Objektorientierte Modellierung

Prof. Dr. Roland Dietrich

6. Entwurf mit UML

Ziele und Aufgaben des Objektorientierten Entwurfs

Entwurfsnotationen der UML

Übersicht

1. Objektorientierte Softwareentwicklung ✓
2. Anforderungsanalyse mit UML ✓
   - Anwendungsfalldiagramme
3. Statische Modellierung mit UML ✓
   - Klassendiagramme
     Objekte und Klassen, Assoziationen, Vererbung
   - Paketdiagramme
4. Der Analyseprozess und Analysemuster ✓
5. Dynamische Modellierung mit UML ✓
   - Interaktionsdiagramme (Sequenz- und Kollaborationsdiagramme)
   - Aktivitätsdiagramme
   - Zustandsautomaten
6. Entwurf mit UML
7. Implementierung in C++

Entwurf

- Fragestellung
  - Analyse:
    - Beschreibung der fachlichen Welt
      - Was gibt es dort an Konzepten?
      - Wie sieht eine fachliche Lösung des Problems aus?
  - Entwurf:
    - **wie** soll das System softwaretechnisch realisiert werden?
      - Aus welchen einzelnen Bestandteilen setzt sich die SW-Lösung zusammen?
      - Wie stehen sie miteinander in Beziehung?
- Ziele
  - Festlegung einer Lösungsstruktur („Grob-Entwurf“):
    Die **SW-Architketur**
    - Gliederung des Systems in überschaubare (handhabbare) Einheiten
      - Evtl. **hierarchische** Gliederung
    - Festlegung der Zusammenarbeit der Einheiten
  - Beschreibung der einzelnen Einheiten („Feinentwurf“)

Objektorientierter Entwurf

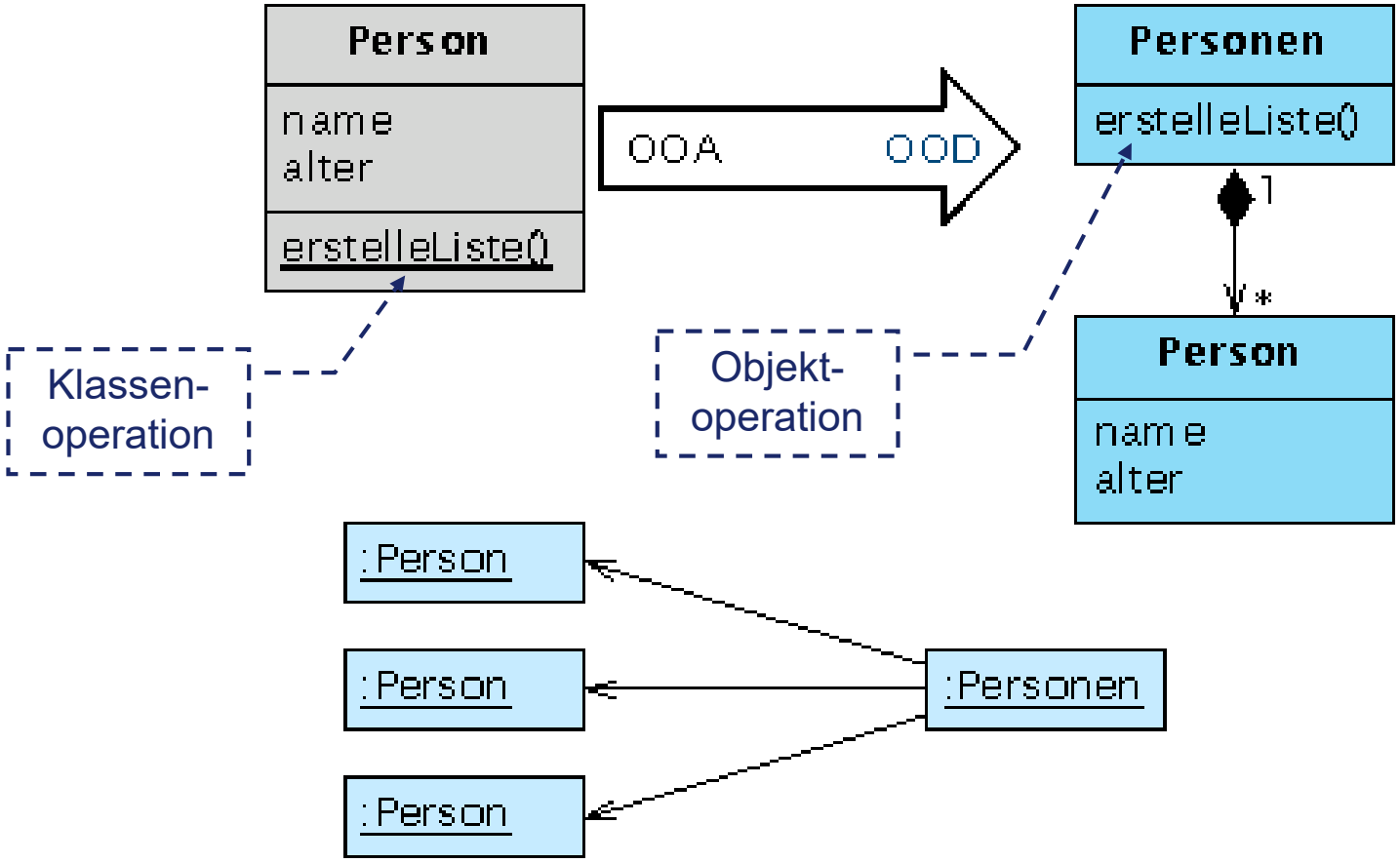
- Ziel der Objektorientierten Analyse (OOA)
  - Objektorientiertes Analysemodell als fachliche Lösung
- Ziel des Objektorientierten Entwurfs (OOD)
  - Objektorientiertes Entwurfsmodell als ...
    - technische Lösung
    - Spiegelbild des Programms auf höherem Abstraktionsniveau
  - Die **Programmiersprache** liegt jetzt fest und wird berücksichtigt
- Vorteil der Objektorientierung
  - Konzepte und Notation der Analyse gelten auch in Entwurf und Implementierung
  - Erweiterung der Konzepte und Notationen für den Entwurf, z.B.
    - Sichtbarkeiten für Attribute und Operationen
    - Navigierbarkeit von Assoziationen
      - unidirektional
      - bidirektional

Objektorientierter Entwurf

- Aufgaben beim Objektorientierten Entwurf
  - Ausgangspunkt
    - Statisches Analysemodell (Klassendiagramm)
    - Anwendungsfälle
  - Das Statische Analysemodell muss detailliert werden
    - Alle Operationen die erforderlich sind, um die Anwendungsfälle auszuführen
      - Ergeben sich teilweise aus der dynamischen Analyse (vgl. Kap. 5)
      - Hilfs- und Standard-Operationen ergänzen
        - » z.B. get-/set-, link-/unlink, Konstruktoren/Destruktoren
    - Anpassen von Klassendetails an die gewählte Programmiersprache
      - z.B. Attributbezeichner und -typen Programmiersprachenkonform wählen
    - Festlegung der Navigierbarkeit von Assoziationen
      - So, dass alle Operationen realisierbar sind
    - Ergänzung weiterer Klassen (technische Klassen), z.B.
      - Zur Realisierung einer Objektverwaltung (vgl. S. 5-9)
      - Zur Darstellung von Objekten an einer grafischen Benutzungsoberfläche
      - Zum Abspeichern von Objekten in einer Datenbank

