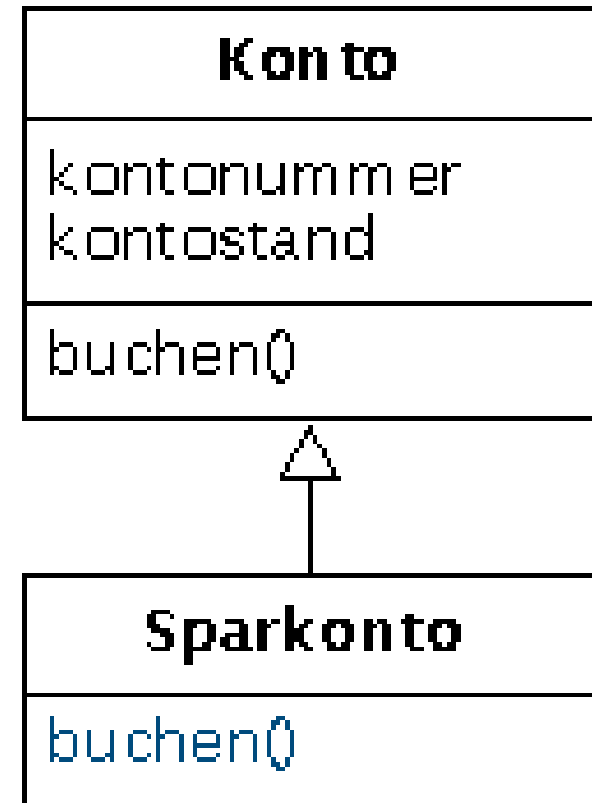


Quelle: [Balzert 05], Abb 2.6-3

- Generalisierung: „**is-a**“-Beziehung
  - „*Ein Objekt der Unterklasse **ist ein** Objekt der Oberklasse*“
  - Beispiel:
    - Ein Student **ist eine** Person, ein Angestellter **ist eine** Person
    - Eine Firmenbuchung **ist eine** Buchung, eine Kundenbuchung **ist eine** Buchung
- Spezialisierung: „**oder**“-Beziehung
  - „*Ein Objekt der Oberklasse ist ein Objekt der Unterklasse 1 **oder** der Unterklasse 2 **oder**...*“
  - Beispiel:
    - Eine Person ist ein Student **oder** ein Angestellter
    - Eine Buchung ist eine Kundenbuchung **oder** eine Firmenbuchung
- Beachte:
  - Vererbung alleine begründet keine Generalisierung/Spezialisierung!

## Überschreiben von Operationen

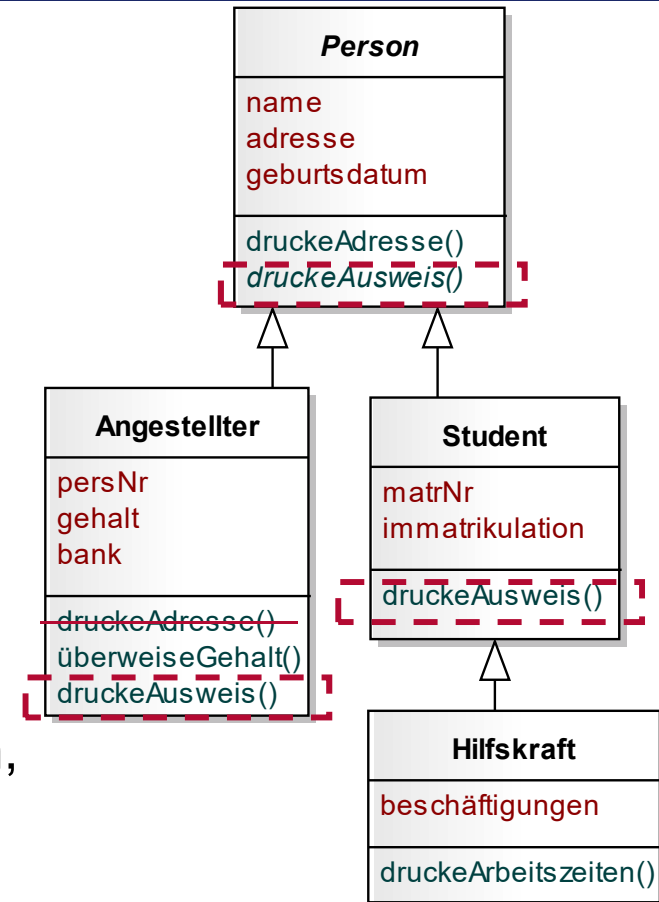
- Unterklassen können das Verhalten ihrer Oberklassen verfeinern, redefinieren bzw. **überschreiben** (*redefine, override*)
- Beispiel:
  - Die Operation *buchen()* ist sowohl auf allgemeine Konten als auch auf die speziellen Sparkonten anwendbar
  - Für ein Sparkonto wird *buchen()* aber anders realisiert als für andere Konten.



Quelle: [Balzert 05], Abb 2.6-4

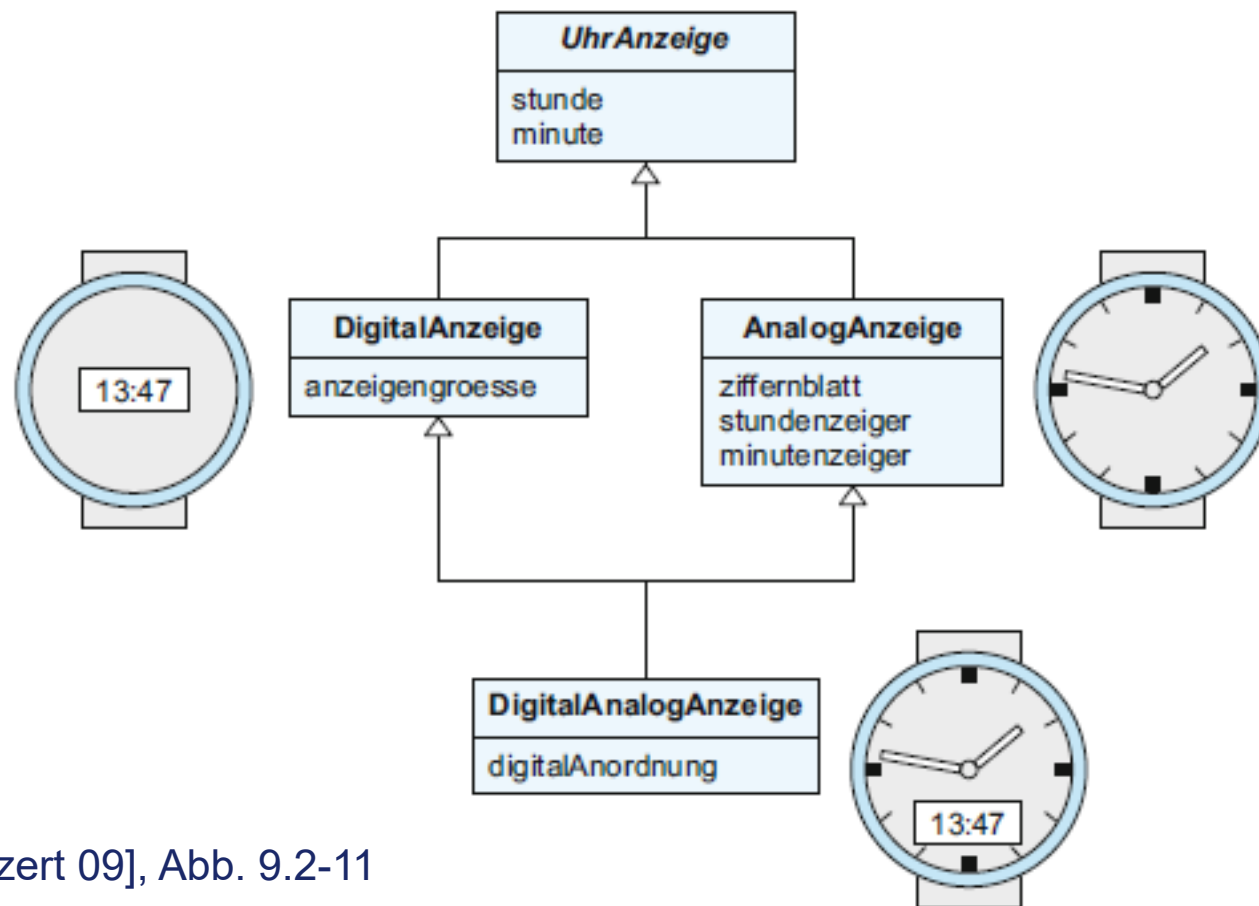
- **Abstrakte Klassen**

- haben keine Objekte (Instanzen)  
→ Klasse *Person*
- fassen Eigenschaften zusammen, die an konkrete Unterklassen vererbt werden
- können konkrete Operationen definieren, die ausschließlich Eigenschaften der abstrakten Klasse ausnützen  
→ *druckeAdresse()*
- Können **abstrakte Operationen** definieren, die in allen konkreten Unterklassen überschrieben werden müssen
  - Beispiel:
    - die Klasse *Person* erhält eine abstrakte Operation *druckeAusweis()*
    - sie wird in den Klassen *Student* und *Angestellter* konkret definiert.
- **Notation:** abstrakte Klassen und abstrakte Operationen werden *kursiv* geschrieben.



