

Arbeitsgruppe Software Engineering Prof. Elke Pulvermüller

Universität Osnabrück
Institut für Informatik, Fachbereich Mathematik / Informatik
Raum 31/318, Albrechtstr. 28, D-49069 Osnabrück

elke.pulvermueller@informatik.uni-osnabrueck.de

<http://www.inf.uos.de/se>

Sprechstunde: mittwochs 14 – 15 und n.V.



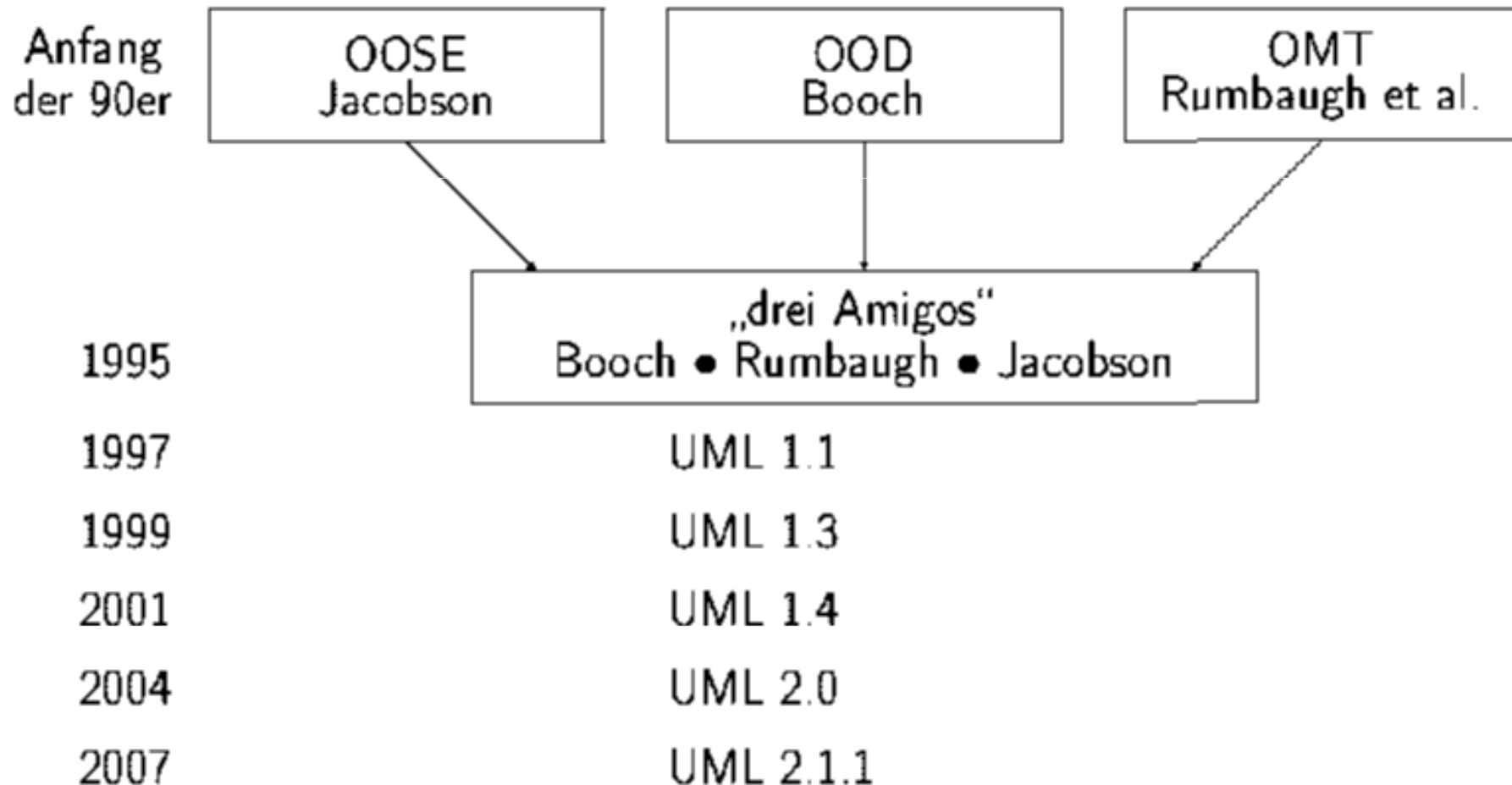
- 1 Software-Krise und Software Engineering**
- 2 Grundlagen des Software Engineering**
- 3 Projektmanagement**
- 4 Konfigurationsmanagement**
- 5 Software-Modelle**
- 6 Software-Entwicklungsphasen, -prozesse, -vorgehensmodelle**
- 7 Qualität**
- 8 ... Fortgeschrittene Techniken**

- 5.1 Grundlagen und Modelltypen**
- 5.2 Programmablaufplan**
- 5.3 Struktogramm**
- 5.4 Funktionsbaum**
- 5.5 Strukturierte Analyse (SA)**
- 5.6 EBNF und Syntaxdiagramm**
- 5.7 Entity-Relationship-Modell (ERM)**
- 5.8 Objektorientierte Modellierung mit UML**

5.8 OO Modellierung mit UML: Grundlagen

- **Unified Modeling Language**
- **UML ist eine „general-purpose“ graphische Modellierungssprache zur Spezifikation, zur Visualisierung, zur Konstruktion und zur Dokumentation / Kommunikation der Artefakte eines Software Systems**
- **Umfasst eine Menge von graphischen Elementen, die nach bestimmten Regeln in verschiedenen Diagrammen (für die verschiedenen Sichten auf das System) kombiniert werden**
- **Keine Programmiersprache, stattdessen Unabhängigkeit von Programmiersprachen (und auch von Anwendungsdomänen)**
- **Nicht vollständig durchformalisiert (halbformal)**
- **Gibt keine Methode zur Softwareentwicklung**
- **Kein Ersatz für geschriebenen Text**

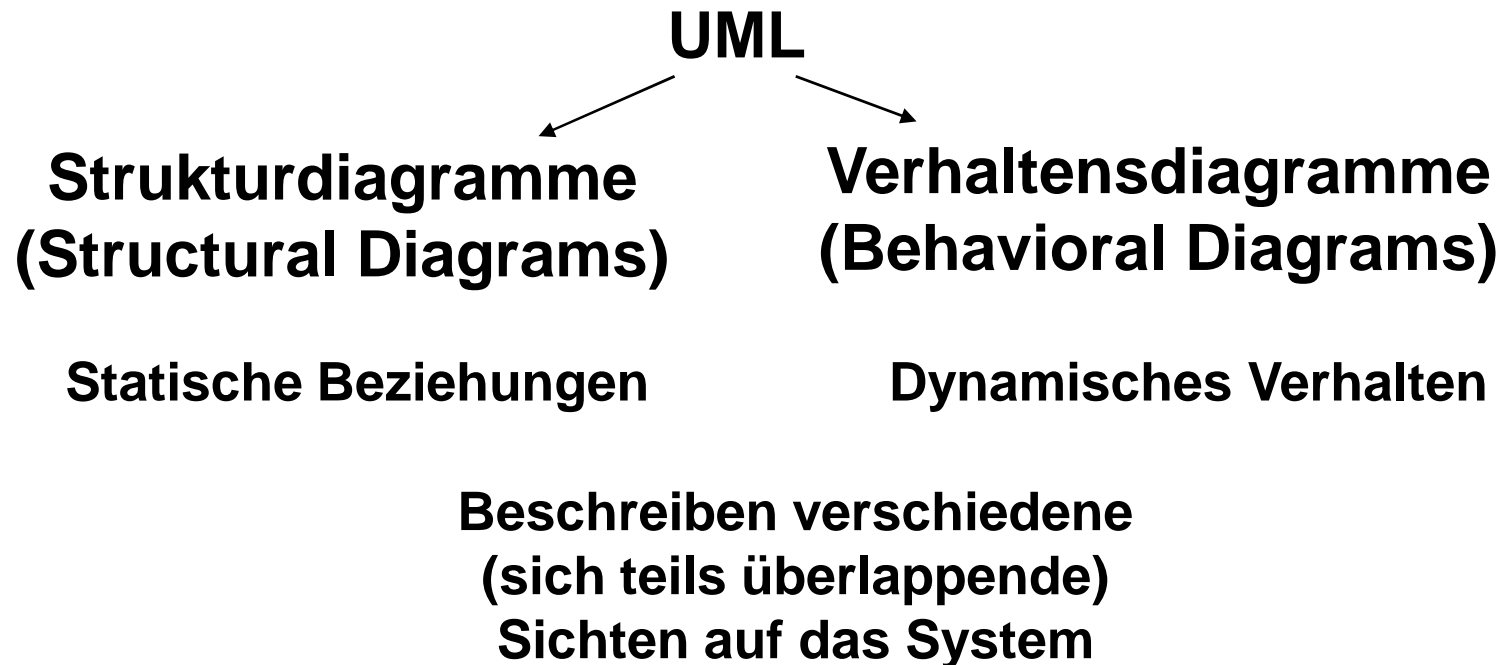
5.8 OO Modellierung mit UML: UML Wurzeln



Standardisiert als Industriestandard von der Object Management Group (OMG)
[<http://www.uml.org>]

5.8 OO Modellierung mit UML: Diagramme

- UML bietet eine Vielzahl unterschiedlicher Diagrammarten an (UML 1.x: 9 Diagrammarten, UML 2.0: 13 Diagrammarten)
- Jede Diagrammart besitzt spezifische grafische Modellierungselemente
- Jede Diagrammart beschreibt eine bestimmte Sicht auf das System durch die Anordnung der Modellierungselemente



5.8 OO Modellierung mit UML: Diagramme

■ Structural Diagrams (static Relations)

Class Diagram (Klassendiagramm)

Object Diagram (Objektdiagramm)

Package Diagram (Paketdiagramm)

Component Diagram (Komponentendiagramm)

Deployment Diagram (Einsatz- und Verteilungsdiagramm)

Composite Structure Diagram (Kompositionsstrukturdiagramm), seit UML 2.x

■ Behavioral Diagrams (dynamic Behavior)

Use Case Diagram (Anwendungsfalldiagramm)

Interaction Diagram (Interaktionsdiagramm)

State Chart / Machine Diagram (Zustandsdiagramm)

Activity Diagram (Aktivitätsdiagramm)

Sequence Diagram
(Sequenzdiagramm)

Timing Diagram
(Zeitdiagramm), seit
UML 2.x

Interaction Overview
Diagram, seit UML 2.x

Communication Diagram (Kommunikationsdiagramm), UML 2.x
= Collaboration Diagram (Kollaborationsdiagramm), UML 1.x

5.8 OO Modellierung mit UML: Diagramme

- **Klassendiagramme (class diagrams):** beschreiben den strukturellen Aufbau eines Systems aus Klassen.
- **Objektdiagramme (object diagrams):** beschreiben zu einem bestimmten Zeitpunkt die Menge der existierenden Objekte samt Attributwerten und ihrer Beziehungen (Snapshot).
- **Paketdiagramme (package diagrams):** stellen die Strukturierung der Klassen eines Systems durch Pakete dar (Gruppierung, Subsystembildung).
- **Komponentendiagramme (component diagrams) und Kompositionsstrukturdiagramme (composite structure diagram):** beschreiben den strukturellen Aufbau auf höherer Abstraktion (Architektur).
- **Verteilungsdiagramme (deployment diagram):** zeigen die Verteilung eines Systems auf Hardware-Einheiten.

- **Anwendungsfalldiagramme (use case diagrams):** beschreiben aus Benutzersicht, was ein System leisten soll
- **Zustandsdiagramme (state machine diagrams):** beschreiben das zustandsabhängige Verhalten von Objekten
- **Aktivitätsdiagramme (activity diagrams):** dienen zur Beschreibung von Abläufen im System
- **Interaktionsdiagramme (interaction diagrams):** beschreiben die dynamischen Interaktionen und Abhängigkeiten der Systemelemente im Ablauf

5.8 OO Modellierung mit UML: Diagramme

- **Sequenzdiagramme (sequence diagrams):** zeigen exemplarisch die Interaktion einer Menge von Objekten.
- **Zeitdiagramme (timing diagrams):** mischen Zustands - und Sequenzdiagramme, um den Objektzustand über eine Zeitspanne hinweg darzustellen (mit den zustandsverändernden Nachrichten).
- **Interaktionsübersichtsdiagramme (interaction overview diagrams):** sind Aktivitätsdiagramme, in dem Teilabläufe durch referenzierte oder eingebettete Sequenzdiagramme repräsentiert sind (Verknüpfung von Sequenzdiagrammen mit Hilfe eines umgebenden Aktivitätsdiagramms; in der Praxis eher selten).
- **Kommunikationsdiagramme (communication diagrams):** zeigen Kommunikations-/Kollaborationsbeziehungen zwischen Objekten zur Laufzeit.