Objektorientierter Entwurf

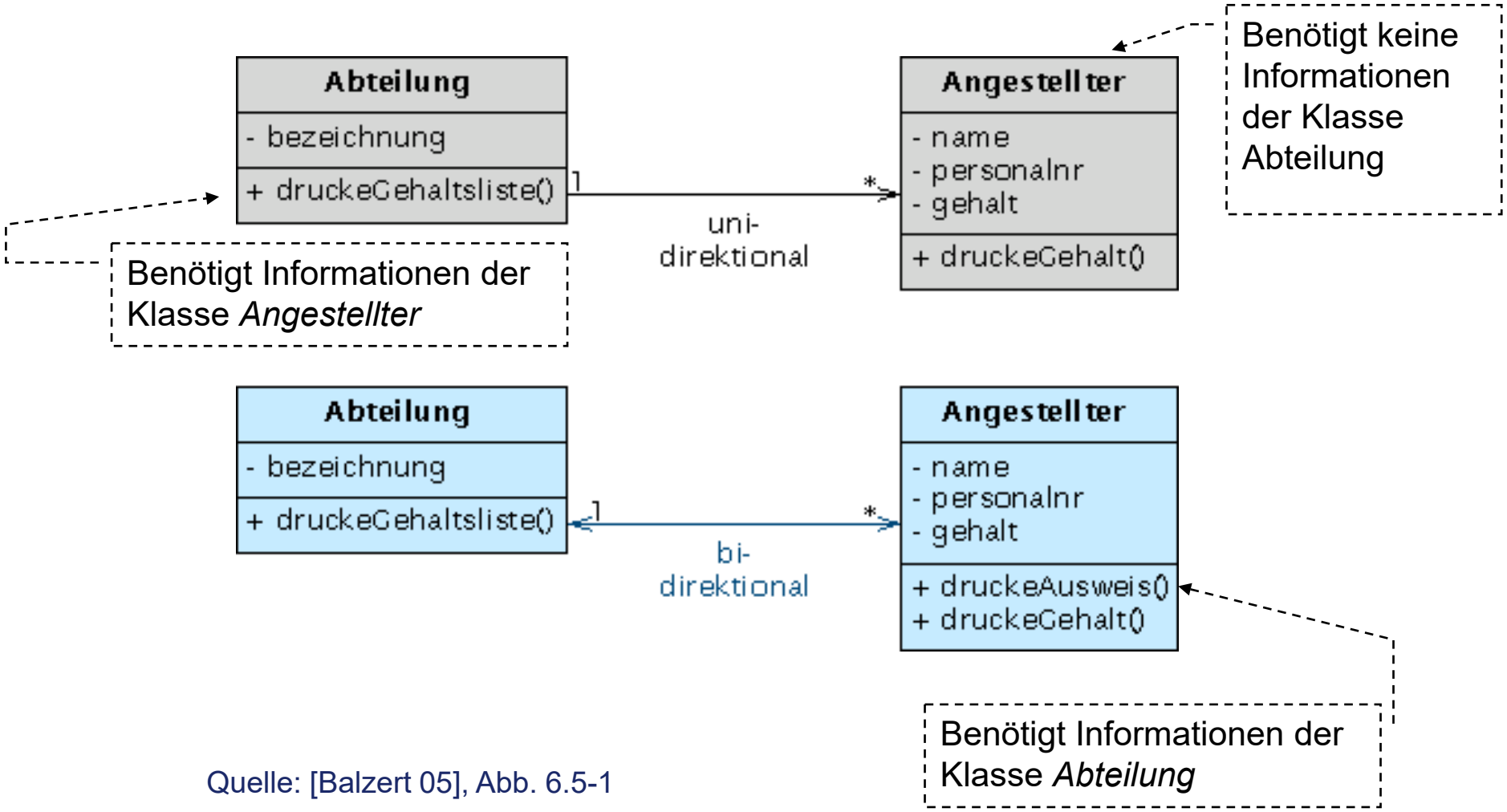
- Aufgaben beim Objektorientierten Entwurf
  - Beispiel: Ergänzung einer Objektverwaltung



Quelle: [Balzert 05], Abb. 6.1-3

Hochschule Aalen

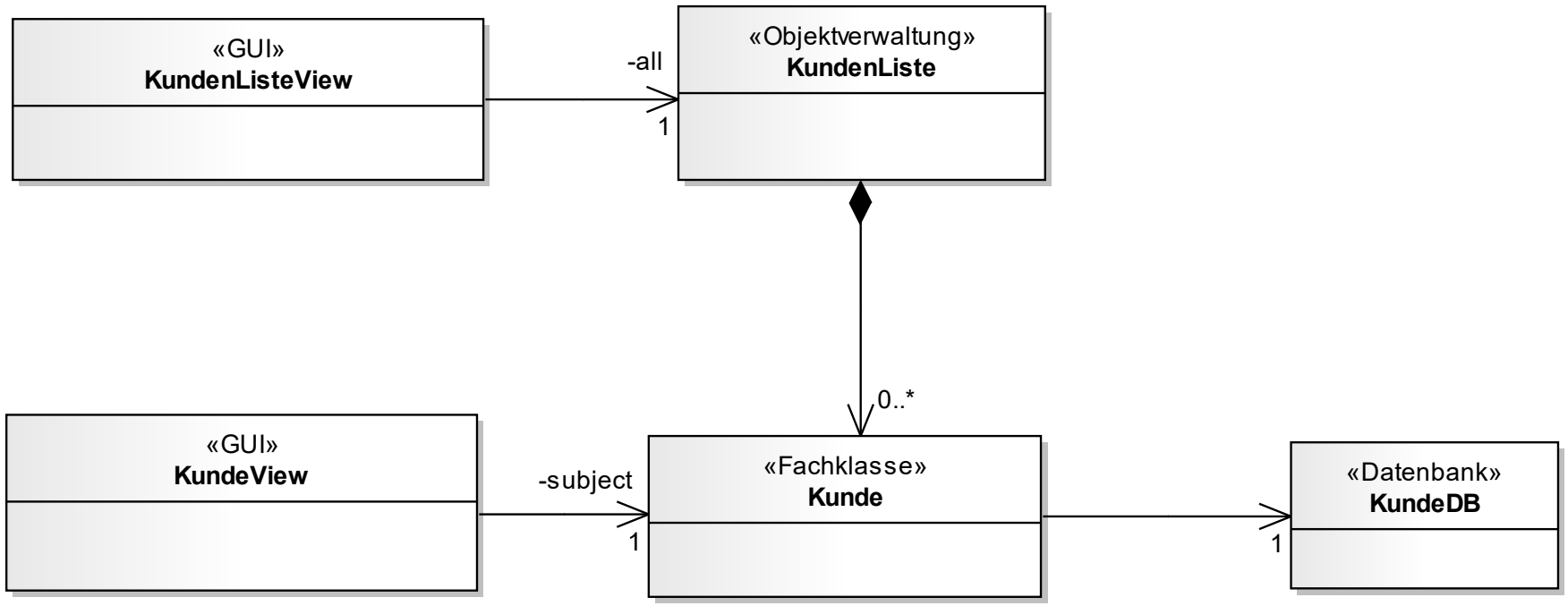
- Aufgaben beim Objektorientierten Entwurf
  - Beispiel: Festlegen der Navigierbarkeit von Assoziationen