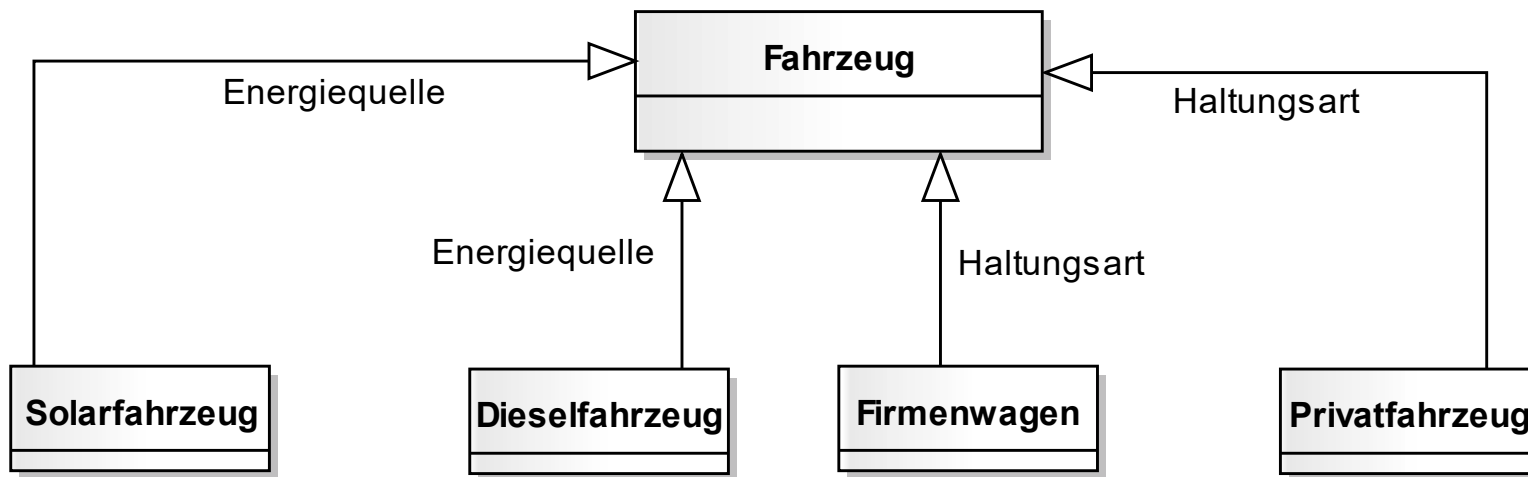


- Mehrfach-Generalisierung:
  - ein Klasse kann Spezialisierung von mehreren Basisklassen sein und von mehreren Basisklassen Merkmale erben



Quelle: [Balzert 09], Abb. 9.2-11

- **Generalisierungsmenge (*generalization set*)**
  - gibt an, nach welchem Kriterium die eine Generalisierungsstruktur gebildet wird
  - Für eine Klasse können mehrere unterschiedliche Generalisierungsmengen definiert werden, die zu unterschiedlichen Spezialisierungen führen
    - Notation: Generalisierungsmenge an die jeweilige Generalisierungsbeziehung schreiben



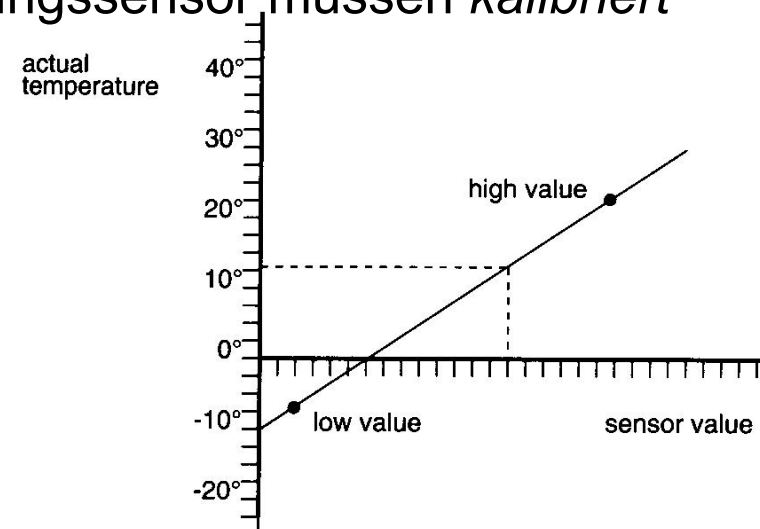
- Eigenschaften von Generalisierungsmengen
  - **complete:**
    - Jedes Objekt der Oberklasse gehört in mindestens Unterklasse
      - Beispiel: Generalisierungsmenge *Haltungsart*
  - **incomplete:**
    - Es gibt Objekte der Oberklasse, die in keine der Unterklassen gehören
      - Beispiel: Generalisierungsmenge *Energiequelle*
  - **disjoint:**
    - Ein Objekt der Oberklasse gehört in höchstens eine der Unterklassen
      - Die Spezialisierung ist eine „entweder-oder“-Beziehung
      - Beispiel: Generalisierungsmenge *Energiequelle* und *Haltungsart*
  - **overlapping:**
    - Es kann Objekte der Oberklasse geben, die in mehreren Unterklassen gehören
      - Die Spezialisierung ist keine „entweder-oder“-Beziehung (nur „oder“)
      - Beispiel: Unterscheide die *Angestellten* eines Softwarehauses nach Ihrer Tätigkeit: *Programmierer*, *Analytiker*, *Manager*.

## • Aufgabe 6

- Für eine Wetterstation werden verschiedene Sensoren benötigt:
  - Mit einem *Windrichtungssensor* wird die aktuelle Windrichtung erfasst.
  - Mit einem *Temperatursensor* wird die aktuelle Temperatur erfasst.
  - Mit einem *Luftdrucksensor* wird der aktuelle Luftdruck erfasst.
  - Mit einem *Luftfeuchtigkeitssensor* wird die aktuelle Luftfeuchtigkeit erfasst
  - Mit einem *Windgeschwindigkeitssensor* wird die aktuelle Windgeschwindigkeit erfasst.

- Alle Sensoren außer dem Windrichtungssensor müssen *kalibriert* werden, d.h.:

- Die Hardware der Sensoren liefert jeweils eine Float-Zahl als Messwert.
- Für zwei Werte (*HighValue*, *LowValue*) muss bekannt sein, welche echten Messwerte (z.B. eine Temperatur) abgebildet werden.
- Die anderen Werte werden durch lineare Interpolation berechnet.
- Kalibrierung bedeutet Festlegen der Werte *HighValue* und *LowValue*

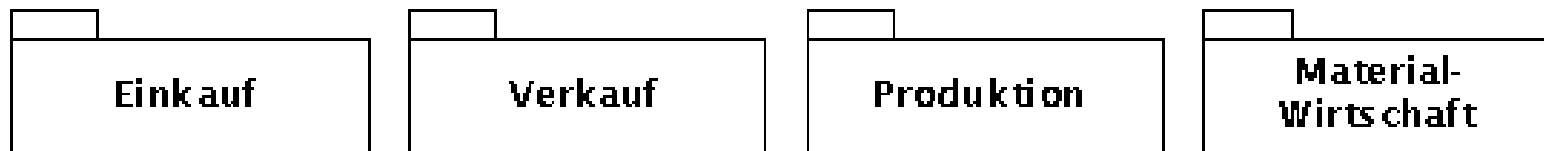


- **Aufgabe 6 (Forts.)**
  - Alle Sensoren außer dem Windrichtungssensor können jeweils Auskunft geben über den *höchsten und niedrigsten gemessenen Wert eines Tages*.
  - Für Temperatur und Luftdruck soll das System jederzeit einen *Trend* ausgeben können.
    - Der Trend wird angegeben als Zahl zwischen -1 und +1
      - 0: keine Veränderung der Werte zu erwarten
      - <0: Verringerung der Werte zu erwarten
      - >0: Steigerung der Werte zu erwarten
  - Modellieren Sie die verschiedenen Sensoren und die Operationen, die für diese Sensoren benötigt werden, als Klassendiagramm!

Aufgabenstellung und Abbildung aus: Grady Booch, *Object Oriented Analysis and Design with Applications*, The Benjamin/Cummings Publishing Company Inc., 1994 Chapter 8.

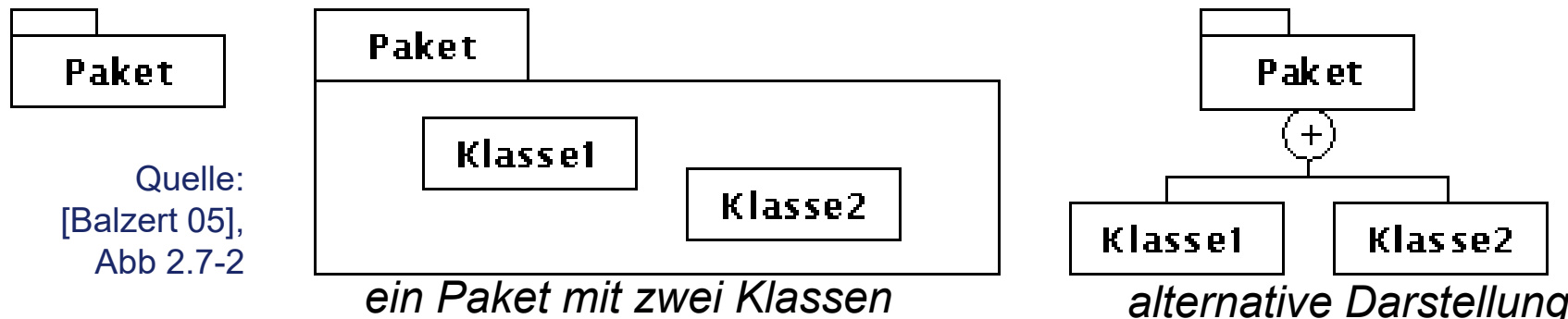
- Aufgabe 7
  - Erstellen Sie ein Klassendiagramm, welches folgende Sachverhalte modelliert:
    - Wir betrachten eine Bank und ihre Kunden. Eine Person wird Kunde, wenn sie ein Konto eröffnet. Ein Kunde kann beliebig viele weitere Konten eröffnen. Für jeden neuen Kunden werden dessen (nicht notwendigerweise eindeutiger) Name, Adresse und das Datum der ersten Kontoeröffnung erfaßt. Bei der Kontoeröffnung muß der Kunde gleich eine erste Einzahlung vornehmen. Wir unterscheiden Girokonten und Sparkonten. Girokonten dürfen bis zu einem bestimmten Betrag überzogen werden. Für jedes Konto wird ein individueller Habenzins, für Girokonten auch ein individueller Sollzins festgelegt.
    - Außerdem besitzt jedes Konto eine eindeutige Kontonummer. Für jedes Sparkonto wird die Art des Sparens – z.B. Festgeld – gespeichert. Ein Kunde kann Beträge einzahlen und abheben. Des Weiteren werden Zinsen gutgeschrieben und bei Girokonten Überziehungszinsen abgebucht. Um die Zinsen zu berechnen, muß für jede Kontobewegung das Datum und der Betrag notiert werden. Die Gutschrift/Abbuchung der Zinsen erfolgt bei den Sparkonten jährlich und bei den Girokonten quartalsweise. Ein Kunde kann jedes seiner Konten wieder auflösen. Bei der Auflösung des letzten Kontos hört er auf, Kunde zu sein.