Wiederholung:

OO Grundlagen

UML Klassendiagramme

UML Sequenzdiagramme (erweitert)

UML Paketdiagramme (erweitert)

5.8 OO Modellierung mit UML: OO Wurzeln

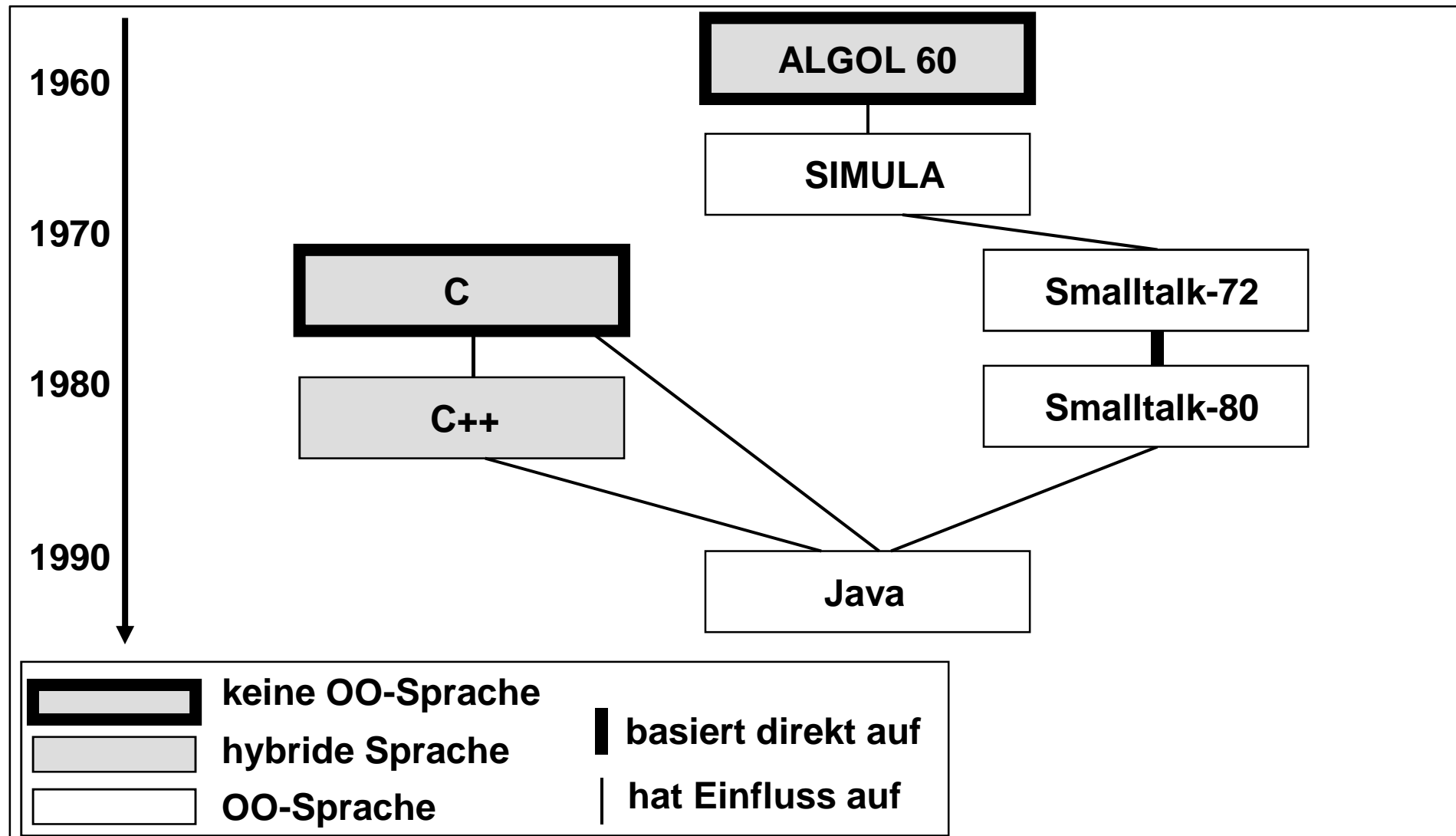
Programmiersprachliche Wurzeln objektorientierter Programmiersprachen:

- **Klassenkonzept von SIMULA 67**
- **Erste objektorientierte Programmiersprache Smalltalk-80 (Xerox)**
- **Objektorientierte Erweiterung von C zu C++ (Bell Labs)**
- **Weitere objektorientierte Sprachen Java (Sun), C# (Microsoft)**

Fortschritte in der Softwaretechnik (begleitend):

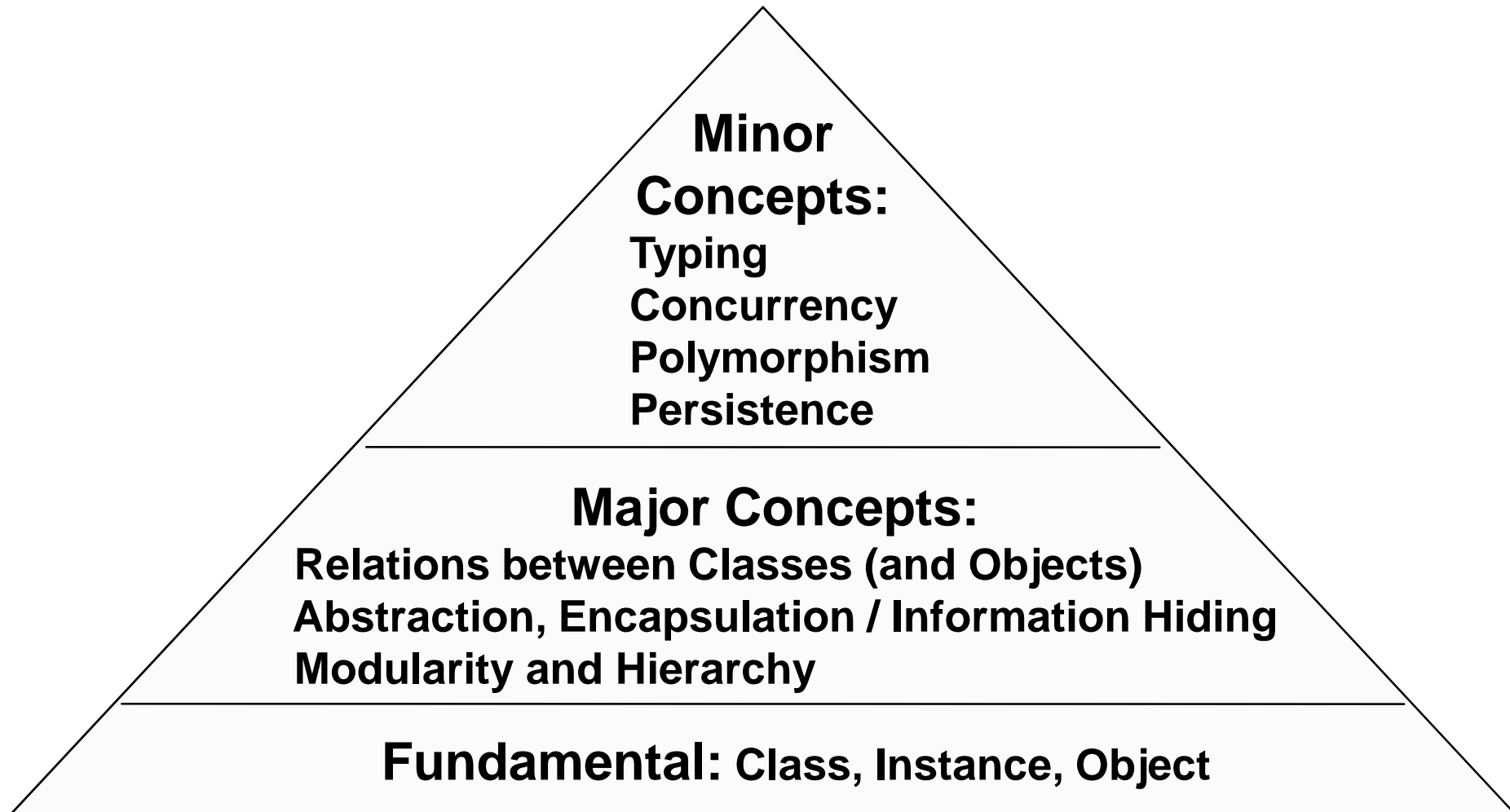
- **Modularisierungskonzepte und Abstraktionskonzepte**
- **Phasenübergreifende Entwurfsmethodik**
- **Objektorientierte Softwareentwicklung/-programmierung (OOP)**
- **Objektorientierte Analyse (OOA) & objektorientierter Entwurf (OOD)**

Programmiersprachliche Wurzeln objektorientierter Programmiersprachen:



5.8 OO Modellierung mit UML: OO Grundbegriffe

Überblick über die Grundbegriffe der Objektorientierung



[Boo94] G. Booch. *Object-Oriented Analysis and Design, 2nd. Edition*, Benjamin/Cummings, Redwood City, CA, 1994

5.8 OO Modellierung mit UML: OO Grundbegriffe

Objekt: Ein Objekt (Exemplar, Instanz) ist eine Ausprägung eines Dings der realen oder gedanklichen Welt: sichtbar oder greifbar oder intellektuell verständlich, Gegenstand der gedanklichen Betrachtung oder physischen Handhabung

Objekte haben einen Zustand:

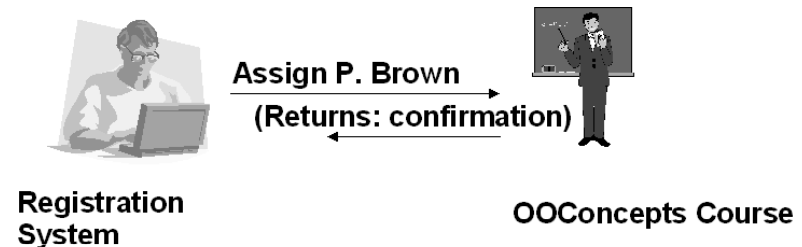
Programmiersprache:
Attribute mit Attributwerten



Name	P. Brown
Employee ID	5999783
Date hired	Oct 18, 2006
Status	Teaching

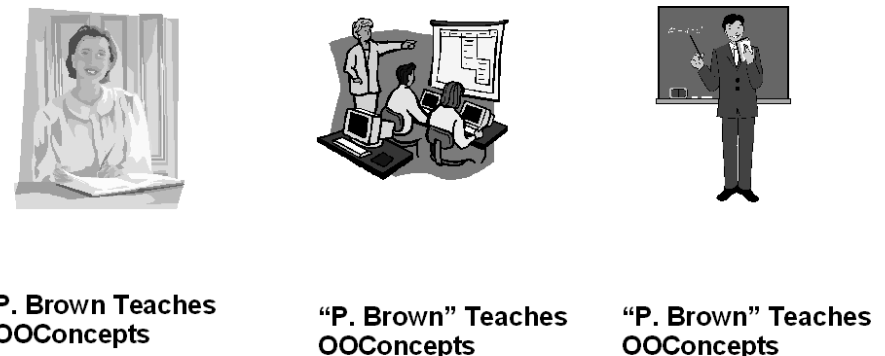
Objekte haben ein Verhalten:

Programmiersprache:
Menge der ausführbaren Methoden

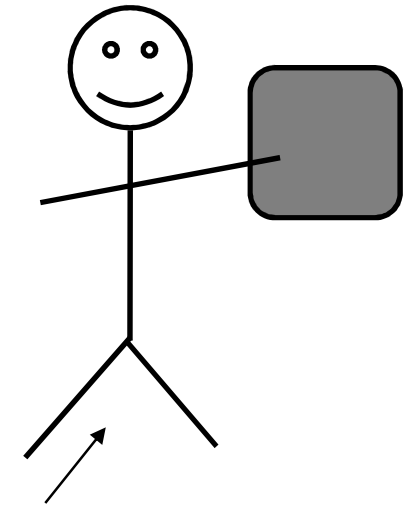


Objekte haben eine Identität:
Programmiersprache:
Identität wird in der Regel intern verwaltet
Achtung:

1. Operationen können die Identität nicht ändern
2. Identität \neq Gleichheit



Klasse:

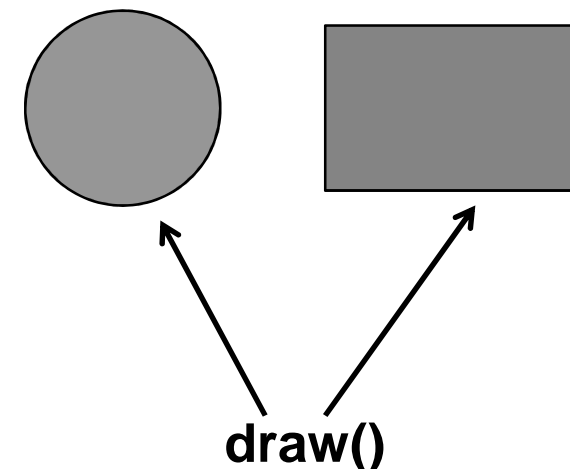


- **Klassen sind Schablonen / Baupläne zur Erzeugung einzelner konkreter Ausprägungen**
- **Klassen sind Gruppen von Objekten mit**
 - gemeinsamen Eigenschaften (Attributen),
 - gemeinsamem Verhalten (Operationen),
 - gemeinsamen Beziehungen
- **Klassen sind Klassifikationen von Objekten**
⇒ **Klassen haben keine Identität**

5.8 OO Modellierung mit UML: OO Grundbegriffe

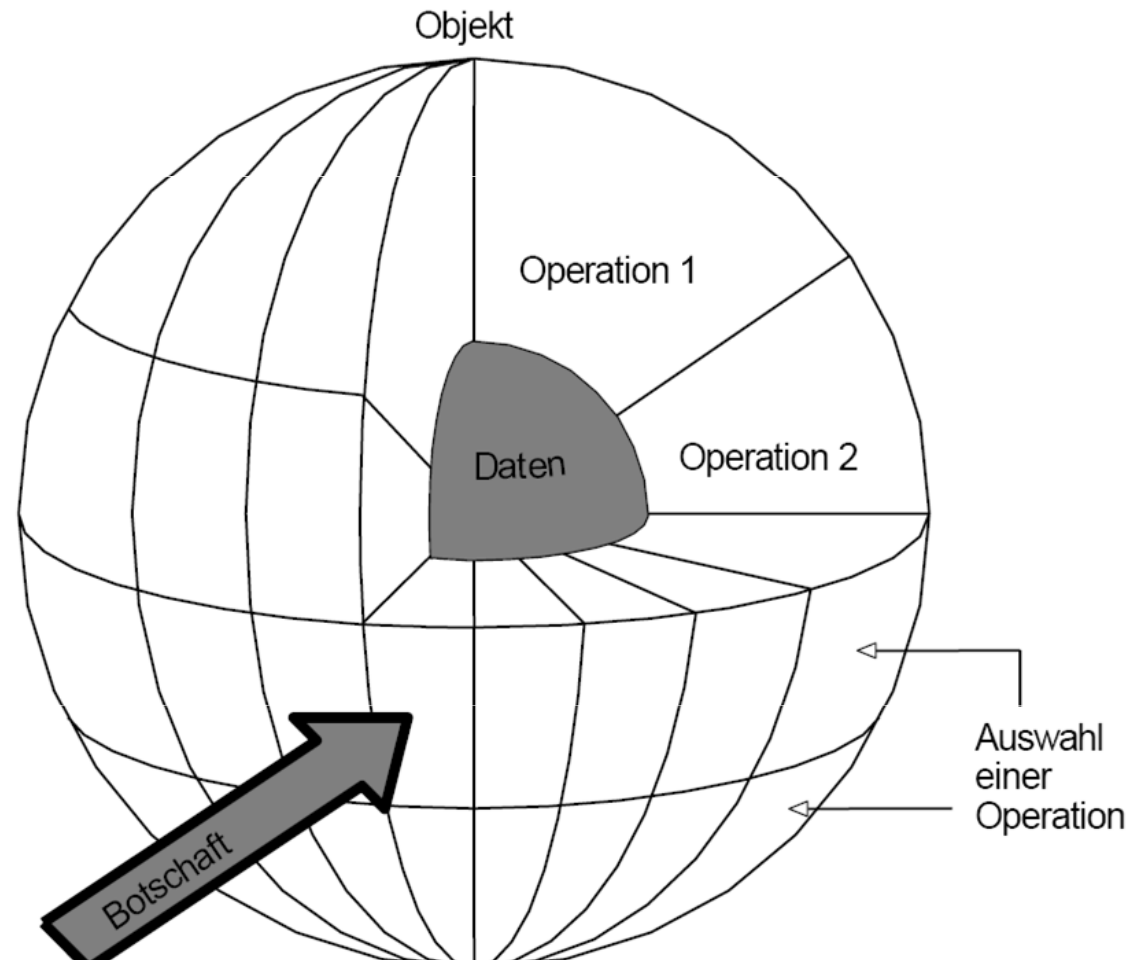
Polymorphie, Polymorphismus

- **Polymorph = Vielgestaltigkeit (griech.):**
Eigenschaft [eines Bezeichners], sich in Abhängigkeit von der Umgebung, in der er verwendet wird, unterschiedlich darzustellen (Gegenteil: Monomorphie)
- Eine Variable bzw. eine Methode kann gleichzeitig mehrere Typen haben (z.B. durch das Substitutionsprinzip)
- Eine Form der Polymorphie:
verschiedene Objekte können auf die gleiche Nachricht unterschiedlich reagieren



- **Kapselung (Encapsulation):** Zusammenfassung von Operationen und Attributen zu einer Einheit (eigentlich: Attribute, Operationen, Zusicherungen, Beziehungen)
- **Geheimnisprinzip (Information Hiding Principle):** bewusstes Verbergen von Implementierungsdetails; nach außen ist nur die Schnittstelle sichtbar

Auf die Attributwerte eines Objekts kann nur mit den Operationen des Objekts lesend oder schreibend zugegriffen werden (Geheimnisprinzip, Information Hiding).



Ausführungsmodell:

Ein Objekt reagiert auf den Empfang einer Nachricht mit der Ausführung der zugehörigen Operation und sendet das Ergebnis zurück

Entwurfsprinzipien:

- **Kohärenzprinzip bzw. Kohäsionsprinzip: jede Klasse soll für genau einen (sachlogischen) Aspekt des Gesamtsystems verantwortlich sein**
- **Objekt-Verantwortlichkeitsprinzip: jedes Objekt – und nur dieses – ist für seine Eigenschaften selbst verantwortlich**
- **Nachrichtenaustauschprinzip: Objekte sind eigenständige Einheiten, die durch Zusendung von Nachrichten miteinander interagieren**

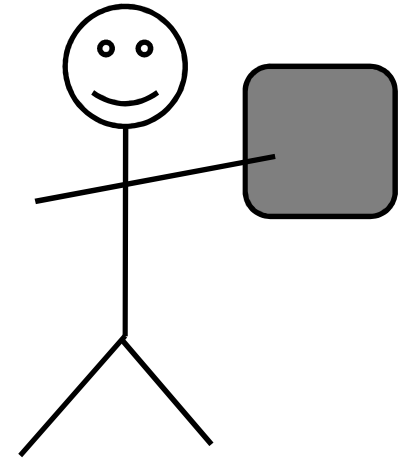
5.8 OO Modellierung mit UML: Klassendiagramm

UML Klassendiagramme:

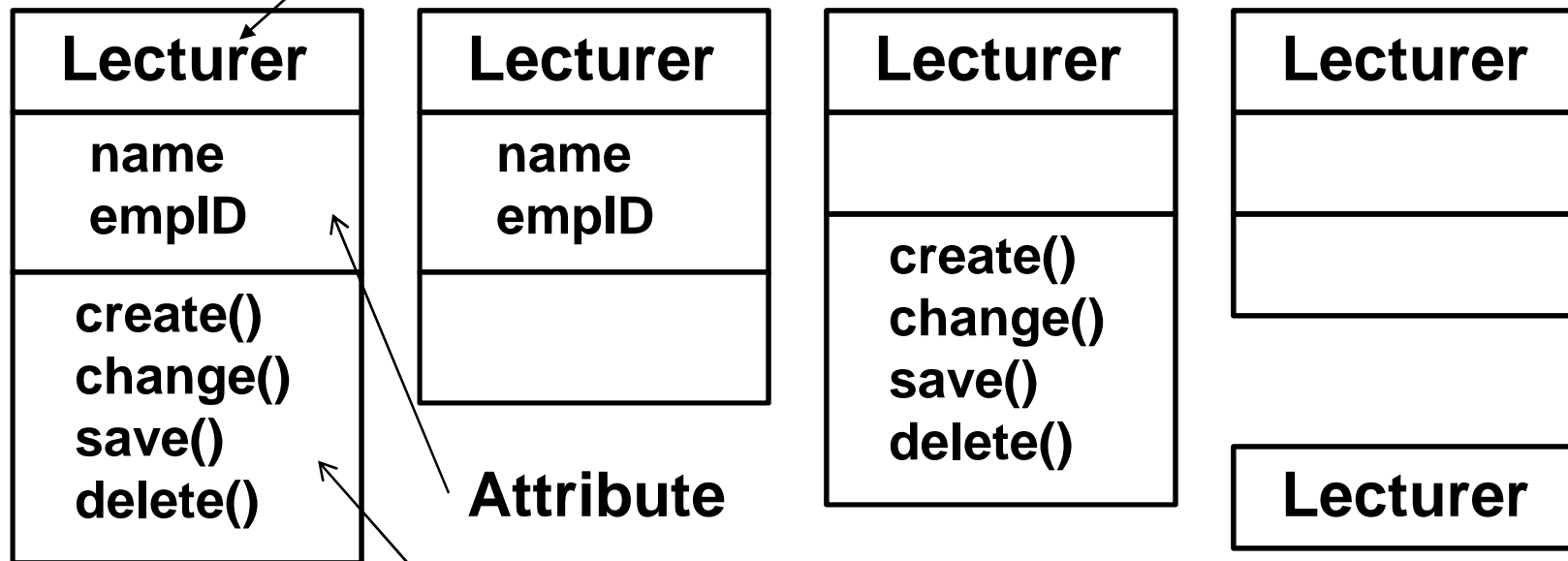
Representation von Klassen in UML als

Rechtecke mit den Abteilungen:

Klassenname, Attribute, Methoden (Operationen)



Klassenname
(modellweit bzw. paketweit eindeutig)

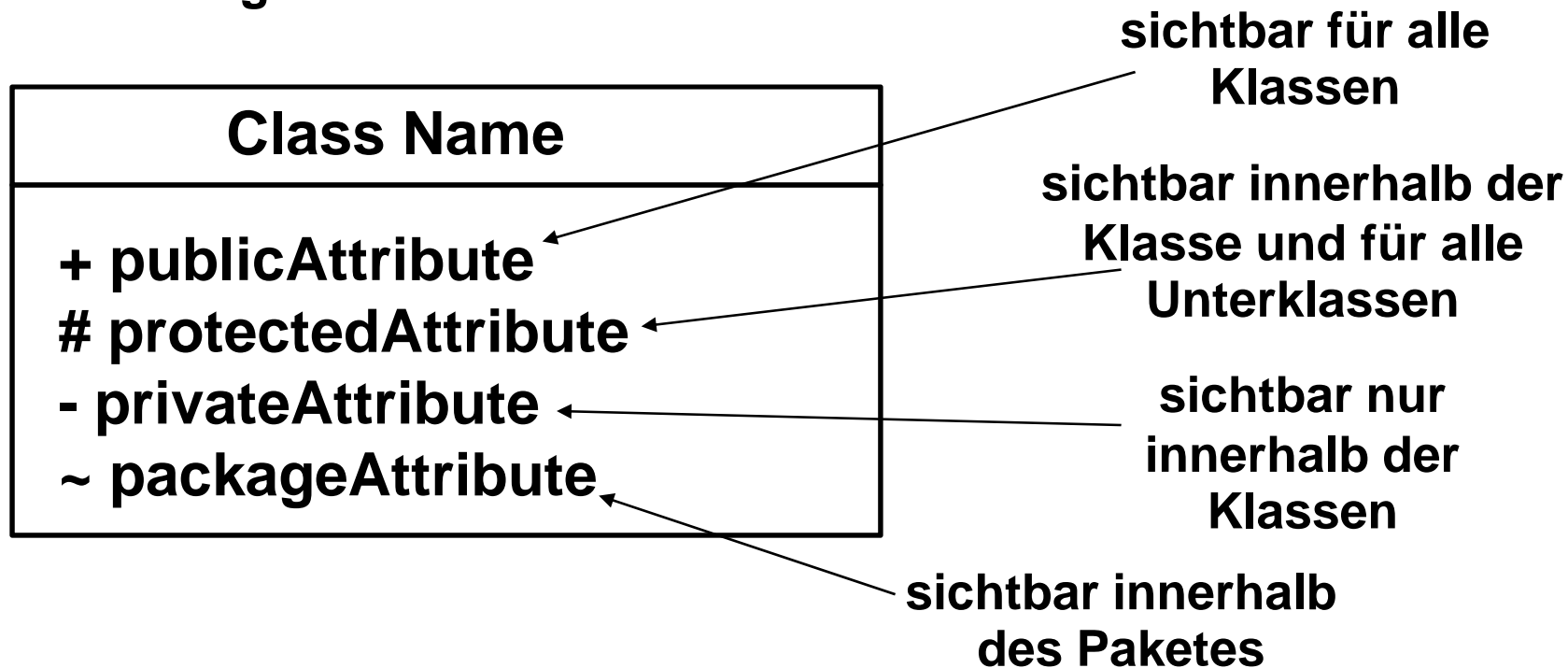


Attribute

Methoden (→ Verhalten)