Quelle: [Balzert 05], Abb. 6.5-1

Objektorientierter Entwurf

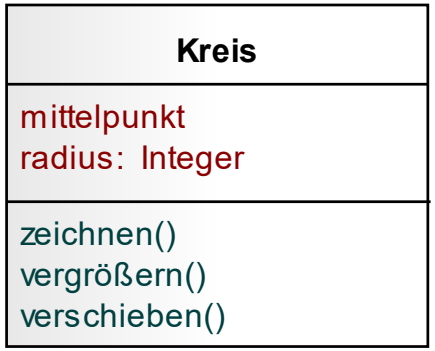
- Aufgaben beim Objektorientierten Entwurf
  - Beispiel: Ergänzung von technischen Klassen für eine Fachklasse



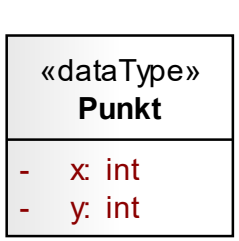
UML im Entwurf

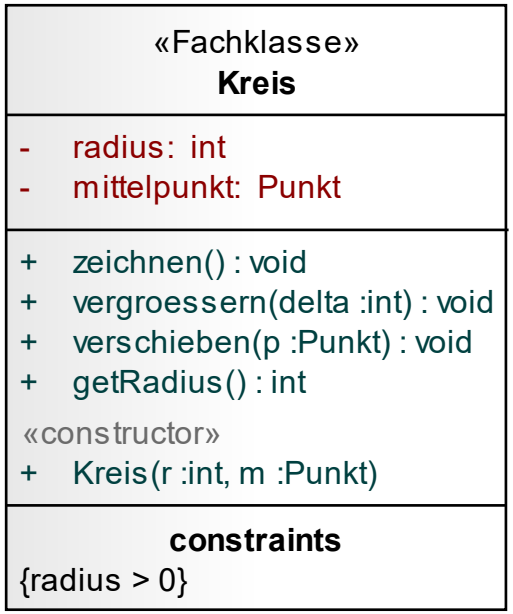
- Klassen im Entwurf
  - Dieselbe Notation wie in der Analyse
    - Typisch für Entwurf:
      - Darstellung von Sichtbarkeiten
      - Darstellung von Attributtypen
      - Darstellung der Signatur von Operationen
  - Namen sollten der Programmiersprachen-Syntax entsprechen
    - Klassen-, Attribut-, Operations- Typnamen
  - Vollständige Operationsliste
    - Get-/Set-Methoden
    - Link-Operationen
    - Konstruktoren/Destruktoren
  - Stereotypen können kennzeichnen
    - Zugehörigkeit von Klassen zu Architekturschichten
      - z.B. <<user interface>>, <<data base>>
    - Verwaltungsoperationen
      - z.B. Konstruktoren/Destruktoren, set-/get-Methoden

Klasse `Kreis` in der Analyse



Klasse `Kreis` im Entwurf

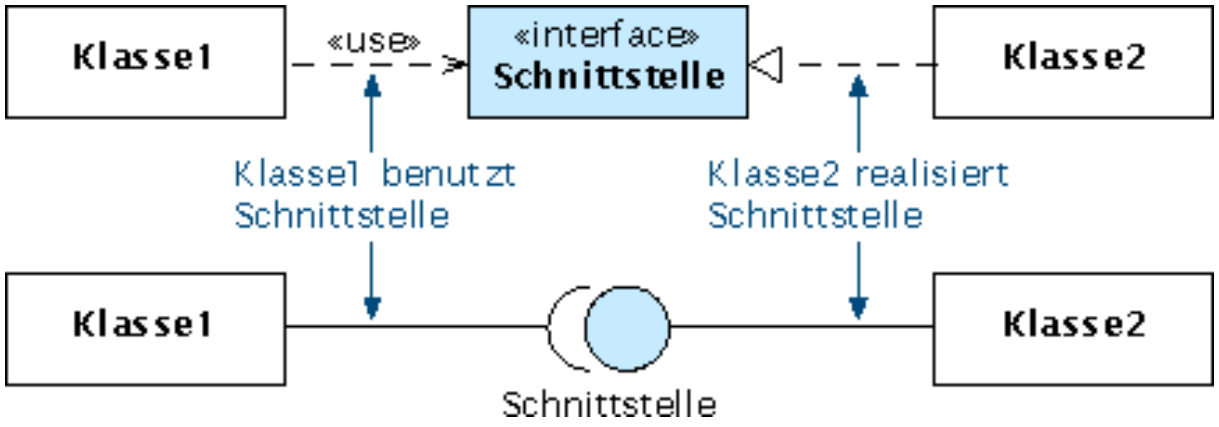




- **Schnittstelle** *(interface)*
  - Spezifiziert einen Ausschnitt aus dem Verhalten einer Klasse
  - Besteht nur aus Signaturen von Operationen
    - **UML:** Klasse mit Stereotyp <<*Interface*>>
      - *Anmerkung:*
        im Allgemeinen nur Operationen
      - Attribute zwar möglich, aber selten sinnvoll
  - Wird durch Klassen realisiert bzw. implementiert
    - **UML:** gestrichelter Vererbungspfeil
  - Können von anderen Klassen benutzt werden
    - d.h. andere Klassen benutzen eine Klasse, die die Schnittstelle implementiert
    - **UML:** Abhängigkeits-Pfeil
  - Verwendung:
    - Bei der Spezifikation von **Software-Komponenten** spielen Schnittstellen eine entscheidende Rolle
      - Siehe „Software Engineering“, 4. Semester