- Ein Paket (*package*) ...
  - fasst Modellelemente (z.B. Klassen) zusammen
  - kann selbst Pakete enthalten
  - Beispiel: Warenwirtschaftssystem



Quelle: [Balzert 05], Abb 2.7-1

- Notation
  - Das Paket wird als Rechteck mit einem Reiter dargestellt
  - Der Name kann entweder im Reiter oder im Rechteck stehen
  - Der Inhalt des Pakets kann im Rechteck dargestellt werden



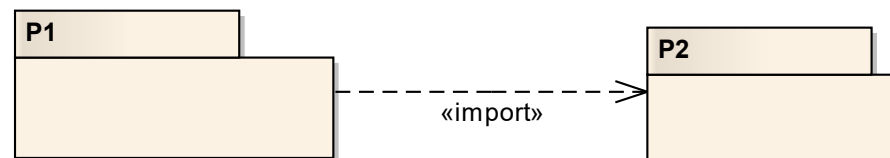
Quelle:  
[Balzert 05],  
Abb 2.7-2

- Namensraum
  - Ein Paket definiert einen Namensraum für alle enthaltenen Elemente
  - Die Elemente haben eine Sichtbarkeit innerhalb des Pakets
    - *public* (+): Elemente können auch in anderen Paketen sichtbar sein
    - *private* (-): Elemente sind nur im eigenen Paket sichtbar
  - "Qualifizierter Name" einer Klasse aus einem Paket:

`Paket::Klasse`

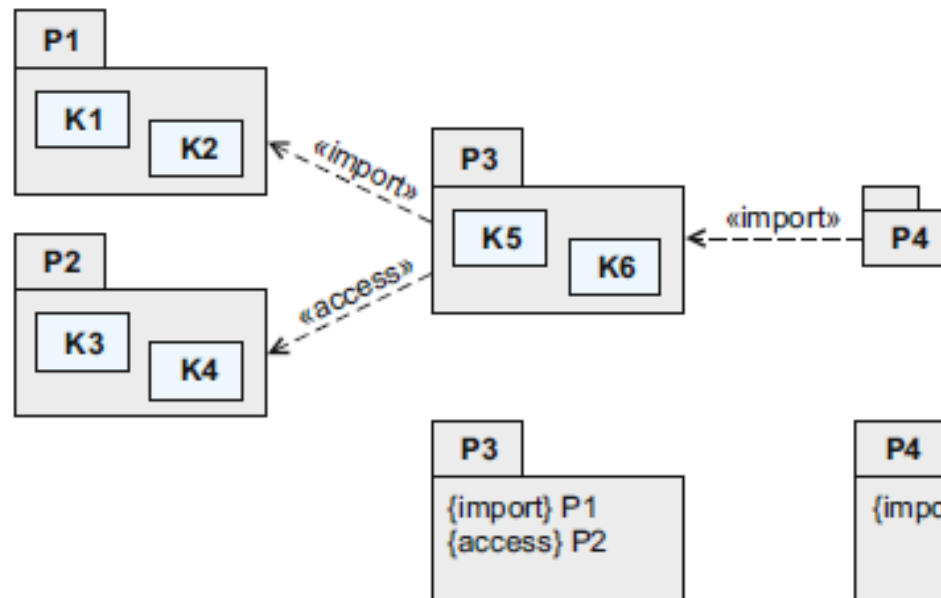
`Paket1::Paket11::Paket111::Klasse`

- Import
  - Erlaubt dem *importierenden* Paket das unqualifizierte Verwenden aller öffentlichen Namen aus dem *importierten* Paket





- Öffentlicher Import (`<<import>>`-Abhängigkeit)
  - Importierte Elemente sind im importierenden Paket wieder öffentlich  
→ sie können von weiteren Paketen importiert werden
- Privater Import (`<<access>>`-Abhängigkeit)
  - Importierte Elemente sind im importierenden Paket privat  
→ sie können nicht von weiteren Paketen importiert werden



Öffentlicher Import (`<<import>>`):

- in P3 darf K1 und K2 benutzt werden
- in P4 darf K1, K2, K5, K6 benutzt werden

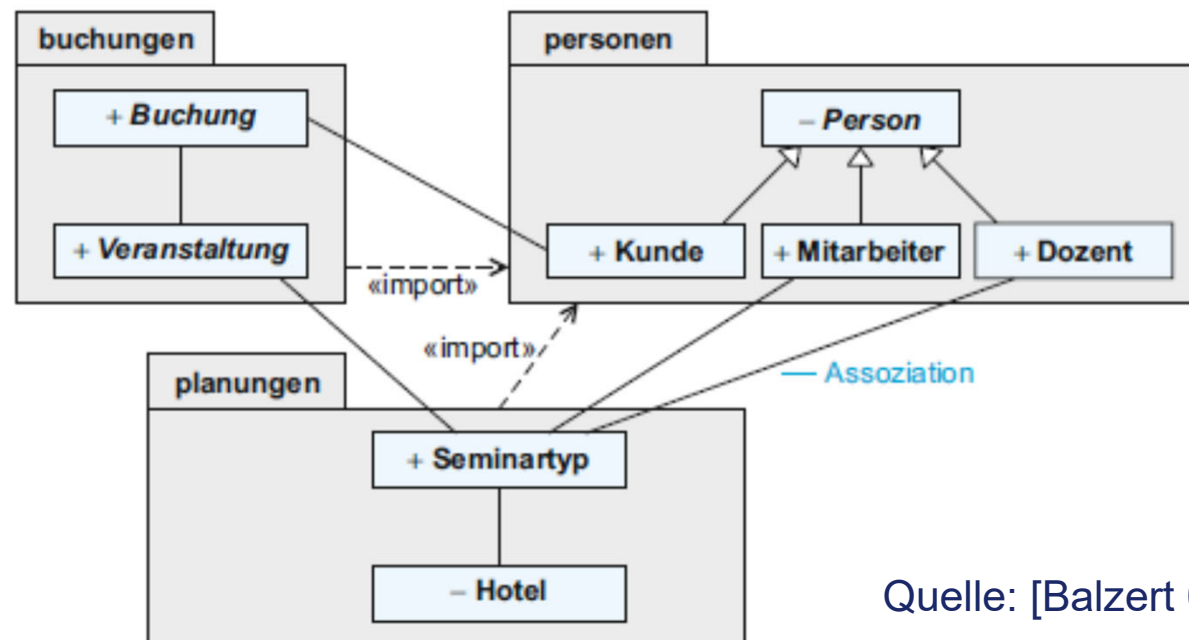
Privater Import (`<<access>>`)

- in P3 darf K3, K4 benutzt werden
- in P4 darf K3 und K4 **nicht** benutzt werden

Alternative Darstellung von Import-Beziehungen

Quelle: [Balzert 09], Abb. 9.2-6

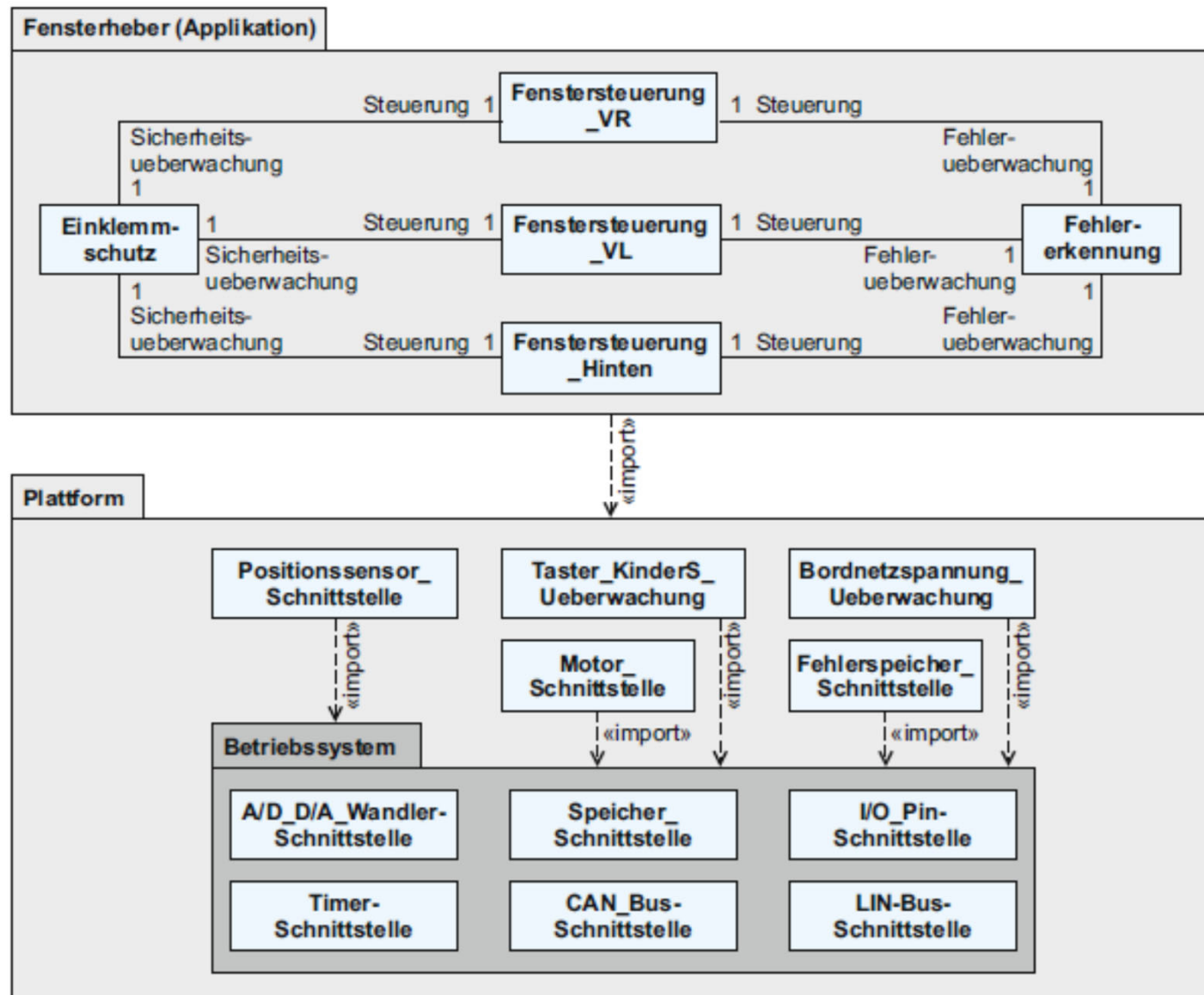
- Anwendung
  - Analyse: Fachliche Strukturierung eines großen Systems in Teilsysteme
    - Beispiel: Pakete für die Seminarorganisation