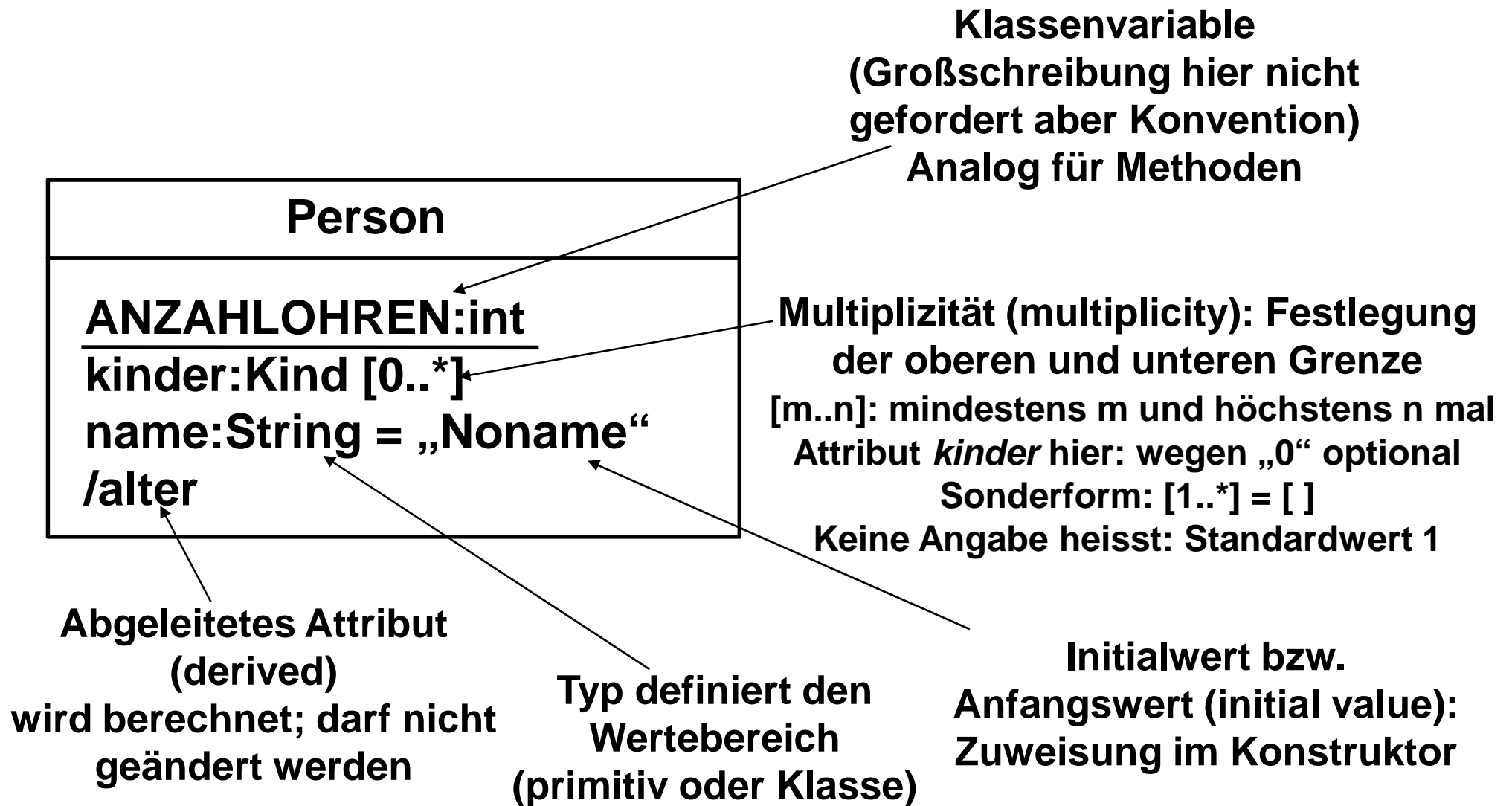
5.8 OO Modellierung mit UML: Klassendiagramm

Sichtbarkeit: Definition welche Klassen und Objekte auf das Element zugreifen können



- **Attributname muss eindeutig in der Menge aller Attribute gewählt sein**
- **Groß-/Kleinschreibung als Konvention**
- **analog für Methoden**

5.8 OO Modellierung mit UML: Klassendiagramm



- **Klassenattribut:** beschreibt einen Attributwert, der allen Objekten dieser Klasse bekannt ist
- **Beispiele für Klassenattribute für Klasse Student:**
 - Anzahl der Studenten,
 - nächste freie Matrikelnummer,
 - durchschnittliche Studiendauer
- **Klassenoperation:**
 - manipuliert die Klassenattributwerte
 - kann auch ohne Instanz verwendet werden
 - kann keine Instanzmethoden aufrufen
 - kann keine Instanzvariablen manipulieren

Beispiel Attributdeklarationen:

Person		
public	name	: String {readOnly}
public	geburtsdatum	: Date
private	gelesen	: Buch[0..*]
private	anschrift	: String
private	alter	: int = 0
public	alterKinder	: int[0..*]
public	verheiratet	: boolean = false

Beispiel Operationsdeklarationen:

Taschenrechner			
public	clear()		
public	add(in summand1:int,in summand2:int)		:int
private	div(in dividend: int,in divisor:int)		:double
public	inc(inout wert:int)		

5.8 OO Modellierung mit UML: Klassendiagramm

Attributdeklaration / Attributvereinbarung (allgemeiner Aufbau in UML):

Attributdeklaration = [Sichtbarkeit] + Name + [':' Typ] + [Multiplizität]
+ [= Anfangswert] + ['{' + {Eigenschaftswert} + '}']

Operationsdeklaration / Vereinbarung einer Operation (allgemeiner Aufbau in UML):

Operationsdeklaration = [Sichtbarkeit] + Name + '('Parameterliste')'
+ [':' + Rückgabetyt] + ['{' + {Eigenschaftswert} + '}']

Eine Parameterliste ist aus einer beliebigen Anzahl von Parametern aufgebaut:

Parameterliste = {Parameter}
Parameter = [Übergaberichtung] + Name + ':' + Typ + [Multiplizität]
+ ['=' + Anfangswert] + ['{' + {Eigenschaftswert} + '}']

Übergaberichtung:

- Die Übergaberichtung einer Operation spezifiziert die Richtung, in die ein Parameter übergeben wird.
 - in: Parameter wird nur gelesen.
 - out: Parameter wird nur schreibend verwendet.
 - inout: Parameter wird gelesen, verarbeitet und neu geschrieben.
- Eine Sonderform ist der Rückgabeparameter, der das Ergebnis der Anwendung einer Operation an den Aufrufer zurückgibt.
- Nicht alle Programmiersprachen unterstützen die Angabe der Übergaberichtung für Methodenparameter explizit.

5.8 OO Modellierung mit UML: Klassendiagramm

Genauere Spezifikation z.B. zur Darstellung von Zusicherungen in UML (und anderen UML Erweiterungen für Operationen und Attribute)

- **Beispiel einer Einschränkung als Eigenschaftswert bzw. Property String (Zusicherung, die die Operation erfüllen muss)**

QuadWurzel
wurzelexponent:int = 2 {readOnly}
quadratWurzel():int {Returnwert > 0}

- **Andere Eigenschaftswerte, z.B.:**
{query}: Operation ändert keine Daten des Systems
{ordered}: Werte des Ergebnisparameters sind geordnet
{redefines}: Operation redefiniert eine geerbte Operation
- **Gleiches Prinzip für Eigenschaftswerte bei Attributen** (z.B. {readOnly}, {ordered} bzw. wenn beide erfüllt sein sollen durch Kommatrennung: {readOnly, ordered})

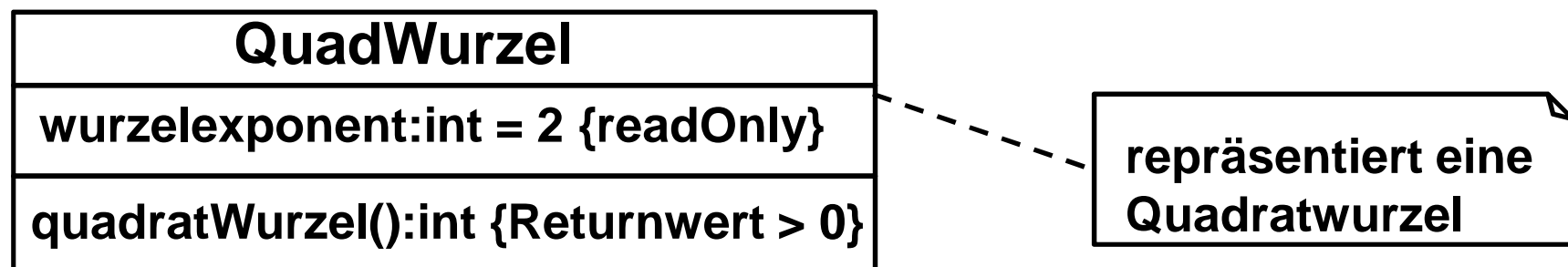
Ergänzungen, die in UML nicht spezifiziert sind, sind außerdem möglich mit:

- **Stereotypen** (z.B. <<interface>>)

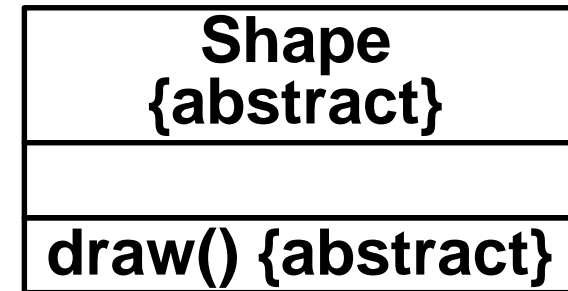
Neue Stereotypdefinitionen müssen auch semantisch definiert werden.

- **Annotation / Notiz / Kommentar**

(v.a. zur Dokumentation und ohne semantische Wirkung)



Eigenschaft „Abstrakt“: {abstract}



Abstrakte Klasse:

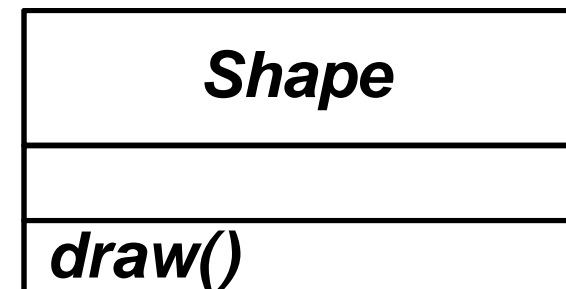
Klassen mit abstrakten Operationen.

Es kann keine Instanzen von abstrakten Klassen geben.