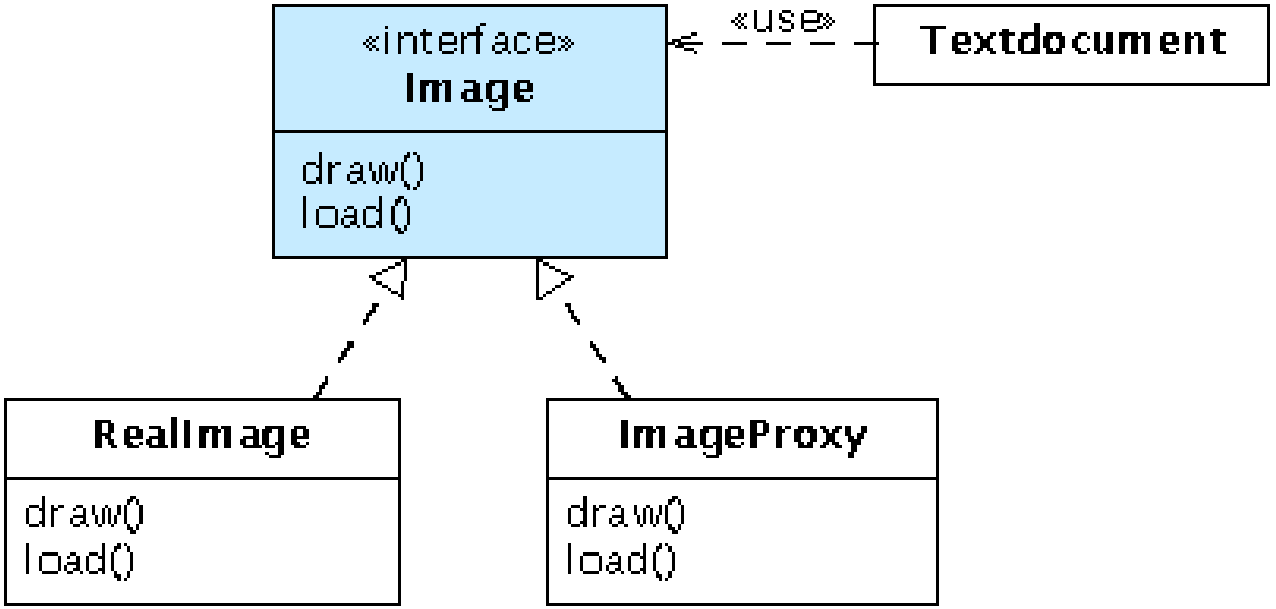
Quelle: [Balzert 05], Abb. 6.1-5

- Schnittstelle (*interface*)
  - Notation für Realisierung und Benutzung von Schnittstellen



Quelle: [Balzert 05], Abb. 6.1-7

- Schnittstelle (*interface*)
  - Beispiel



Quelle: [Balzert 05], Abb. 6.1-6

- Sichtbarkeit (Visibility)
  - Klassen bilden einen **Namensraum** für ihre Elemente (Attribute und Operationen)
  - Die Elemente eines Namensraums können eine **Sichtbarkeit** haben
    - ***public:*** Element ist für andere Elemente außerhalb des Namensraums sichtbar
    - ***private:*** Element ist nur innerhalb des eigenen Namensraums sichtbar
    - ***protected:*** Element ist innerhalb aller Elemente sichtbar, die Spezialisierungen zum eigenen Namensraum bilden
    - ***package:*** Elemente dürfen nur zu einem Namensraum gehören, der kein Paket darstellt, und sind sichtbar für alle Elemente im gleichen Paket
      - Beispiel: ein Attribut einer Klasse mit Sichtbarkeit *package* ist in allen Klassen des selben Pakets ebenfalls sichtbar.
    - Notation Sichtbarkeiten in UML:

      public: **+**, *protected:* **#**, *private:* **-**, *package:* **~**

UML im Entwurf

- Geheimnisprinzip
  - Attribute grundsätzlich als protected oder private vereinbaren
  - set-/get-Methoden zum Setzen/Lesen von Attributwerten
- Attributspezifikationen (vgl. 3-17)
  - Vollständig

```
Sichtbarkeit / name : Typ [Multiplizität] =
  Anfangswert {Eigenschaftswert}
```

  - Kurzform (minimale Spezifikation im Entwurf):

```
Sichtbarkeit name : Typ
```

  - Beispiel:

Quelle: [Balzert 05], Abb. 6.2-2

UML im Entwurf

- Eigenschaftswerte für Attribute
  - ***readOnly:*** Attributwert darf nicht mehr geändert werden

    ```
    kontonr {readOnly}
    ```
  - ***ordered:*** Attribut besteht aus einer Menge von geordneten Werten; Duplikate sind **nicht** erlaubt

    ```
    vorname[1..3] {ordered}
    ```
    mit den Attributwerten Daniela, Maria und Elke
  - ***bag:*** Attribut besteht aus einer Menge von ungeordneten Werten; Duplikate sind erlaubt

    ```
    noten[1..5] {bag}
    ```
    mit den Attributwerten 1.0, 2.0, 1.3, 2.0, 2.3
  - ***sequence:*** Attribut besteht aus einer Menge von geordneten Werten; Duplikate sind erlaubt

    ```
    wohnsitze[1..3] {sequence}
    ```
    mit den Attributwerten Dortmund, Bochum und Dortmund

- Eigenschaftswerte für Attribute
  - ***subsets:*** Attribut besteht aus einer Menge von Werten; zulässige Attributwerte bilden eine Teilmenge eines anderen Attributs

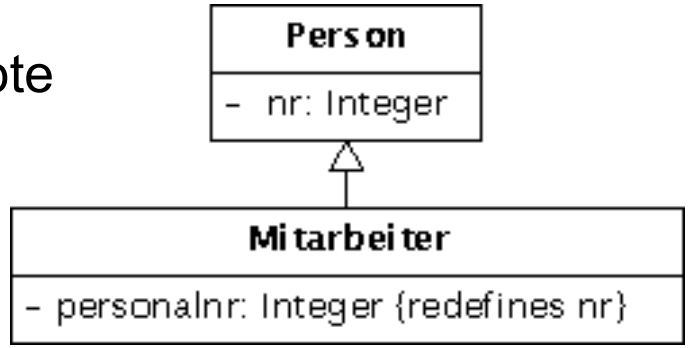
    `geradeZiffern {subsets ziffern}` mit den Werten 2, 4, 6, 8

    `ungeradeZiffern {subsets ziffern}` mit den Werten 1, 3, 5, 7, 9

    `nullZiffer {subsets ziffern}` mit dem Wert 0
  - ***union:*** Attribut besteht aus einer Menge von Werten; es ergibt sich aus der Vereinigung aller mit *subsets* definierten Teilmengen

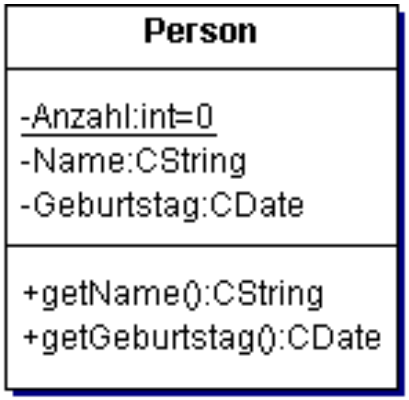
    `ziffern {union}`
  - ***redefines:*** Attribut überschreibt eine geerbte Attributdefinition
  - Weitere Eigenschaftswerte und Einschränkungen sind beliebig definierbar