Quelle: [Balzert 09], Abb. 9.2-7

- Entwurf: Paket-Bildung für Architektur-Schichten
  - Benutzungsoberfläche – Fachkonzept – Kommunikationsschicht - Datenhaltung

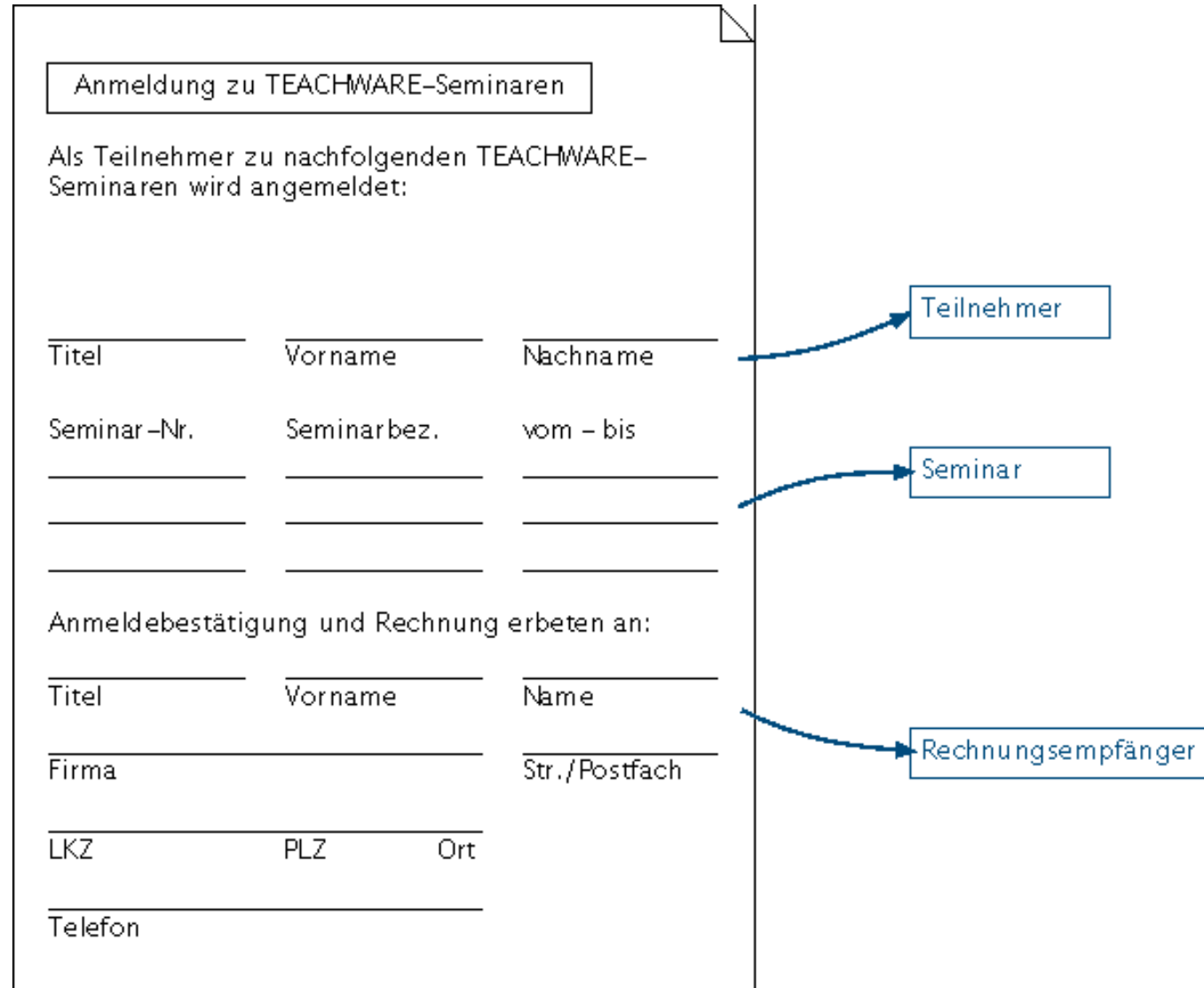
- Beispiel



Quelle: [Balzert 09], Abb. 27.0-2

- Dokumentanalyse
  - Aus Formularen und Listen Attribute entnehmen und zu Klassen zusammen fassen (→ „**bottom-up**“-Vorgehen)
  - Bei Re-Engineering-Systemen:
    - Benutzerhandbücher
    - Bildschirmmasken
    - Dateibeschreibungen
    - Funktionalität im laufenden System
  - Dokumentanalyse dient auch zum Identifizieren von Assoziationen
    - gegebenenfalls gleich mit darstellen!

- Beispiel:  
Seminar-  
organisation



Anmeldung zu TEACHWARE-Seminaren

Als Teilnehmer zu nachfolgenden TEACHWARE-Seminaren wird angemeldet:

Titel	Vorname	Nachname
Seminar-Nr.	Seminarbez.	vom - bis

Anmeldebestätigung und Rechnung erbeten an:

Titel	Vorname	Name
Firma		Str./Postfach
LKZ	PLZ	Ort
Telefon		

Teilnehmer

Seminar

Rechnungsempfänger

Quelle: [Balzert 05], Abb. 4.4-1

- Beschreibung der Anwendungsfälle
  - Durchsuchen des Textes nach Klassen (→ „**top-down**“-Vorgehen)
    - Oft sind die **Substantive** potentielle Klassen
    - Potentielle Klasse auf Attribute überprüfen
      - Klassen ohne Attribute sind überflüssig
    - Eine Klasse kann sich auch hinter Verben verbergen
      - Wenn z.B. über Vorgänge Daten gespeichert werden, z.B.:
        - » buchen (→ Buchung)
        - » reservieren (→ Reservierung)
    - Akteure, über die man sich etwas „merken“ muss
      - In einem „Web-Shop“ sind z.B. Kunden Akteure
  - Nicht benötigte Klassen streichen
    - Ausschlussgründe
      - Klassen ohne Attribute
      - Begriffe, die Attribute anderer Klassen beschreiben
      - Begriffe, die Synonyme sind für bereits identifizierte Klassen
      - Container (Objektverwaltung)
        - » In der Analyse Klassenoperationen verwenden!

- Beispiel: Seminarorganisation

**Anwendungsfall: anmelden eines neuen Teilnehmers**

Interessenten melden sich schriftlich an

Der betreffende Interessent und die gebuchten Seminare werden in die Kundenkartei aufgenommen

Ein Seminar wird durch einen Seminartyp beschrieben. Seminartypen und Seminare sind in der Seminarkartei gespeichert

Die Seminargebühr ist für alle Seminare eines Typs gleich

Einige Kunden erhalten Ermäßigungen.

Jede Anmeldung wird von dem Teachware-Mitarbeiter Mitarbeiter schriftlich bestätigt



**Klassen**



**keine Klassen**



**fragliche Klassen**

- Beispiel: Seminarorganisation

**Anwendungsfall: absagen eines gebuchten Seminars**

Gebuchte Seminarveranstaltungen können durch Kunden zu folgenden Bedingungen abgesagt werden

Bei einer Absage bis zu drei Wochen vor Seminarbeginn entstehen keine Kosten

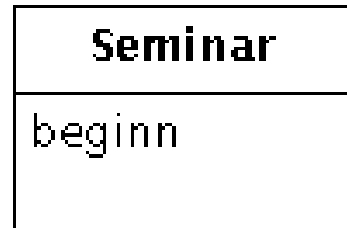
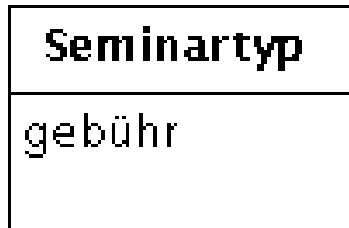
Bei einer späteren Absage werden 50 Prozent der Seminargebühr berechnet

Jede Absage wird von der Firma Teachware schriftlich bestätigt

Wenn der Kunde nicht rechtzeitig absagt, erhält er nach der Veranstaltung eine Rechnung, obwohl er nicht teilnimmt



- Beispiel: Seminarorganisation
  - Identifizierte Klassen



Quelle: [Balzert 05], Abb. 4.4-2

- Nicht benötigte Klassen
  - *Teachware-Mitarbeiter* (keine Attribute)
  - *Seminar-Gebühr* (Attribut der Klasse Seminartyp)
  - *Beginn* (Attribut der Klasse Seminar)
  - *Interessent* (Synonym für Kunde)
  - *Seminarkartei / Kundenkartei* (Objektverwaltung)
- Noch fraglich:
  - *Anmeldung* (werden Daten zu dem Vorgang gespeichert?)
  - *Rechnung*
    - eigentlich nur Druckausgabe von Daten aus anderen Klassen
    - Gibt es spezifische Daten, die gespeichert werden müssen?