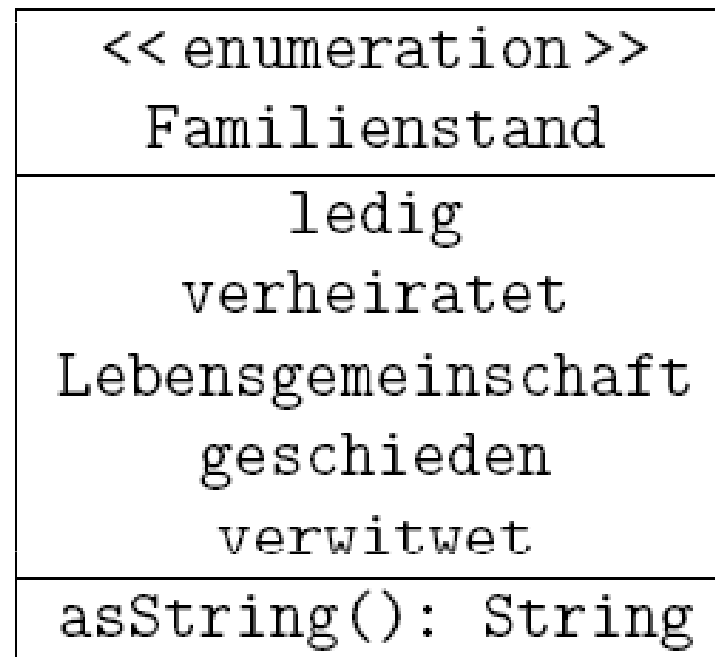
Abstrakte Operation:

Für eine abstrakte Operation ist nur die Bezeichnung der Operation (Signatur) vorhanden, aber kein Operationsrumpf.

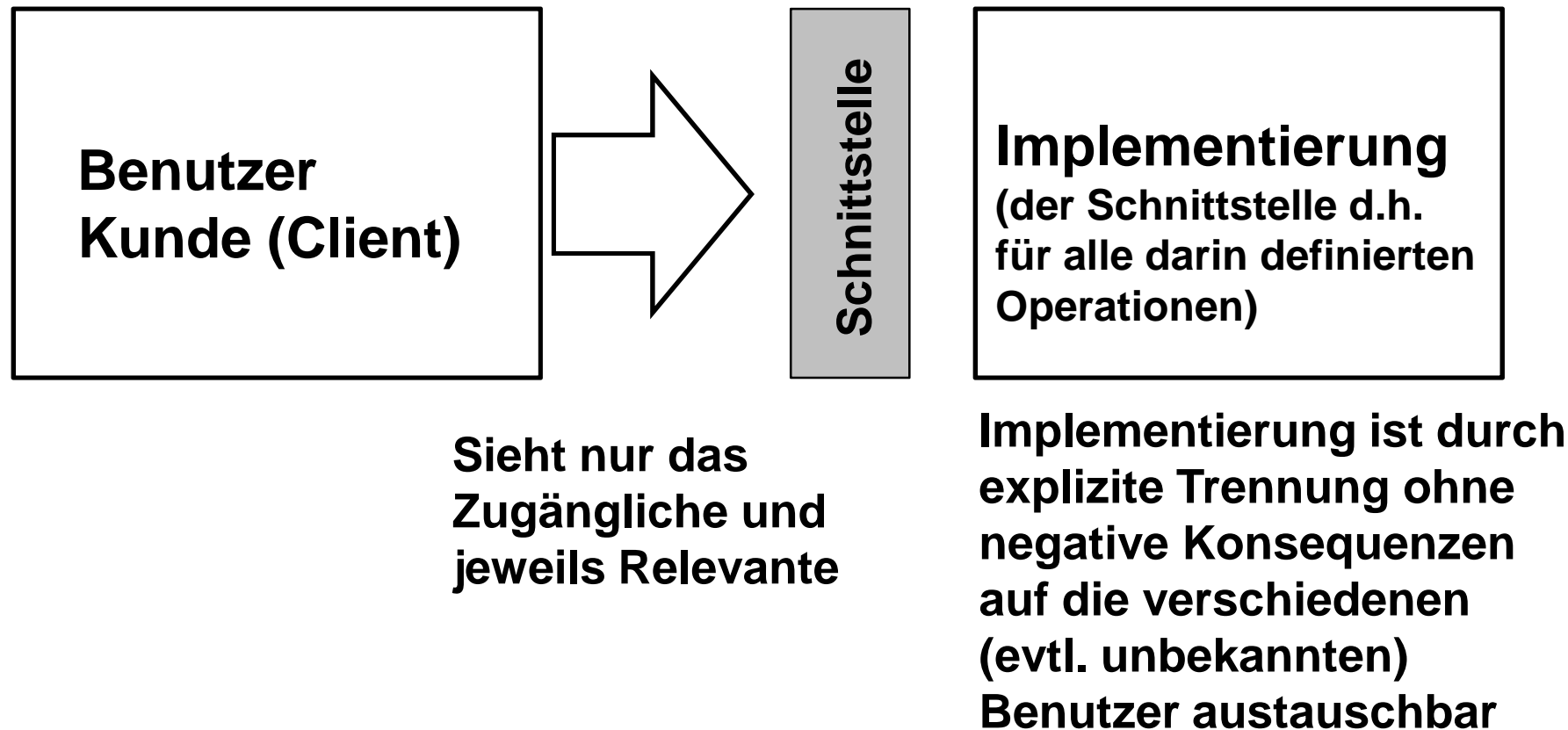
Alternative UML-Darstellung:
Kursiv-Schreibweise



Aufzählungstypen: Darstellung mit Hilfe von Stereotypen



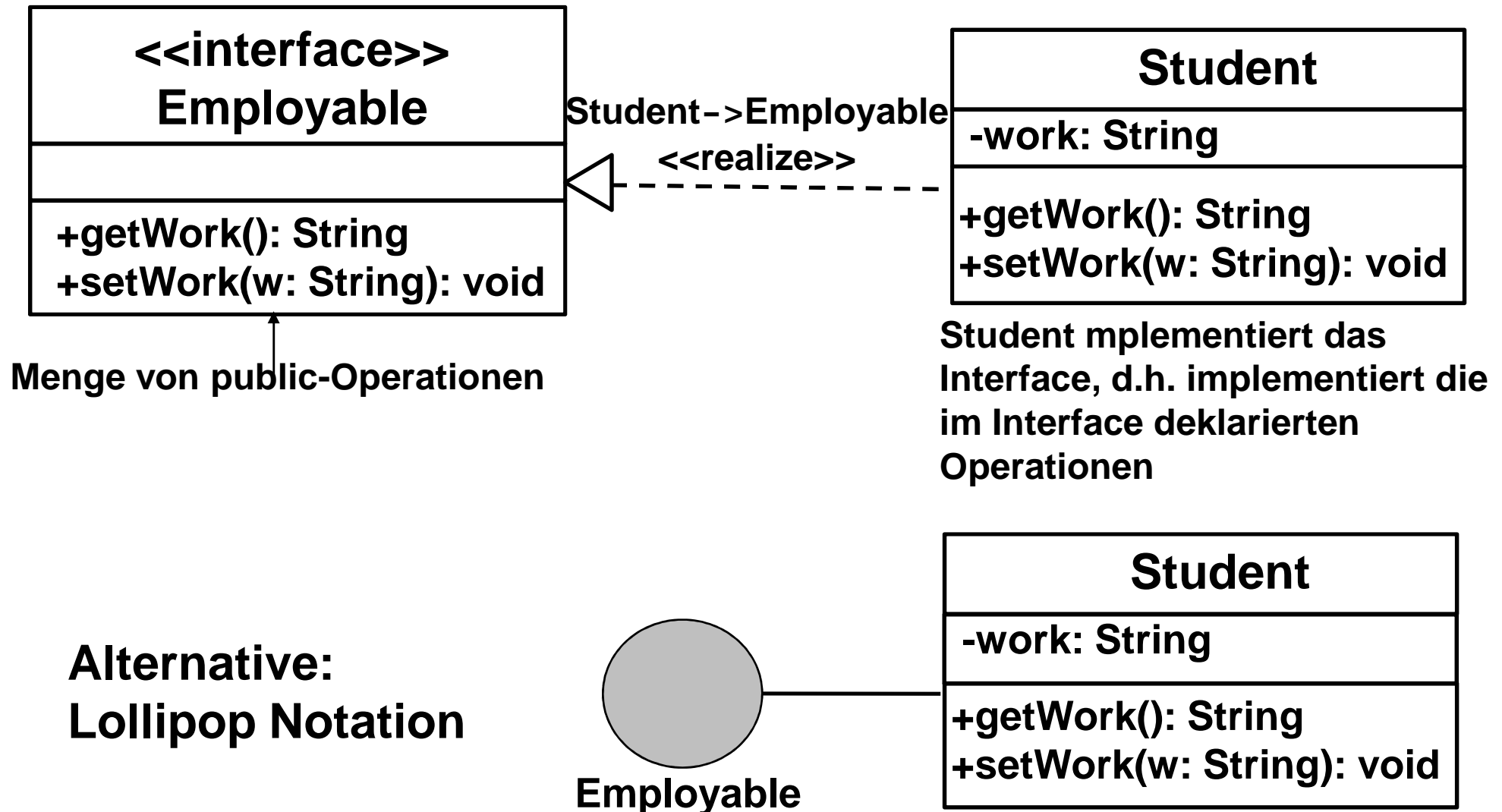
5.8 OO Modellierung mit UML: Klassendiagramm



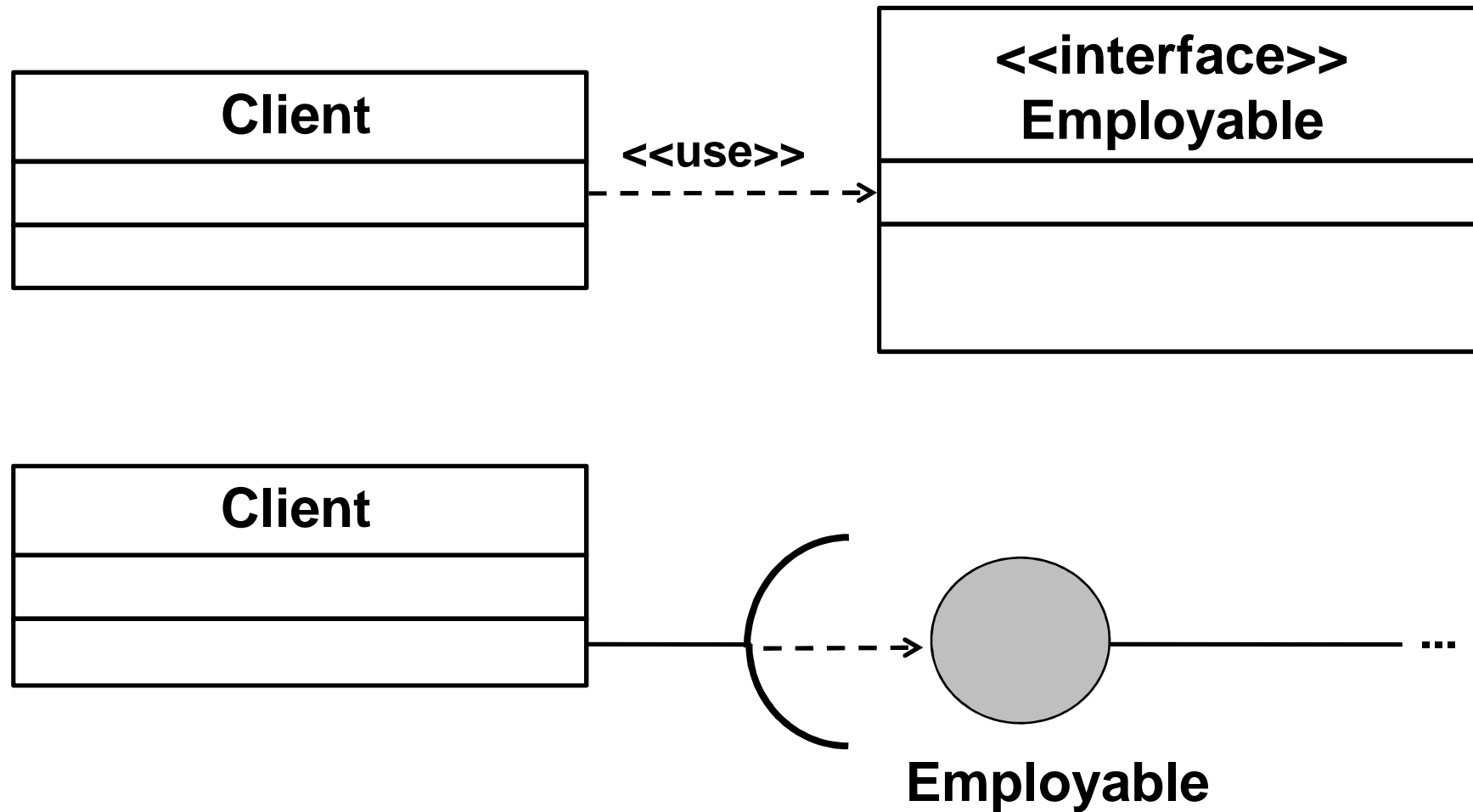
- Isolierte Entwicklung von Implementierungen möglich
- Anwendungen können nur auf Interfaces aufbauen (Implementierung später bzw. einfacher Austausch)
- Aufgabenaufteilung im Team
- **Bemerkung: Eine Klasse kann mehrere Schnittstellen implementieren (Rollen)**

Bemerkung:
Die Grafik oben
ist nicht in UML-
Notation!