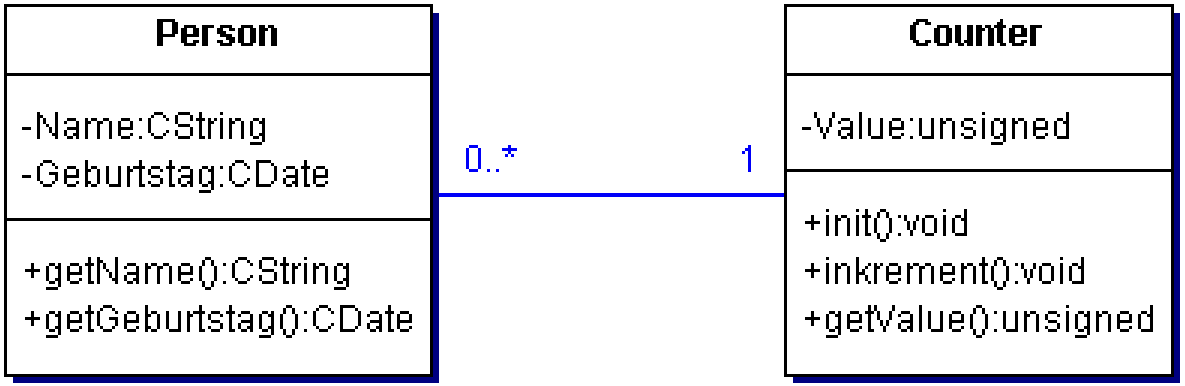
    ```
    nummer: int {key}
    hinflug: Date
    rueckflug: Date {rueckflug >= hinflug}
    wert: int {wert > 4}
    ```



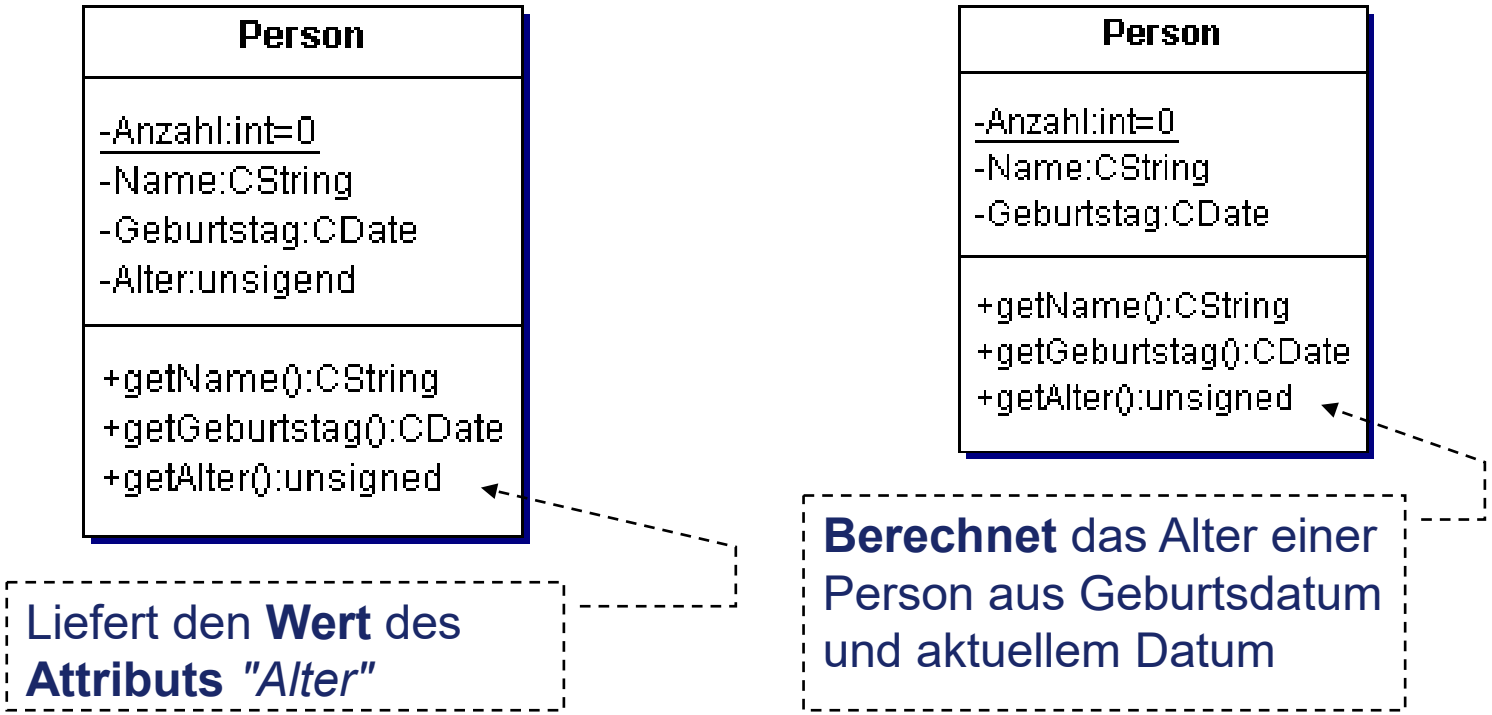
Quelle: [Balzert 05], Abb. 6.2-3

- Klassenattribut
  - Als Klassenattribut realisieren
    - Falls die Programmiersprache das ermöglicht
      - In Java und C++: **static**-Attribute (siehe Kap. 7)
  - Als Objektattribut einer separaten Klasse realisieren
    - Diese Klasse besitzt dann nur ein einziges Objekt mit dem Wert des Klassenattributs
    - Alle Objekte, die das Klassenattribut haben, haben Zugriff auf dieses Objekt





- Abgeleitetes Attribut
  - Als Operation realisieren, die stets den aktuellen Wert ermittelt
  - Als Attribut realisieren (→ Konsistenzprüfung!)
  - Beispiel:



UML im Entwurf

- Operationen
  - Notation der Signatur

    ```
    Sichtbarkeit name (Parameterliste) : Ergebnistyp
      {Eigenschaftswert}
    ```

  - Kurzform:

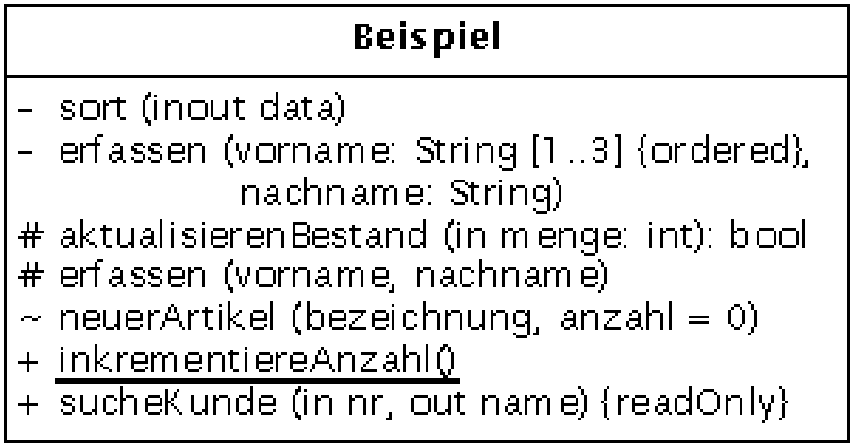
    ```
    Sichtbarkeit name()
    ```

  - Parameterliste
    - besteht aus einem oder mehreren Parametern, die durch Komma getrennt sind
  - Notation:

    ```
    Richtung parametername: Typ [Multiplizität] =
      Anfangswert {Eigenschaftswert}
    ```

  - Überladen *(overloading)*
    - Mehrfache Verwendung des gleichen Operationsnamens in einer Klasse
    - Operationen müssen sich in ihrer Parameterliste unterscheiden
    - Wird von vielen Programmiersprachen unterstützt (Java, C++, C#)

Beispiel



Quelle: [Balzert 05], Abb. 6.3-2

- Operationen
  - Richtung eines Parameters
    - **in**, wenn es sich um einen reinen Eingabeparameter handelt
      - Operation kann nur lesend darauf zugreifen

        ```
        erfassen (in name: String)
        ```
    - **out**, wenn es sich um einen reinen Ausgabeparameter handelt
      - Operation erzeugt die Werte dieser Parameter und gibt sie an die aufrufende Operation

        ```
        sucheKunde (in nummer: int, out name: String)
        ```
    - **inout**, wenn es sich sowohl um einen Eingabe- als auch Ausgabeparameter handelt
      - Operation liest diese Werte, verändert sie und gibt sie an die aufrufende Operation zurück

        ```
        sortieren (inout zahlenfolge : int [1..10])
        ```
    - **return**, wenn es sich um Rückgabeparameter handelt

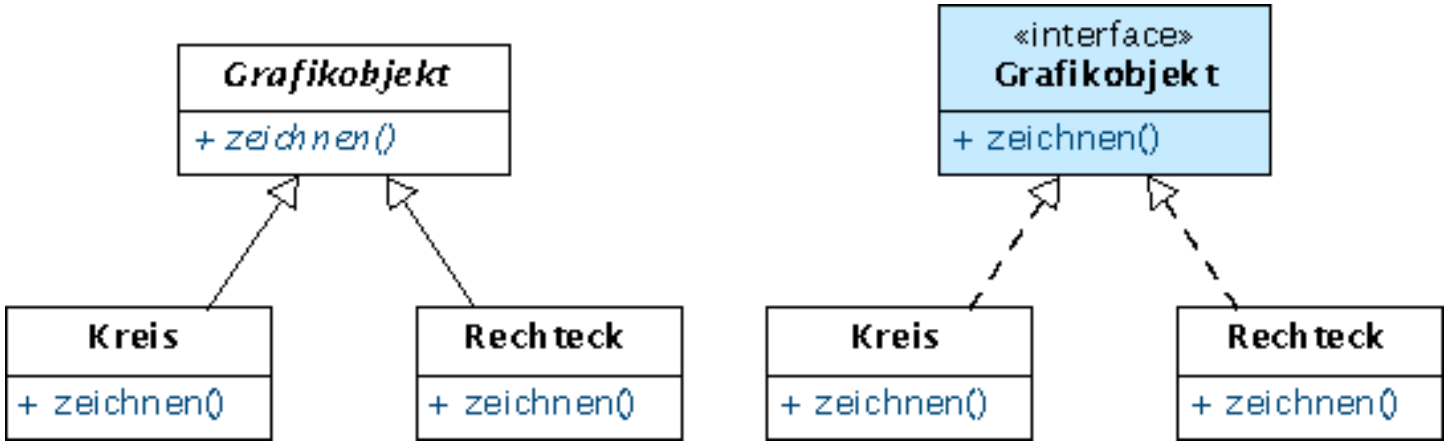
      ```
      berechneQuadrat (in zahl: int, return quadrat: int)
      ```
  - Ergebnistyp
    - Typ des von der Funktion an den Aufrufer zurückgegeben Werts

      ```
      Quadrat (in zahl: int): int
      ```

- Operationen
  - Eigenschaftswert
    - ***readOnly*** – Operation kann keine Attribute verändern, sondern nur lesenen Zugriff durchführen

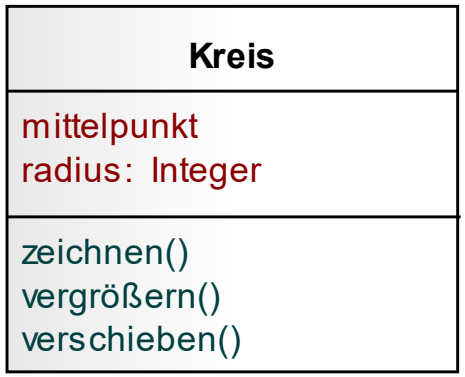
      ```
      leseNamen(): String {readOnly}
      ```
  - Beschreibung einer Operation
    - Meistens umgangssprachliche Beschreibung ausreichend
    - Bei Bedarf Beschreibung mittels Vor- und Nachbedingung
      - **Vorbedingung** *(precondition)*: Beschreibt, welche Bedingungen vor dem Aktivieren einer Operation erfüllt sein müssen
      - **Nachbedingung** *(postcondition)*: Beschreibt die Änderung, die durch die Operationen bewirkt wird

- Abstrakte Operation
  - Besteht nur aus der **Signatur**
    - Name der Operation
    - Namen und Typen aller Parameter
    - Ergebnistyp
  - Besitzt keine Implementierung
  - Definiert gemeinsame Schnittstelle für Unterklassen
  - Notation: *Kursiv*-Schreibung



Quelle: [Balzert 05], Abb. 6.3-3

- Operationen – von der Analyse zum Entwurf
  - Analyse
    - Eintrag von Operationen, die für das Verständnis des Fachkonzepts wichtig sind
    - Keine oder unvollständige Signaturen
  - Entwurf
    - Eintrag **aller** Operationen, die benötigt werden
    - Vollständige Signaturen
    - Weitere Details, z.B. Eigenschaftswerte