- Kategorien
  - Konkrete Objekte bzw. Dinge - z.B. *Pkw, Buch, ...*
    - Beispiel: „Türme von Hanoi“
  - Personen und deren Rollen - z.B. *Kunde, Referent, Mitarbeiter, ...*
  - Informationen über Aktionen - z.B. *Buchung, Anmeldung, ...*
  - Orte - z.B. *Hörsaal, Standort (eines Buches) ...*
  - Organisationen - z.B. *Bankfiliale*
  - Behälter - z.B. *Lagerplatz*
    - Vorsicht: Keine Objektverwaltung modellieren!
  - Dinge in einem Behälter - z.B. *Paletten*
  - Ereignisse (*über die Informationen gespeichert werden*)
  - Kataloge - z.B.: *Produktkatalog*
  - Verträge - z.B.: *Kaufvertrag*

- Dokumentanalyse
  - Einfache Attribute zu Datenstrukturen zusammenfassen
  - Prüfen, ob alle Attribute wirklich notwendig sind
    - Nimmt ein Attribut jemals einen Wert an?
    - Ist dieser Wert an der Benutzungsoberfläche sichtbar?
- Beschreibung Anwendungsfälle
  - Benötigte Daten zur Ausführung der Aufgaben eines Anwendungsfalls
    - Türme von Hanoi: Größe einer Scheibe, Anzahl der Scheiben
- Geeignete Attributtypen wählen!
  - Vorgegebene Typen nur, falls problemadäquat
  - Gegebenfalls eigene Typen definieren
    - z.B. als *Primitive*, *dataType* oder *enumeration*
      - Türme von Hanoi: `StapelId`

- Klassenname
  - Fachterminologie
  - Substantiv im Singular
  - So konkret wie möglich
  - Soll nicht die Rolle dieser Klasse in einer Beziehung zu einer anderen Klasse beschreiben
    - ...es sei denn, die Objekte treten nur in einer einzigen Rolle auf
      - Beispiel: „Kunde“ statt „Käufer“  
„Mitarbeiter“ statt „Projektleiter“, „Abteilungsleiter“
  - Soll eindeutig im Paket bzw. im System sein
  - Darf nicht dasselbe ausdrücken wie der Name einer anderen Klasse

- Abstraktionsniveau
  - Das eine Extrem
    - Das System wird durch eine einzige Klasse modelliert, die alle Attribute und alle Operationen enthält
      - Keine Strukturierung
  - Das andere Extrem
    - Bildung sehr vieler Klassen
    - Attribute sind dann oft vom elementaren Typ
    - Bei vielen „Klassen“ handelt es sich um Attribute anderer Klassen
      - unübersichtliches Modell
- Fehlerquellen
  - Klasse modelliert eine reine Objektverwaltung
    - Gegebenenfalls Klassenoperationen verwenden
  - Klasse modelliert Benutzungsoberfläche
  - Klasse modelliert Entwurfs- oder Implementierungsdetails

- Attributnamen
  - Kurz, eindeutig und verständlich im Kontext der Klasse
  - Substantiv oder substantivierte Adjektive (z.B. „Größe“)
  - Namen der Klasse nicht wiederholen (z.B. „**Scheibengröße**“)
    - Ausnahme: feststehende Begriffe (z.B. „**Personalnummer**“)
  - Nur fachspezifische oder allgemeine übliche Abkürzungen verwenden (z.B. Nr, ID, PLZ)
- Abstraktionsniveau für Attribute
  - Anzahl Attribute pro Klasse soll angemessen sein
  - Falls erforderlich, komplexe Attribute mit geeigneten Datentypen bilden
- Abgeleitete Attribute nur verwenden wenn...
  - Information für den Benutzer sichtbar ist
  - Lesbarkeit wird verbessert wird

Seminarveranstaltung
beginn ende /dauer

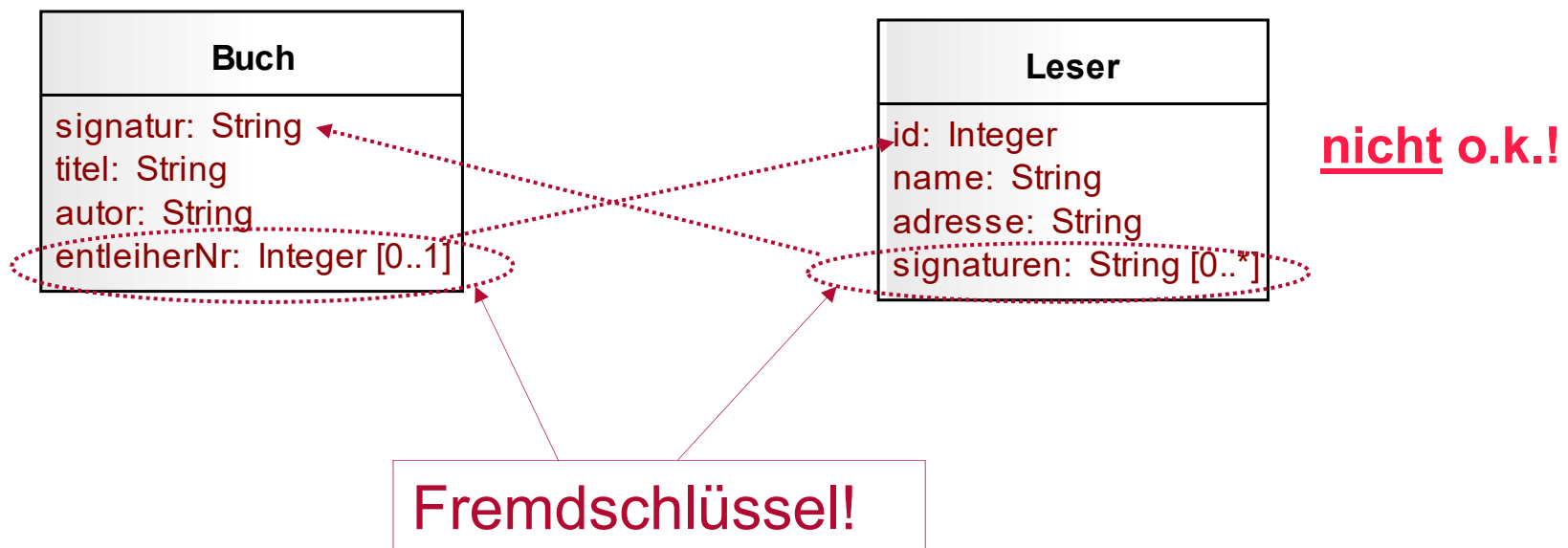
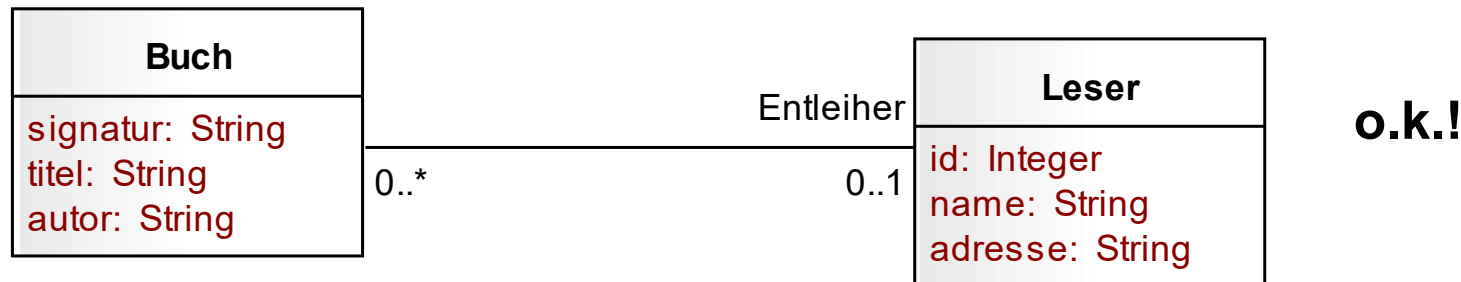
Quelle: [Balzert 05], Abb. 4.6-3

- Klassenattribute
  - Ein Wert für alle Objekte der Klasse
  - Informationen über die Gesamtheit aller Objekte
    - Türme von Hanoi: „Scheibenzahl“ als Klassenattribut der Klasse *Scheibe* möglich, aber fachlich auch der Klasse *Spiel* zuzuordnen
- Schlüsselattribute fachlich notwendig?
  - Schlüsselattribut identifiziert jedes Objekt innerhalb einer Klasse eindeutig (→ Eigenschaftswert {**key**})
  - Schlüsselattribute werden in der Analyse nur dann modelliert, wenn sie Bestandteil des Fachkonzepts sind.
    - Beispiel: Matrikelnummer, Bestellnummer
  - Also **keine künstlichen Schlüsselattribute** hinzufügen!