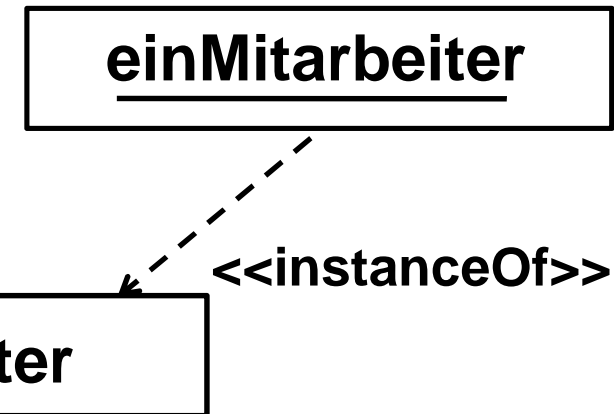
Darstellung einer Schnittstelle im UML Klassendiagramm:



Darstellung einer Schnittstelle in UML: Interface Nutzung



Beziehungen im UML Klassendiagramm



- Beziehungen zwischen Klassen und Instanzen (Instanziierungsbeziehungen)
- Beziehungen zwischen Klassen bzw. Beziehungen zwischen Instanzen
 - Abhängigkeit
 - Assoziation
 - Aggregation
 - Komposition (Composition)
 - Benutzt-Beziehung (Using / gerichtete Assoziation)
 - Vererbung
- Programmiersprachensicht vs. OO/UML-Sicht (nicht immer eine 1:1 Abbildung)

5.8 OO Modellierung mit UML: Klassendiagramm

Vererbungsbeziehung:

Eine Unterklasse übernimmt durch Vererbung alle (sichtbaren) Attribute und Operationen von ihrer direkten Oberklasse.

Die Unterklasse kann übernehmen und spezialisieren, aber nicht eliminieren.

Vererbungsstruktur:

Bei einer Einfachvererbung besitzt jede Klasse außer der Wurzel genau eine direkte Oberklasse -- die Vererbungsstruktur ist ein Baum.

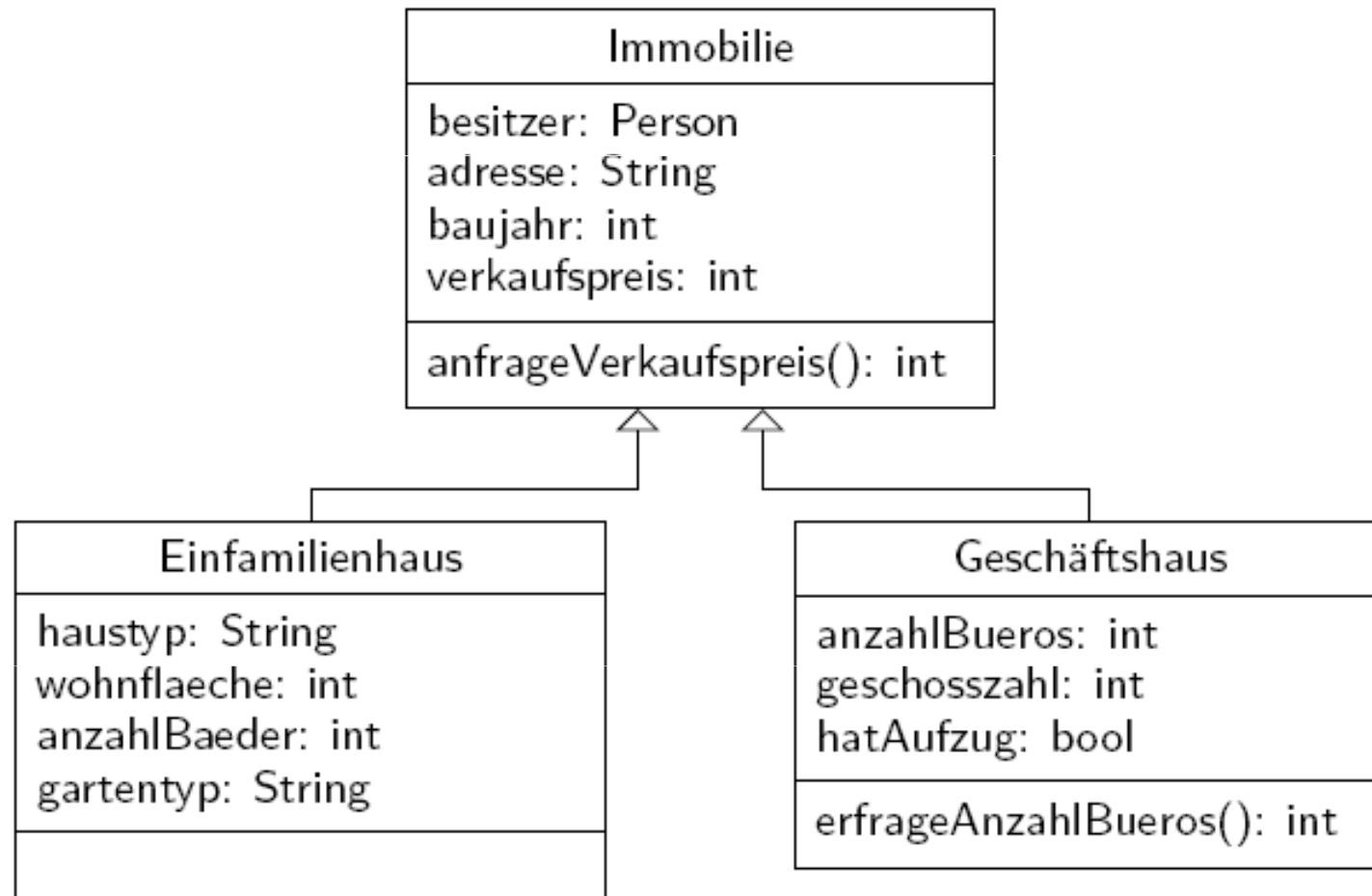
Bei einer Mehrfachvererbung kann eine Klasse auch mehrere direkte Oberklassen besitzen -- die Vererbungsstruktur ist ein gerichteter azyklischer Graph (dag).

Beispiel: Klassen ohne Vererbung

Einfamilienhaus
haustyp: String besitzer: Person adresse: String wohnflaeche: int anzahlBaeder: int gartentyp: String baujahr: int verkaufspreis: int
anfrageVerkaufspreis():int

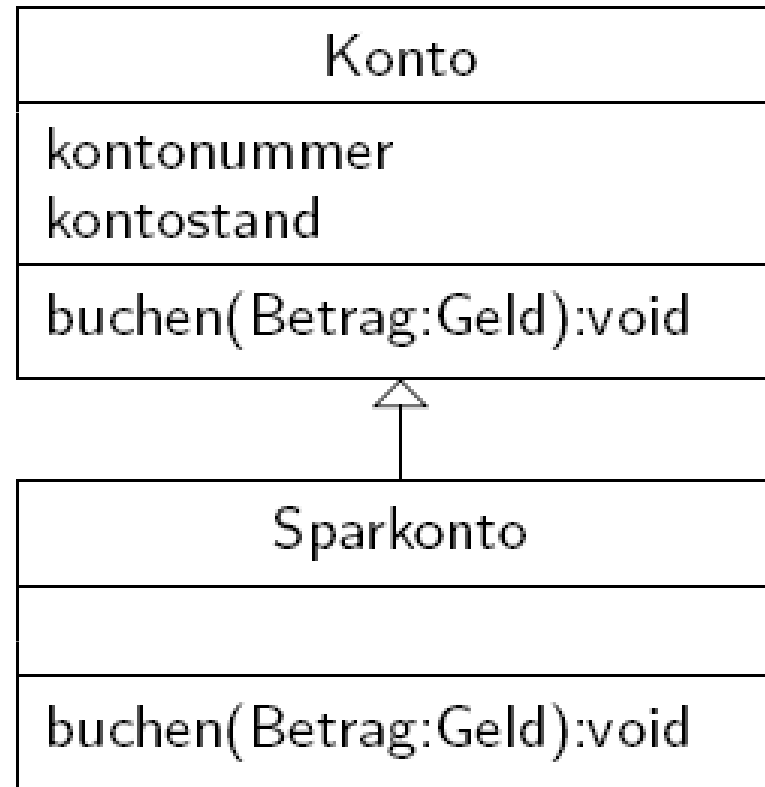
Geschäftshaus
besitzer: Person adresse: String anzahlBueros: int geschosszahl: int hatAufzug: bool baujahr: int verkaufspreis: int
anfrageVerkaufspreis(): int erfrageAnzahlBueros(): int

Beispiel mit Vererbung:



**Vererbung ist eine Möglichkeit, eine Beziehung
Generalisierung ↔ Spezialisierung auszudrücken**

Überschreiben von Operationen:

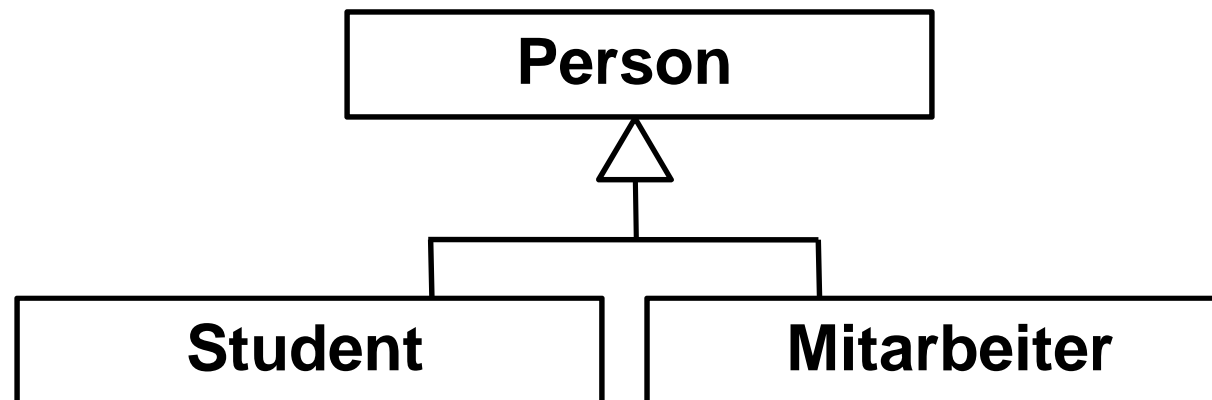


Eine Operation einer Unterklasse überschreibt eine Operation einer Oberklasse gleichen Namens. Bei dieser Umdefinition muss die Schnittstelle der Operation in der Unterklasse konform zur Schnittstelle der Operation in der Oberklasse sein (→ Substitutionsprinzip).