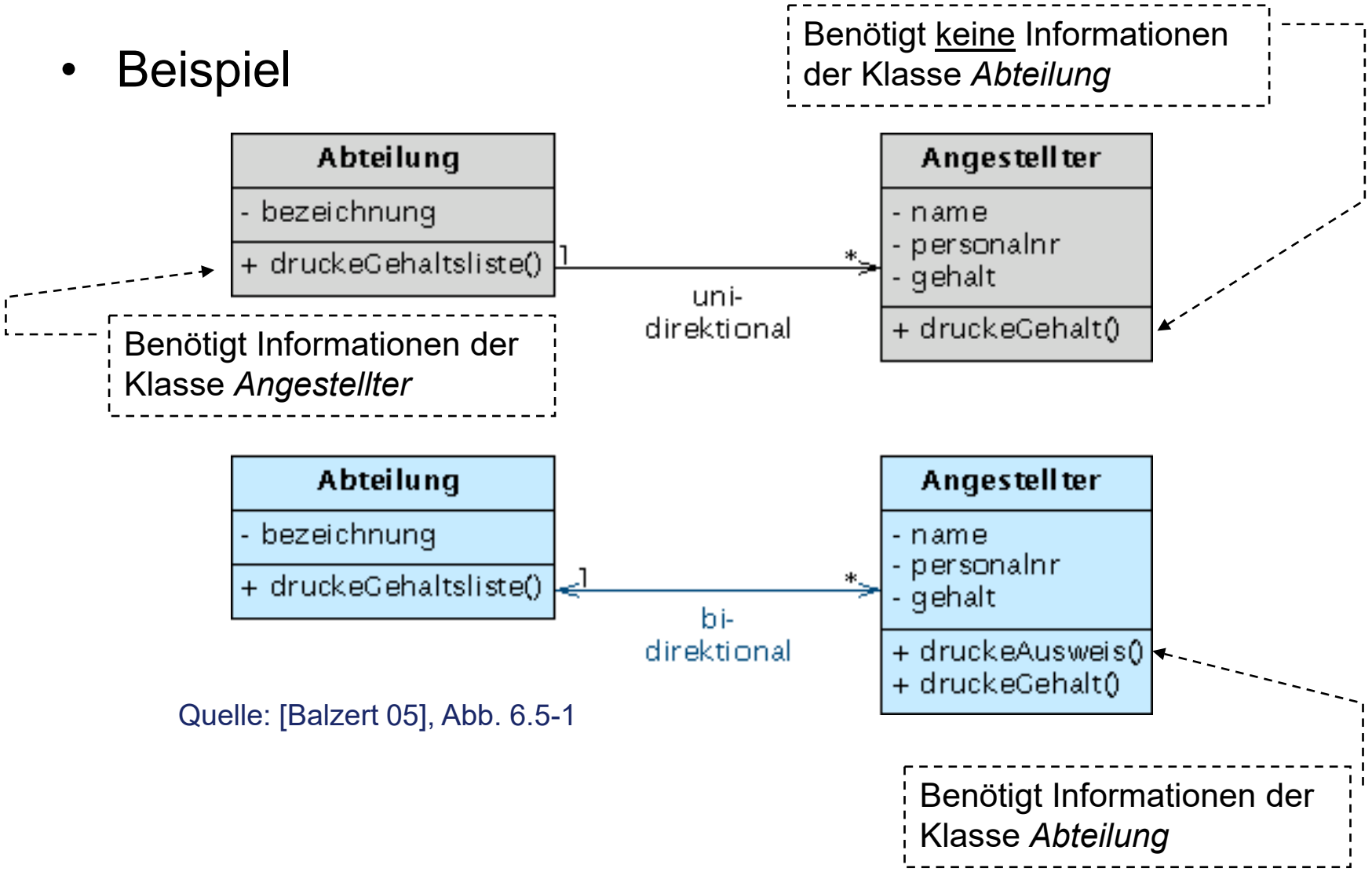


OOA → OOD

- Assoziationen
  - Navigierbarkeit
    - Definition
      - Besteht zwischen zwei Klassen A und B eine Assoziation, dann ist diese Assoziation **von A nach B navigierbar**, wenn Objekte von A auf Objekte von B zugreifen können
    - Analyse
      - Alle Assoziationen inhärent bidirektional
        - » Eine Assoziation zwischen Klassen A und B ist sowohl von A nach B als auch von B nach A navigierbar.
    - Entwurf
      - Festlegung, ob uni- oder bidirektionale Implementierung der Navigierbarkeit der Assoziation
      - Richtungen werden durch die notwendigen Zugriffe im Klassendiagramm bei der Ausführung der Operation bestimmt

Hochschule Aalen

- Beispiel

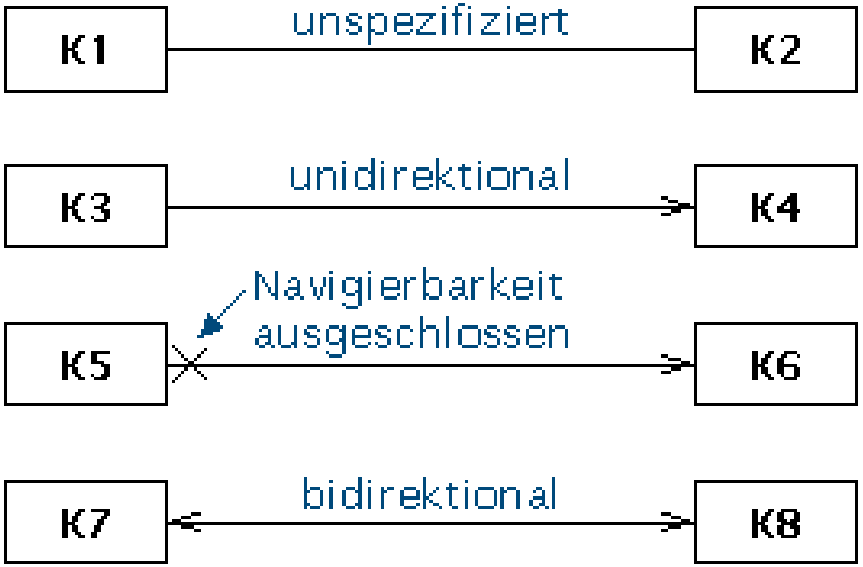


Quelle: [Balzert 05], Abb. 6.5-1

Hochschule Aalen

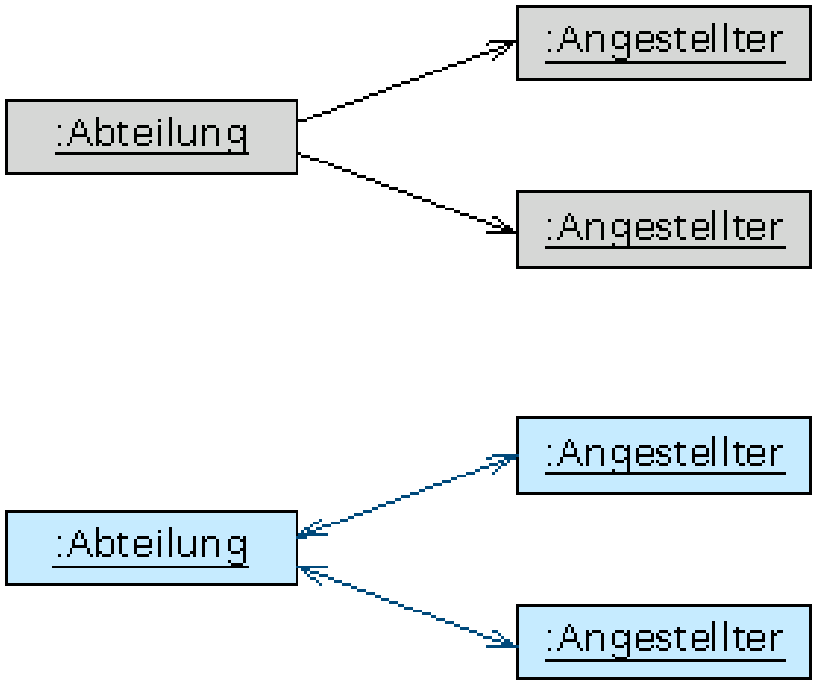
- Assoziationen
  - Notation Navigierbarkeit UML (vgl. S. 7)