- Attribute, die in der Analyse **nicht** eingetragen werden
  - Interne Zustandsattribute
    - Türme von Hanoi: die Höhe eines Stapels
      - aber**: der Attributwert ist an der Benutzeroberfläche sichtbar  
(*eigentlich abgeleitet*)  
→ man kann ihn ausnahmsweise eintragen!
  - Attribute, die Entwurfs- und Implementierungsdetails beschreiben
  - Aus *Performance*-Gründen abgeleitetes Attribut
- Fehlerquellen
  - Zu viele Attribute mit einfachem Typ definieren, statt strukturierte Datentypen zu bilden
  - Attribute statt Assoziationen
    - falsche Verwendung von Fremdschlüsseln



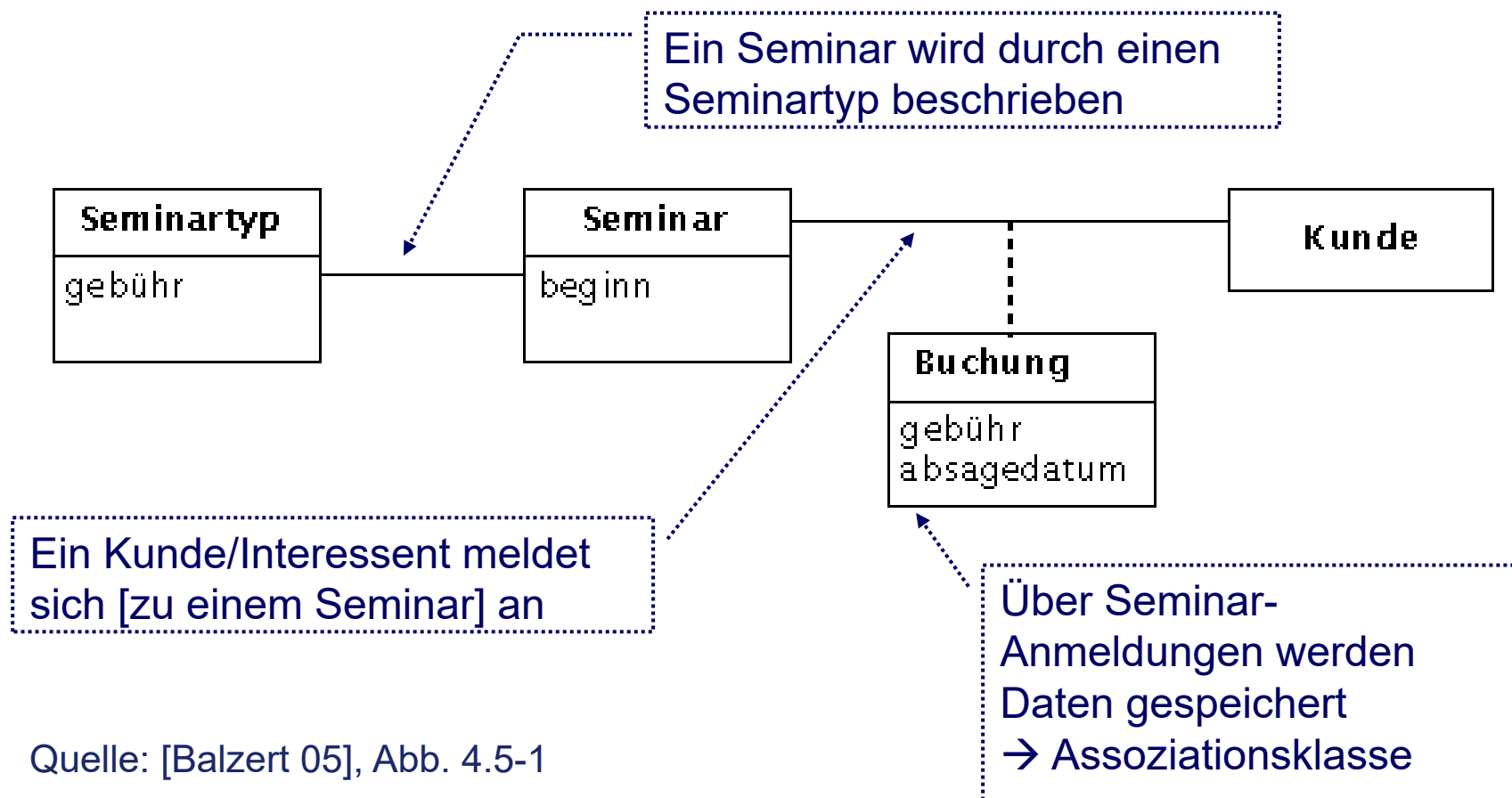
- Beispiel: Attribut statt Assoziation?



- Grundsätzliches Vorgehen
  - **Zuerst:** Identifizieren der puren Verbindung (nur Linie eintragen)
  - **Dann:** Ermittlung der zugehörigen Multiplizitäten
  - **Zuletzt:** Entscheidung Assoziation/Aggregation/Komposition
- Dokumentanalyse
  - Traditionelle Datenverarbeitung
    - Keine Objekte mit impliziter Identität
    - Identifizierung der »Objekte« durch Nummern
    - Diese Nummern finden sich als **Primär- und Fremdschlüssel** in den Dokumenten
  - Beispiel: Seminarverwaltung (vgl. [S. 77](#))
    - *Seminarnummer* ist Primärschlüssel eines Seminars
      - sie kann sich als Fremdschlüssel auf diversen anderen Dokumenten wieder finden:
        - » Anmeldeformular
        - » Rechnung

- Beschreibung Anwendungsfälle
  - Nach **Verben** suchen, die Folgendes ausdrücken:
    - Räumliche Nähe („...steht neben...“, „...befindet sich an...“)
    - Aktionen („...fährt nach...“, „...meldet sich an zu...“)
    - Kommunikation („...redet mit...“, „...teilt mit...“)
    - Besitz („...hat...“, „...gehört zu...“)
  - Beispiele:
    - Anwendungsfälle der Seminarorganisation
      - Siehe nächste Seite!
    - Türme von Hanoi:
      - Scheiben *befinden sich* auf Stapeln
      - Stapel *gehören zum* Spiel

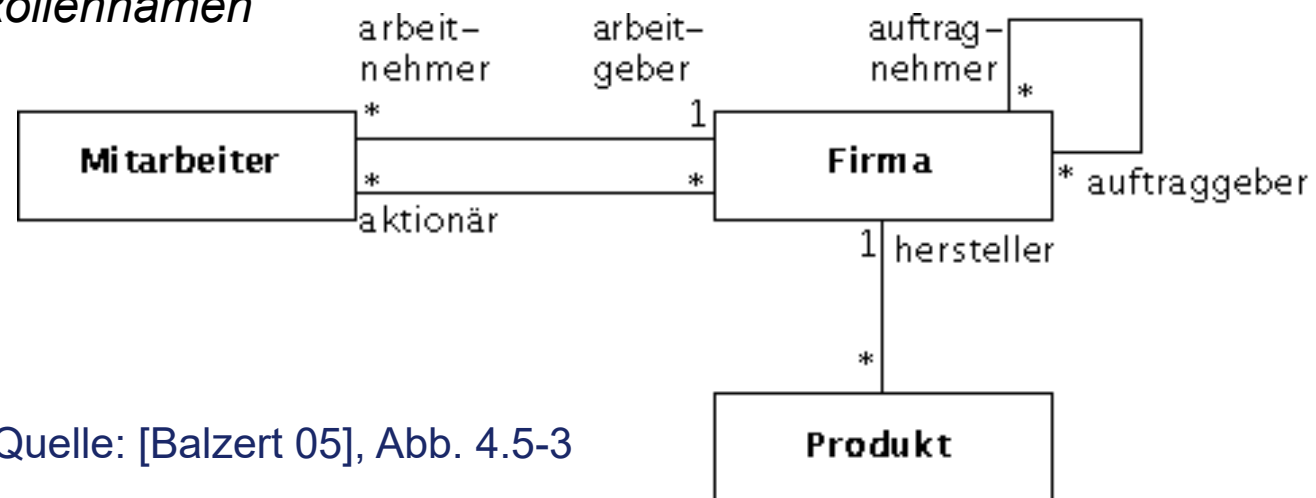
- Beispiel: Seminarorganisation
  - Assoziationen aus der Beschreibung von Anwendungsfällen (vgl. S. [79/80](#))



Quelle: [Balzert 05], Abb. 4.5-1

- Rollen- oder Assoziationsname?
  - Namen sind notwendig, wenn zwischen zwei Klassen mehrere Assoziationen bestehen
  - Rollennamen vs. Assoziationsnamen
    - „Klasse1 Assoziationsname Klasse2“ ergibt einen sinnvollen Satz
    - Welche Rolle spielt die Klasse X gegenüber einer Klasse Y?
    - Rollennamen häufig aussagekräftiger als Assoziationsnamen
    - Rollennamen sind bei reflexiven Assoziationen immer notwendig
    - Rollennamen sind Substantive, Assoziationsnamen enthalten Verben

Beispiel: Rollennamen

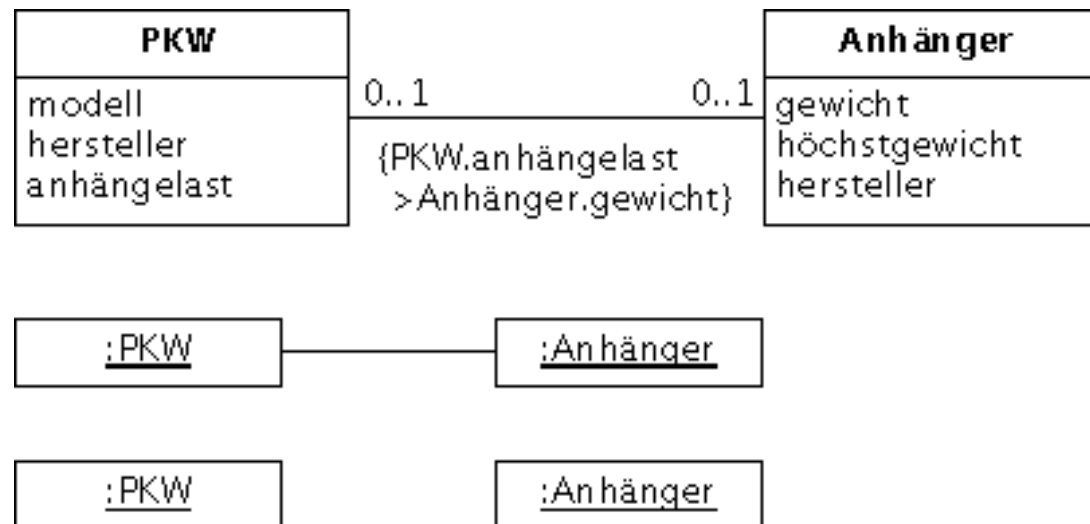


Quelle: [Balzert 05], Abb. 4.5-3

- 1:1-Assoziation
  - Können die beiden Klassen zu einer zusammengefasst werden?
  - Zusammenfassung nicht sinnvoll, falls...
    - ... Verbindungen sich ändern können
    - ... es sich um umfangreiche Klassen handelt
    - ... die beiden Klassen unterschiedliche Bedeutung haben