Vererbung im UML Klassendiagramm:

Vorgehensweise

Spezialisierung
↓

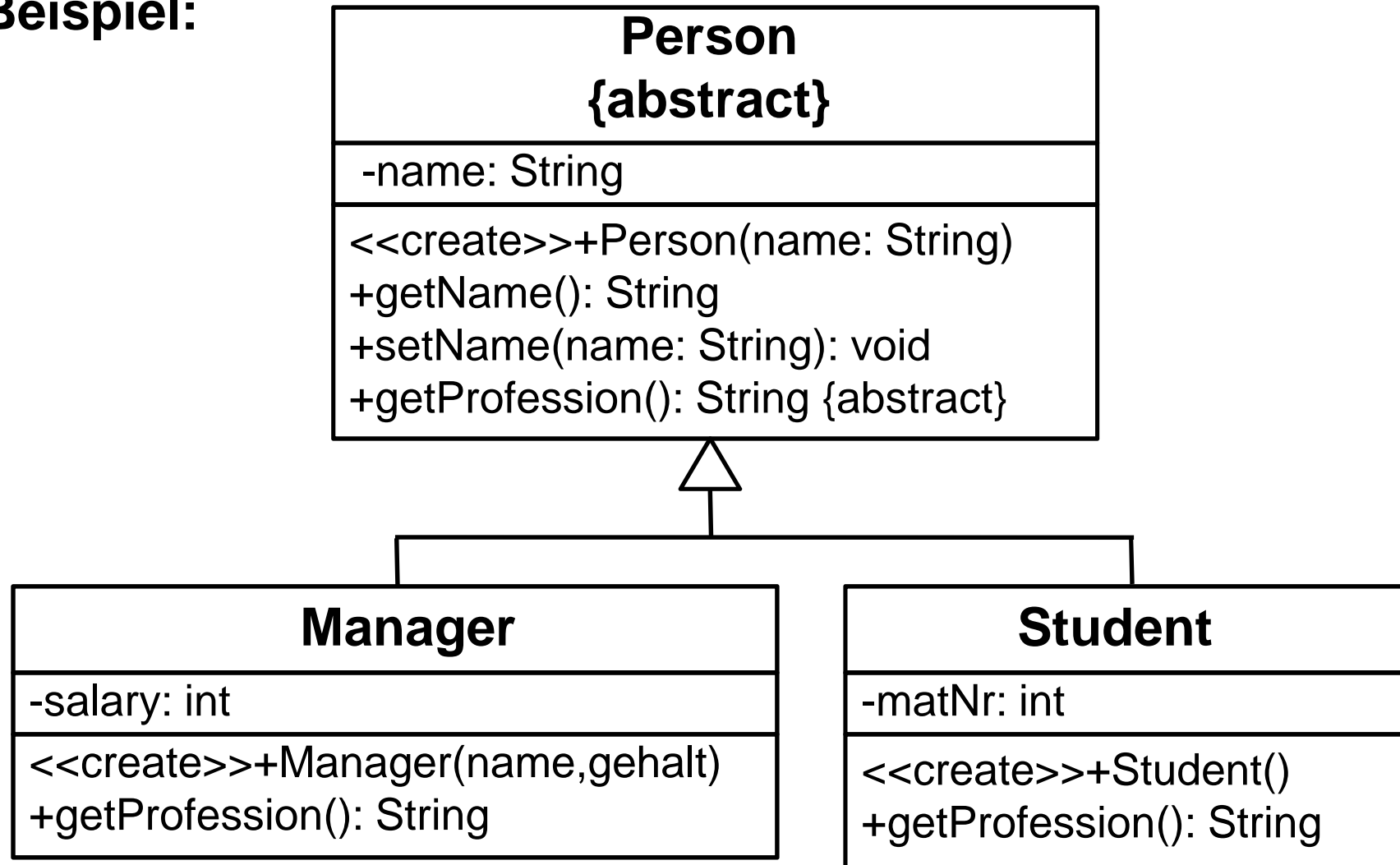


Elter
Oberklasse,
Superklasse,
Base Class

Generalisierung
(„is a“, „is kind of“)

Kinder
Unterklasse, Subklasse,
Derived/abgeleitete Klasse

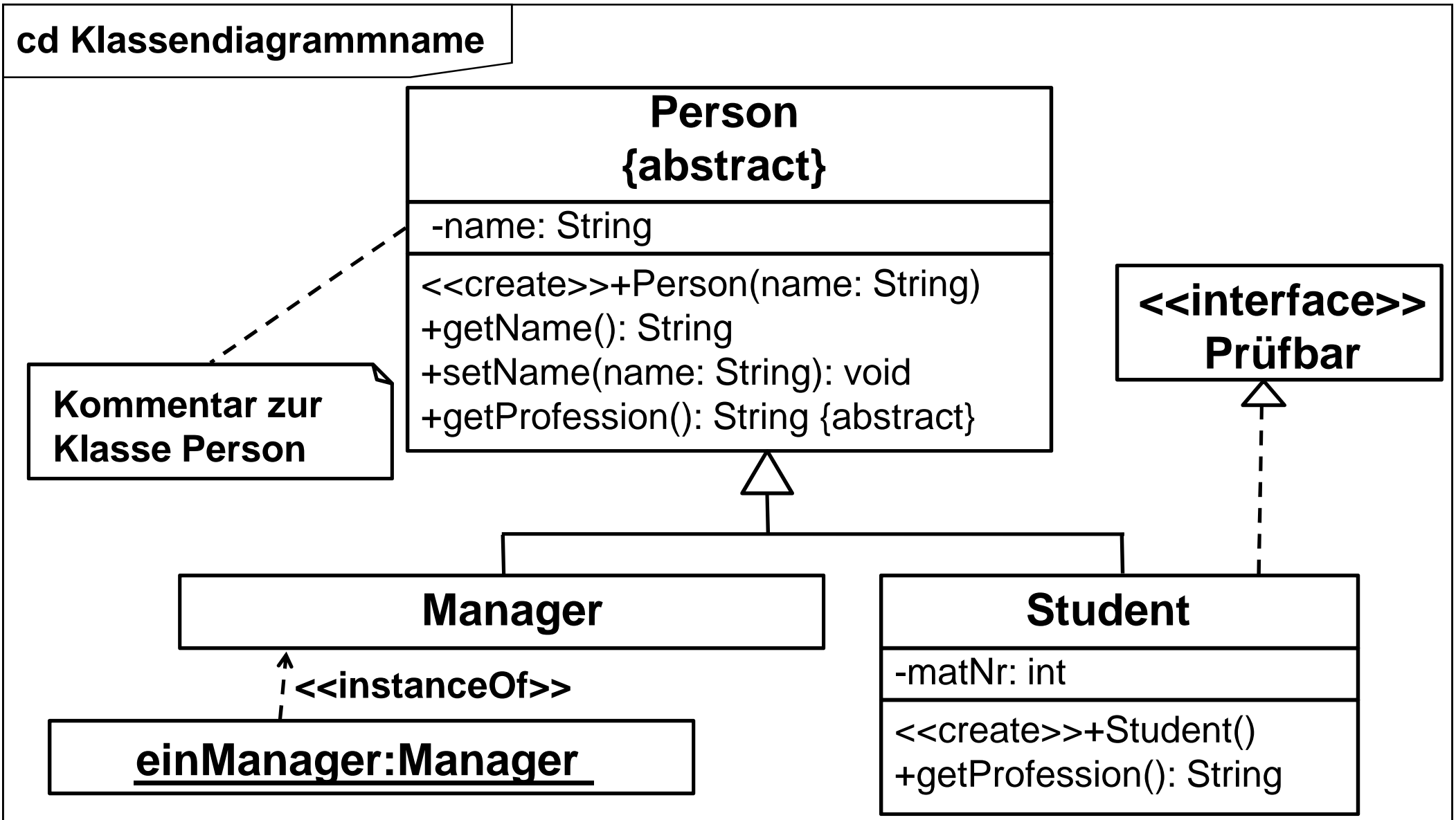
Beispiel:



Alternative Schreibweise zu {abstract}: Kursiv-Schreibweise
Aber: Kursiv-Schreibweise wird schnell übersehen

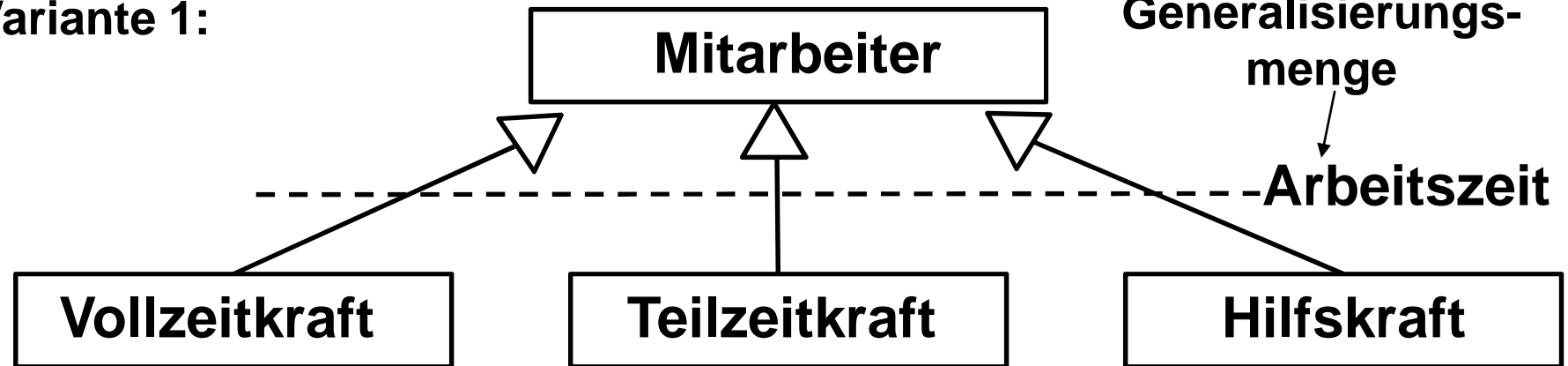
5.8 OO Modellierung mit UML: Klassendiagramm

Beispiel: struktureller Aufbau des Systems aus Klassen (mit Rahmen und Name)

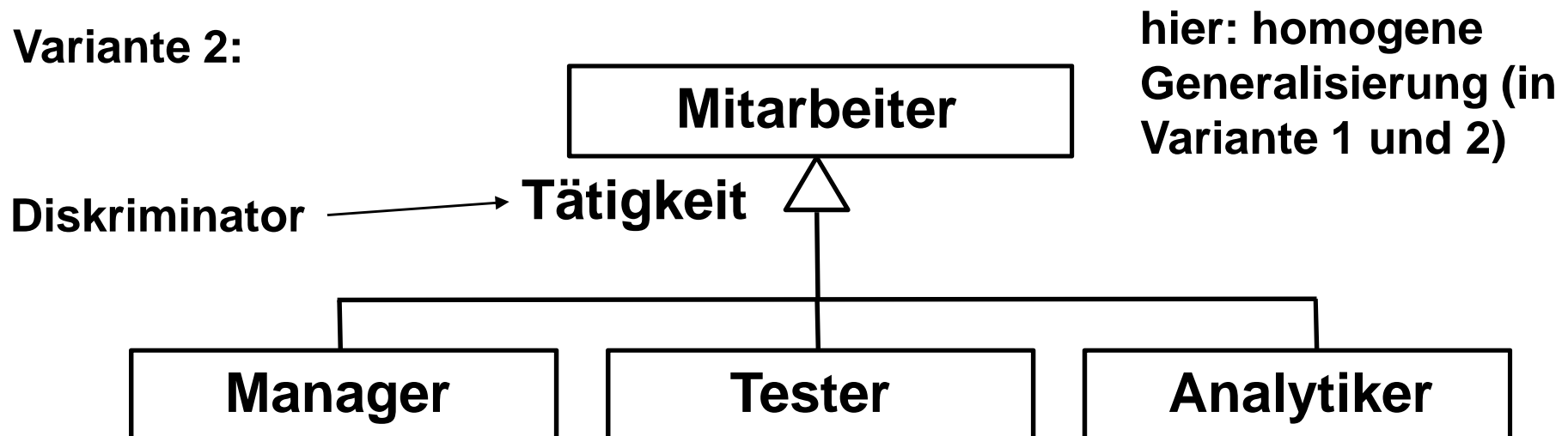


Vererbung und Diskriminator:

Variante 1:



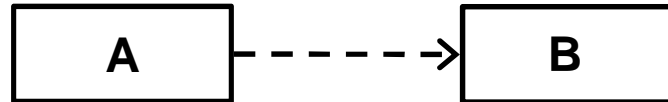
Variante 2:



Nutzen der Vererbung?