Im Klassendiagramm



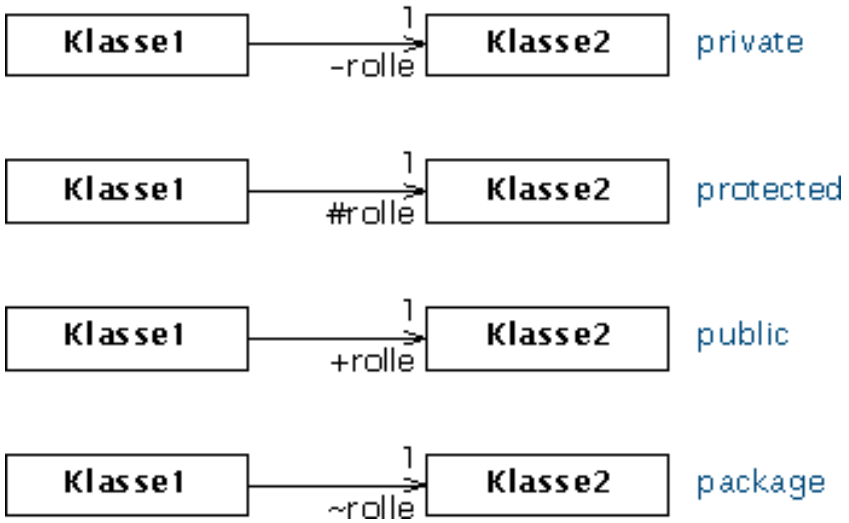
Quelle: [Balzert 05], Abb. 6.5-2

Im Objektdiagramm



Quelle: [Balzert 05], Abb. 6.5-3

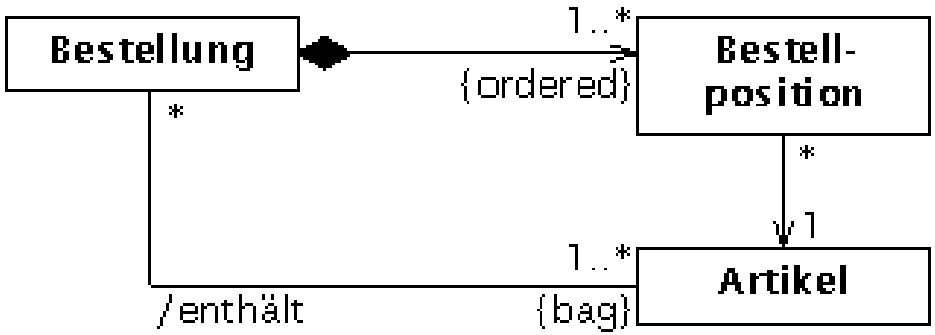
- Assoziationen
  - Multiplizität
    - Im OOD-Modell kann Angabe der Multiplizität auf einer Seite fehlen, wenn in dieser Richtung keine Navigation stattfindet
      - Sie ist in diesem Fall irrelevant
  - Sichtbarkeit
    - Für Assoziationen können in der UML zusätzlich Sichtbarkeiten angegeben werden
    - Analog zu Attributen und Operationen
      - Verwendung von `+, #, -, ~` als Präfix des Rollennamens



Quelle: [Balzert 05], Abb. 6.5-8

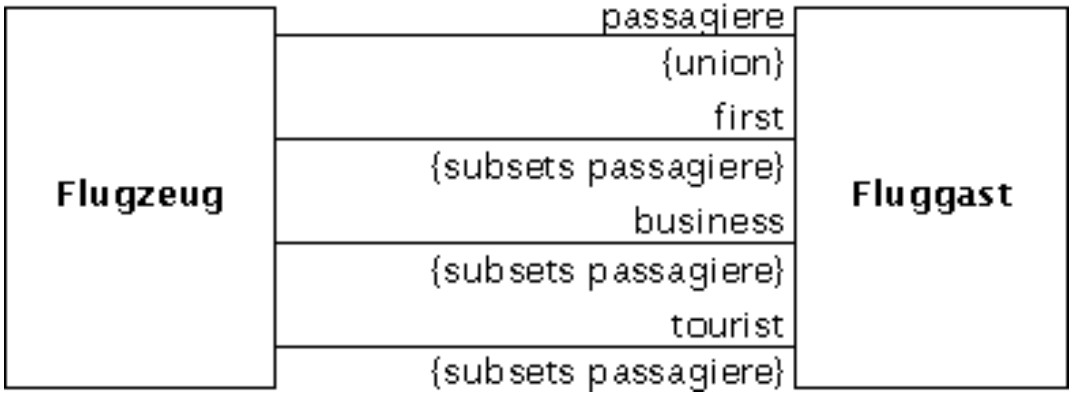
- Assoziationen: Eigenschaftswerte
  - Standardmäßige Eigenschaftswerte:
    - {readOnly}:
      - definiert, dass ein einmal assoziiertes Objekt nicht mehr gelöscht oder durch ein anderes ersetzt werden kann
    - {subsets <Eigenschaft>}:
      - beschreibt eine Teilmenge von Objektbeziehungen
    - {union}:
      - definiert, dass dieses Assoziationsende die Vereinigungsmenge aller Assoziationen bildet, die mit subsets gekennzeichnet sind
    - {redefines}:
      - definiert das Assoziationsende als Redefinition eines anderen Assoziationsendes
    - {ordered}:
      - definiert eine Ordnung auf der Menge der Objektbeziehungen

- Assoziationen: Eigenschaftswerte
  - Standardmäßige Eigenschaftswerte:
    - {bag}:
      - definiert, dass ein Objekt mehrmals in einer Menge von Objektbeziehungen vorkommen darf
    - {sequence}, {seq}:
      - definiert die Kombination von {ordered} und {bag}, d.h. ein Objekt kann mehrmals in einer Menge von Objektbeziehungen vorkommen, wobei die Objekte zusätzlich geordnet sind
  - Beispiel:
    - Geordnete Bestellpositionen innerhalb einer Bestellung



Quelle: [Balzert 05], Abb. 6.5-9

- Assoziationen: Eigenschaftswerte
  - Beispiel:
    - Unterteilung von Flugpassagieren in First-, Business- und Tourist-Passagiere



Quelle: [Balzert 05], Abb. 6.5-10