- Beispiel1:

- Zuordnungen können sich ändern
- Zusammenfassung ist nicht möglich

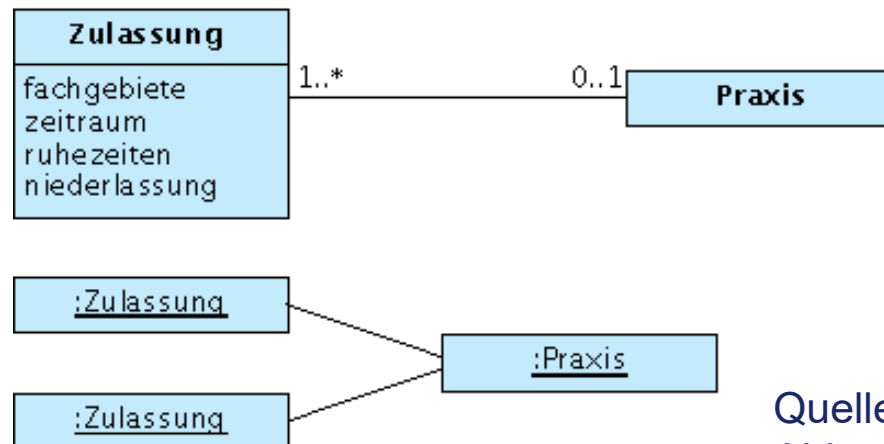
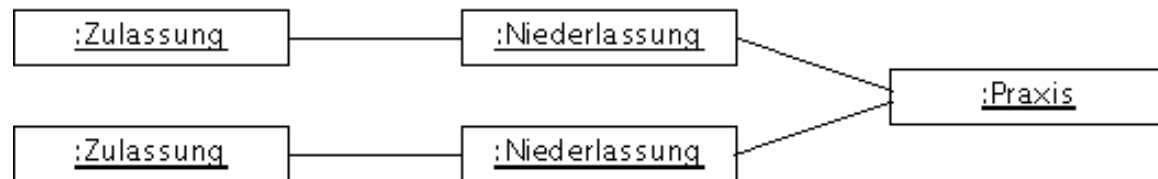
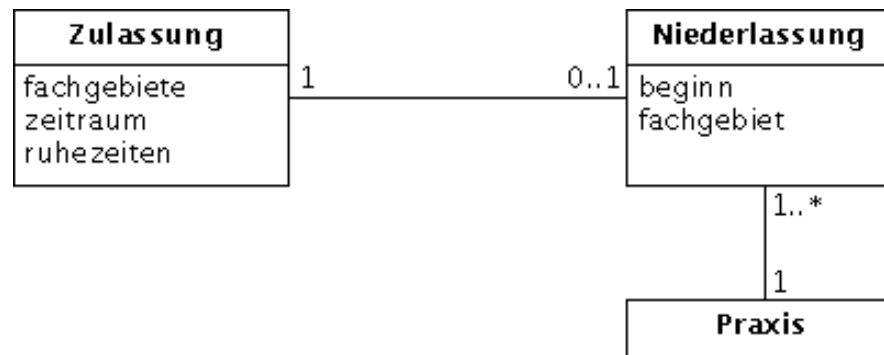


Quelle: [Balzert 05], Abb. 4.5-5

- 1:1-Assoziation

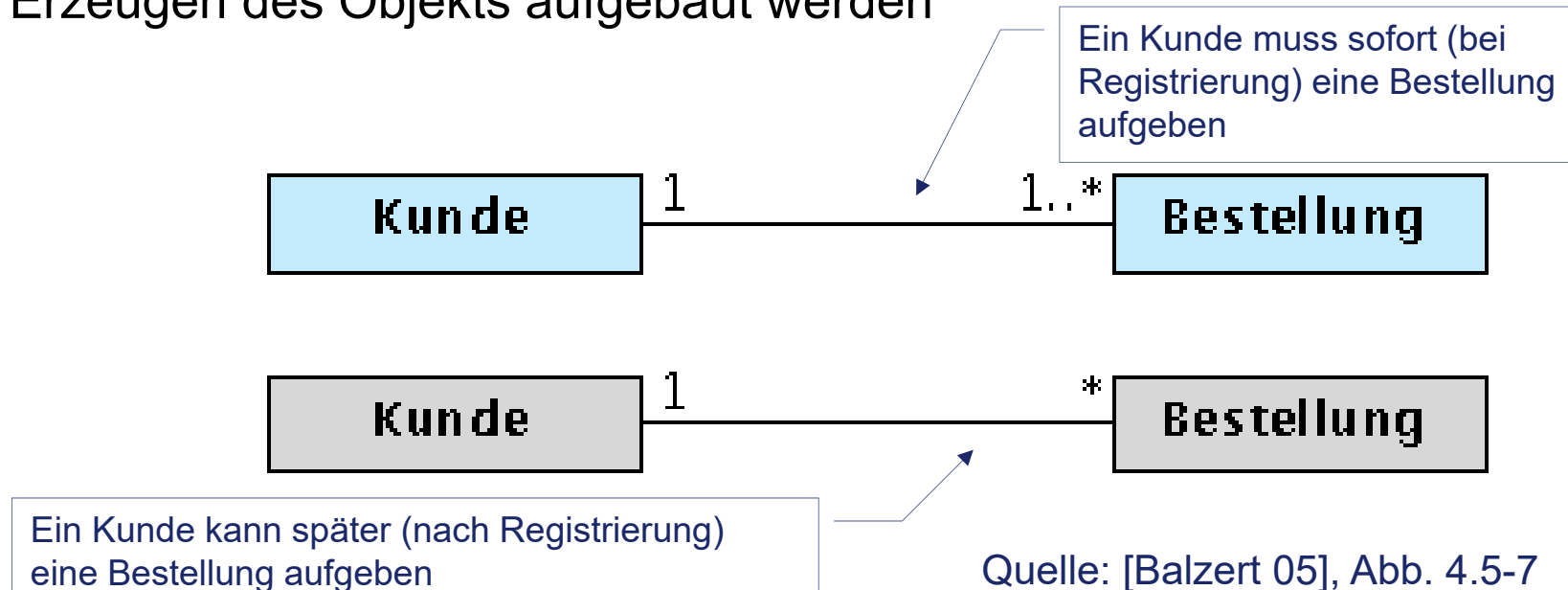
- Beispiel 2:

- Regel: „Auf eine Zulassung muss in spätestens 6 Monaten eine Niederlassung erfolgen. Die Zuordnung der Niederlassung ändert sich nicht mehr“
- Zusammenfassung ist möglich (ggfs. Attribute mit Multiplizität [0..1])
- Achtung: Multiplizitäten ändern sich!



Quelle: [Balzert 05],  
Abb. 4.5-4

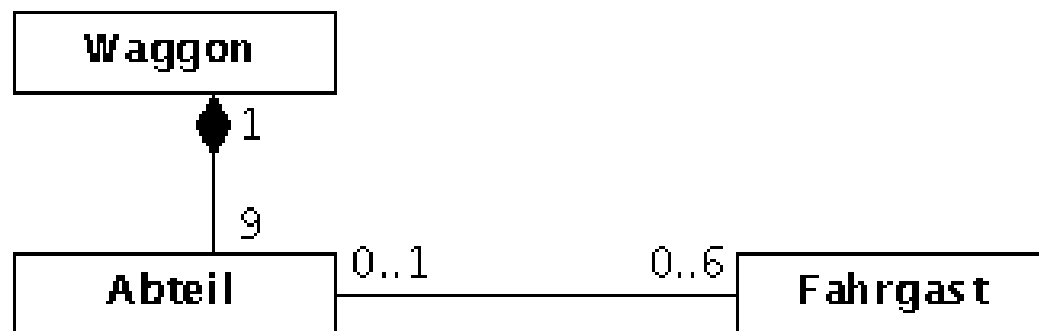
- Schnappschuss oder Historie? (Vgl. [S. 39](#))
- Muss- oder Kann-Assoziation?
  - Wie und wann werden Objekte der Klassen erzeugt?
  - Können beteiligte Objekte gelöscht werden und welche Konsequenzen hat dies?
  - Kann-Verbindung kann zu einem beliebigen Zeitpunkt nach dem Erzeugen des Objekts aufgebaut werden



Quelle: [Balzert 05], Abb. 4.5-7



- Feste Multiplizitätsgrenzen?
  - Unter- bzw. Obergrenze vom Problembereich fest vorgegeben
  - Im Zweifelsfall mit variabler Obergrenze bzw. mit der Untergrenze 0 arbeiten
  - Gelten besondere Restriktionen für Multiplizität, z.B. gerade Anzahl?
  - Beispiel1:

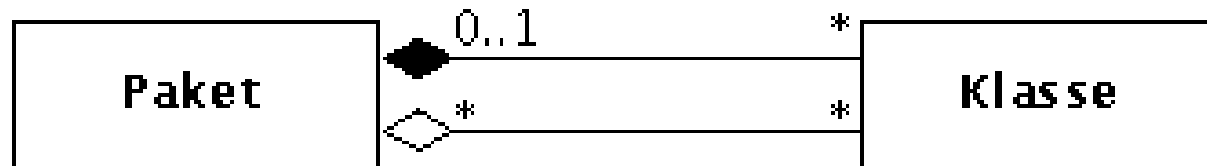


Quelle: [Balzert 05], Abb. 4.5-8

- Beispiel2: Türme von Hanoi

- Art der Assoziation
  - Komposition
    - Kriterien der Definition anwenden
      - „Ist-Teil-Von“-Beziehung (→ physisches Enthaltensein)
      - Lebensdauer der Teile an die Lebensdauer des ganzen gebunden
      - Funktionen des ganzen werden automatisch auf die Teile angewendet (z.B. „kopieren“)
      - Zuordnungen sind fest
    - Muster anwenden (vgl. „Analysemuster“, Kap. 4)
      - Liste, Baugruppe, Stückliste
  - Aggregation
    - Relativ selten
    - Assoziation, die logisches Enthaltensein beschreibt
    - keine feste Zuordnung der Teile zum Ganzen
  - Im Zweifelsfall immer einfache Assoziation!
  - Beispiel1: Türme von Hanoi

- Art der Assoziation
  - Beispiel2:
    - Eine Klasse ist in höchstens einem Paket enthalten  
→ Komposition
    - Eine Klasse kann in mehreren Paketen referenziert werden  
→ Aggregation

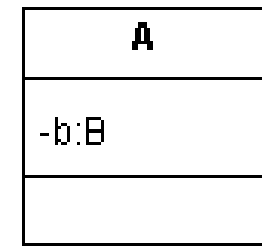


Quelle: [Balzert 05], Abb. 4.5-10

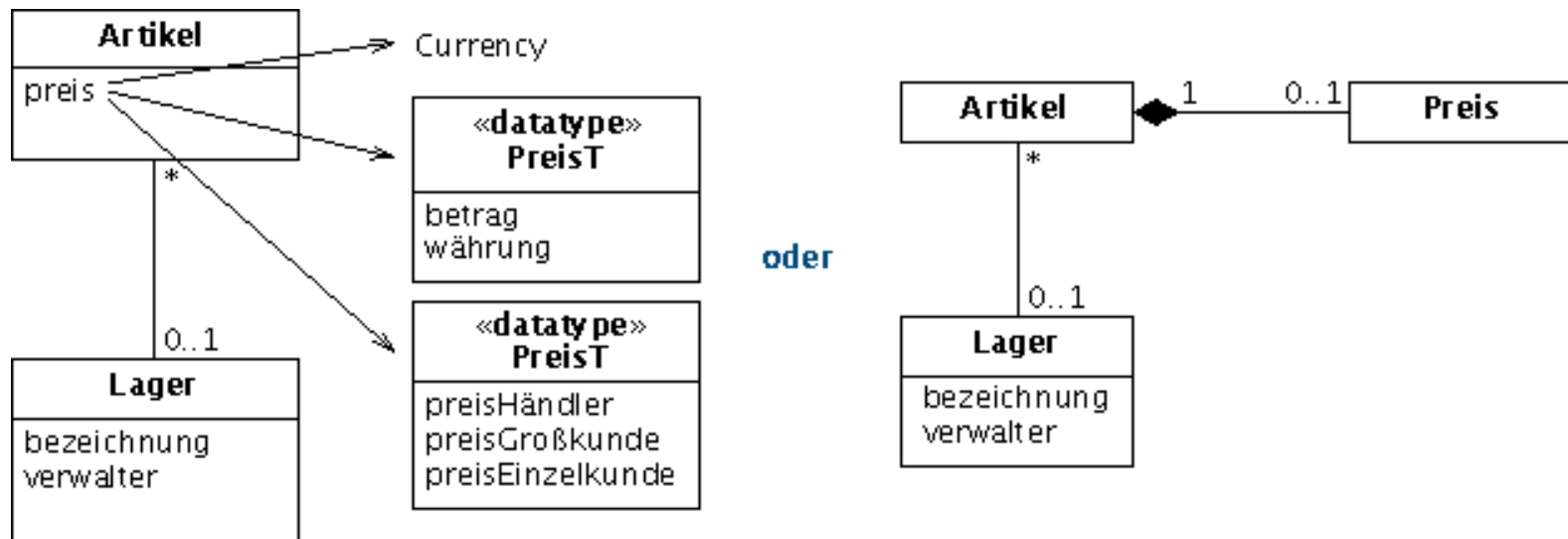
- Assoziation/Komposition mit Klasse oder Attribut?
  - Gründe für die Verwendung einer Assoziation/Komposition
    - Eigene Identität der verknüpften Objekte
    - Gleichgewichtige Bedeutung im System
    - Existenz unabhängig von der Existenz anderer Objekte
      - Dann keine Komposition modellieren!
    - Navigation in beiden Richtungen grundsätzlich möglich



- Gründe vor die Verwendung von Attributen
  - Keine Objektidentität
  - Existenz abhängig von Existenz anderer Objekte
  - Zugriff auf Attributwert immer über Objekt
  - Untergeordnete Bedeutung im System



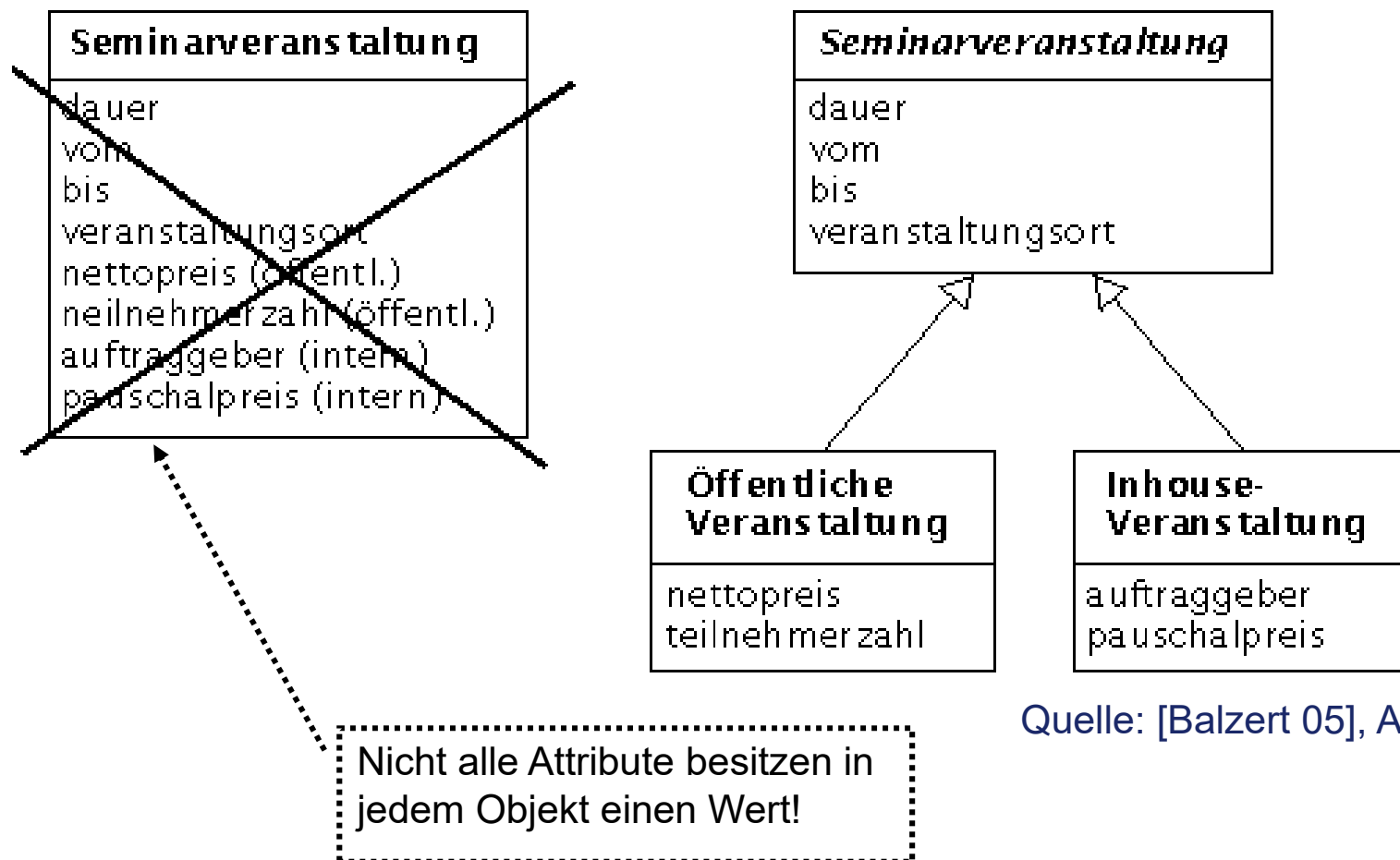
- Beispiel: Klasse oder Attribut?
  - Hier ist die Variante mit einem Attribut „Preis“ sinnvoller!



Quelle: [Balzert 05], Abb. 4.6-1

- **Bottom up:** Generalisierung
  - Können aus gleichartigen Klassen neue Oberklassen gebildet werden?
  - Ist eine vorhandene Klasse als Oberklasse geeignet?
- **Top down:** Spezialisierung
  - Kann jedes Attribut einer Klasse für jedes Objekt einen Wert annehmen?
    - Falls nein, prüfen ob es sinnvolle Unterklassen gibt
  - Kann jede Operation auf jedes Objekt angewendet werden?
    - Falls nein, prüfen ob es sinnvolle Unterklassen gibt
- Ziel von Generalisierung/Spezialisierung
  - Klassendiagramm besser strukturieren
  - Leichter Verständlichkeit des Modells

- Beispiel:



Quelle: [Balzert 05], Abb. 4.7-1

- „Gute“ Vererbung
  - Verbessert das Verständnis des Modells
  - Jede Unterklasse benötigt die geerbten Attribute, Operationen und Assoziationen
  - „IS-A“ - Beziehung liegt vor
    - Ein Objekt der Unterklasse „ist ein“ Objekt der Oberklasse
  - Entwickelte Struktur entspricht den „natürlichen“ Strukturen des Problembereichs
  - Möglichst flache Hierarchien