- **Modularisierung, Abstraktion, Wiederverwendung und Schnittstellenbildung (mittels abstrakter Vererbung)**
- **Aber: Keine expliziten Grenzen mehr (nichtlokale Abhängigkeiten, “Fragile Base Class Problem”)**
- **Alternativen der Vererbung: z.B. Delegation, Schnittstellen, generische Ansätze, generative Ansätze**
- **Vermeide Vererbung, wenn es Alternativen gibt**

5.8 OO Modellierung mit UML: Klassendiagramm

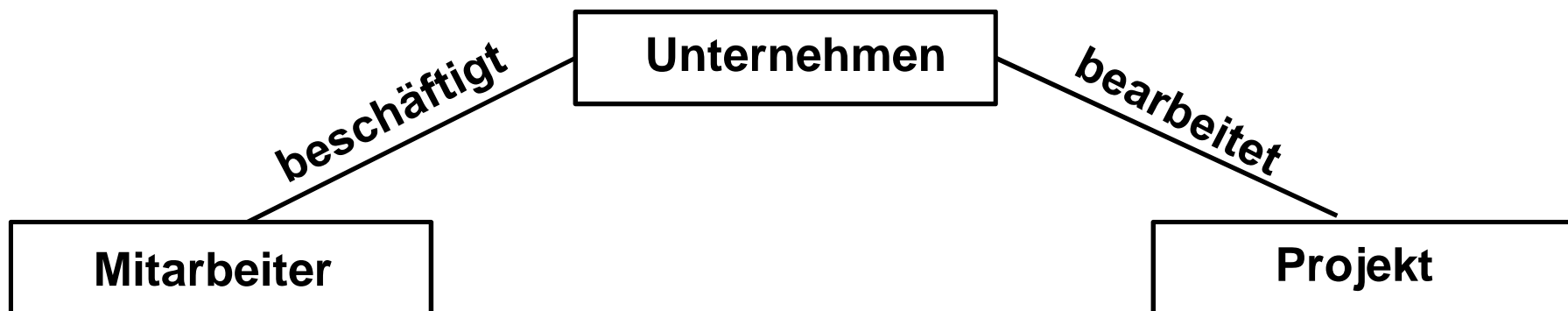


- **Abhängigkeitsbeziehung**
- **Englisch: Dependency (auch: Refinement, Realization)**
- **A ist abhängig von B (erst durch B wird A vollständig)**
- **Art der Abhängigkeit kann durch ein *Stereotyp* näher spezifiziert werden**
- **Abhängigkeit zunächst nicht näher festgelegt (sondern individuell)**
- **Beispiel:**
Abhängigkeit durch Nutzung eines Interfaces (z.B. mit Stereotyp <<use>>)

5.8 OO Modellierung mit UML: Klassendiagramm

Assoziation:

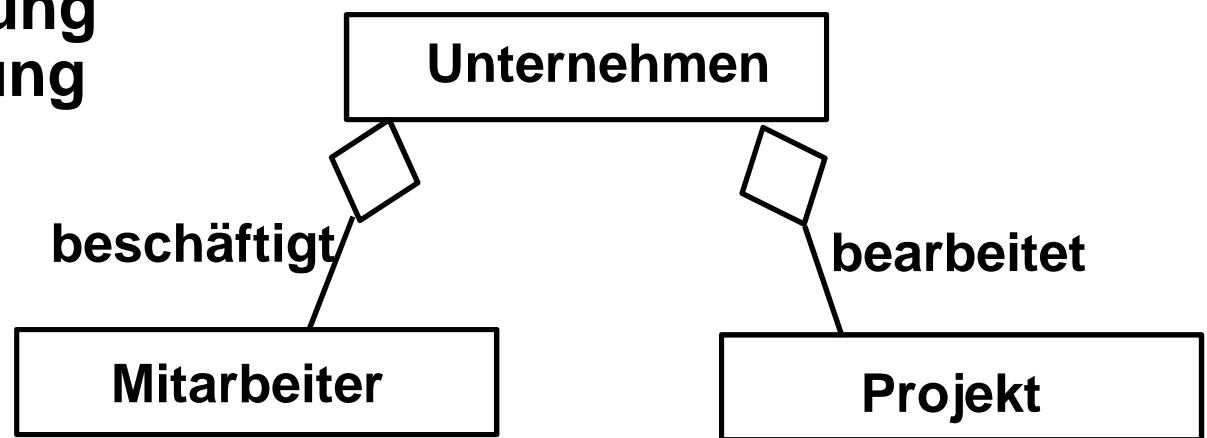
- Beziehung: für den Nachrichtenaustausch muss ein Objekt ein anderes kennen und kann so Anforderungen weitergeben
- Objektbeziehungen bzw. Objektverbindungen (engl. Link) sind Exemplare einer Assoziation (analog zu: "Objekte sind Exemplare einer Klasse")
- Eine Assoziation ist ein unspezifischer Beziehungstyp (keine Aussage über die genauere Realisierung)
- Wir wissen nur, dass es eine Art von Beziehung und Abhängigkeit gibt, aber (noch) nicht genau welche
- Typisch für frühes Stadium im SW-Entwicklungsprozess: die Beziehung kann später verfeinert werden



5.8 OO Modellierung mit UML: Klassendiagramm

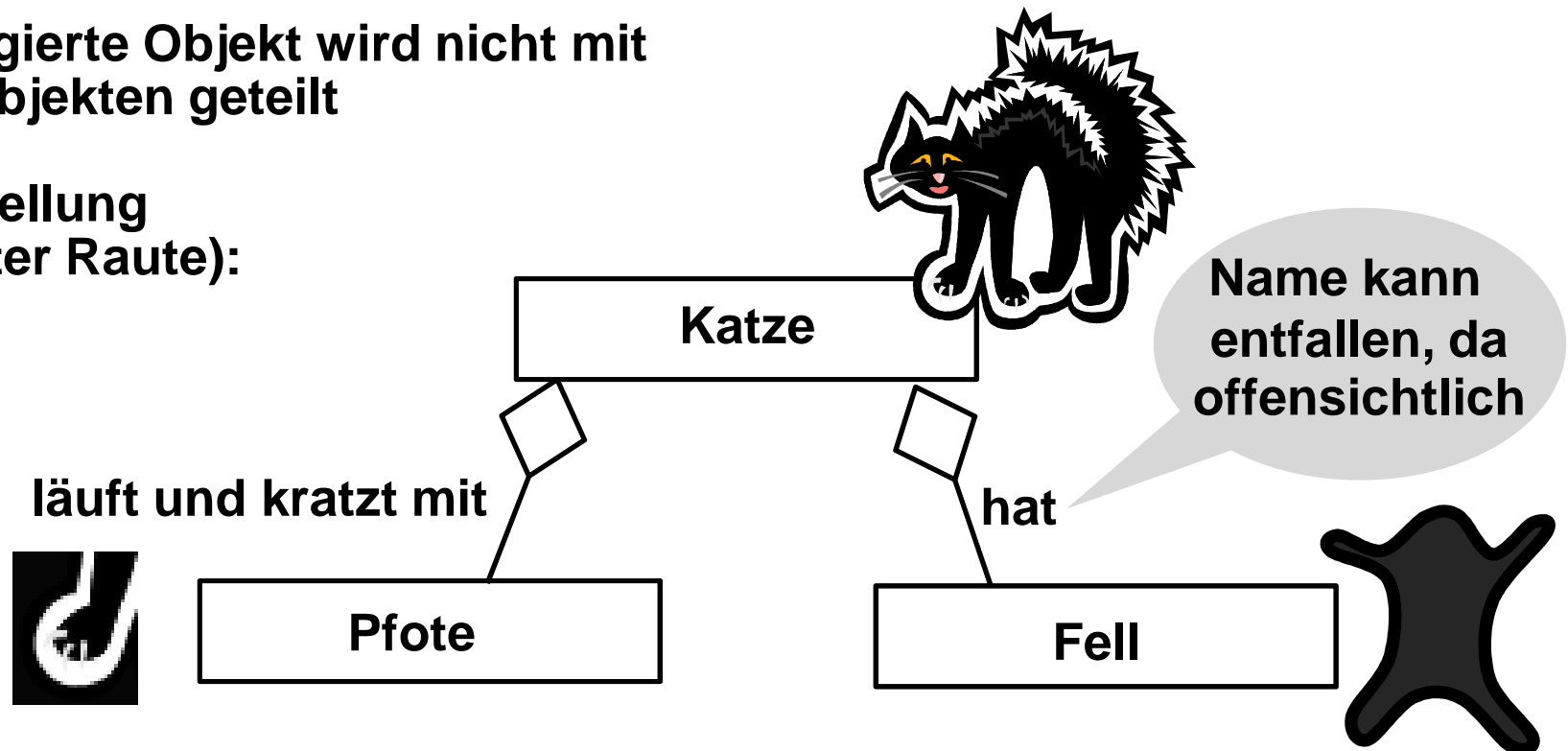
Aggregation (Name auch: Shared Aggregate):

- **Spezielle Assoziation**
- **Ein Objekt ist Teil eines anderen**
- **Ein Komposit (Composit) bzw. Ganzes (Whole) bzw. Aggregat besteht aus (evtl. mehreren) Komponenten (Components) bzw. Teilen (Parts)**
- **Auch: Whole/Part Beziehung bzw. Ganzes/Teil Beziehung**
- **UML Darstellung:**



Komposition (Name auch: Composite Aggregate)

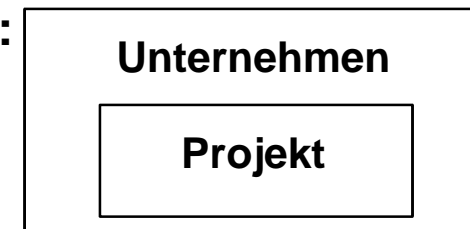
- Spezielle (stärkere) Form einer Aggregation
- “Strong Ownership”: die Teile können nicht das Ganze überleben (Übereinstimmung der Lebensdauern, existenzabhängige Teile)
- Das aggregierte Objekt wird nicht mit anderen Objekten geteilt
- UML Darstellung (mit gefüllter Raute):



5.8 OO Modellierung mit UML: Klassendiagramm

- Zu einem Zeitpunkt kann ein Teil nur einem Aggregatobjekt zugeordnet sein (und von diesem existenzabhängig sein)
- Das Ganze (Whole) erzeugt (meist bei dessen Erzeugung) auch seine Teile
- Wird das Ganze gelöscht, werden automatisch auch seine Teile gelöscht
- Aber in UML gilt auch:
Teile dürfen (z.B. vor einer Löschung) einem anderen Aggregat-Objekt zugeordnet werden / verantwortlich übergeben werden

Alternative Darstellung einer
Komposition:



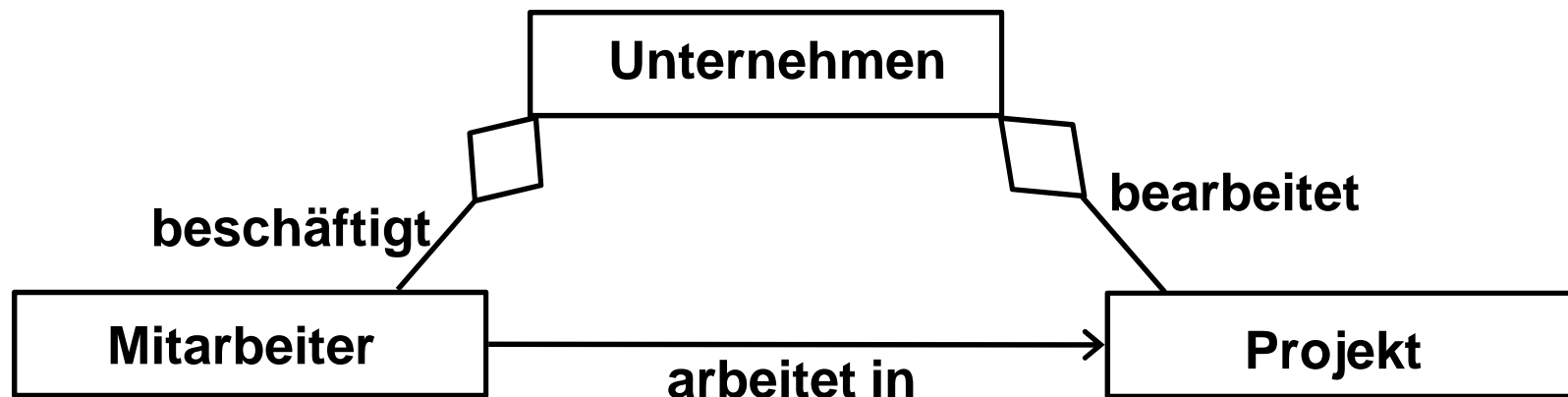
Umsetzung der Aggregation und Komposition

- Das Komposit besitzt eine Referenz auf die Komponentenobjekte
- Das Komposit initiiert die Instanziierung und Zerstörung der referenzierten Komponentenobjekte (Kontrolle der Lebensdauern der Komponenten durch das Komposit)

```
public class Katze {  
    private Pfote meineRVorderPfote;  
    private Fell meinFell;  
  
    public Katze() {  
        meineRVorderPfote = new Pfote();  
        meinFell = new Fell();  
    }  
  
    protected finalize() {  
        // without garbage collection  
        // do delete in the destructor  
    }  
}
```



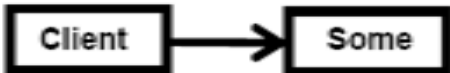
Benutzt (Using):

- Ähnlich zur Aggregationsbeziehung, aber:
- Keine Instanziierung der Komponente
- Keine Berechtigung, die Komponente zu zerstören
- Beispiel: Beziehung zwischen Mitarbeiter und Projekt


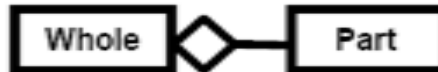
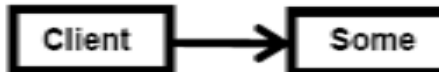


- Mitarbeiter muss vom Unternehmen eine Referenz auf das Projekt bekommen

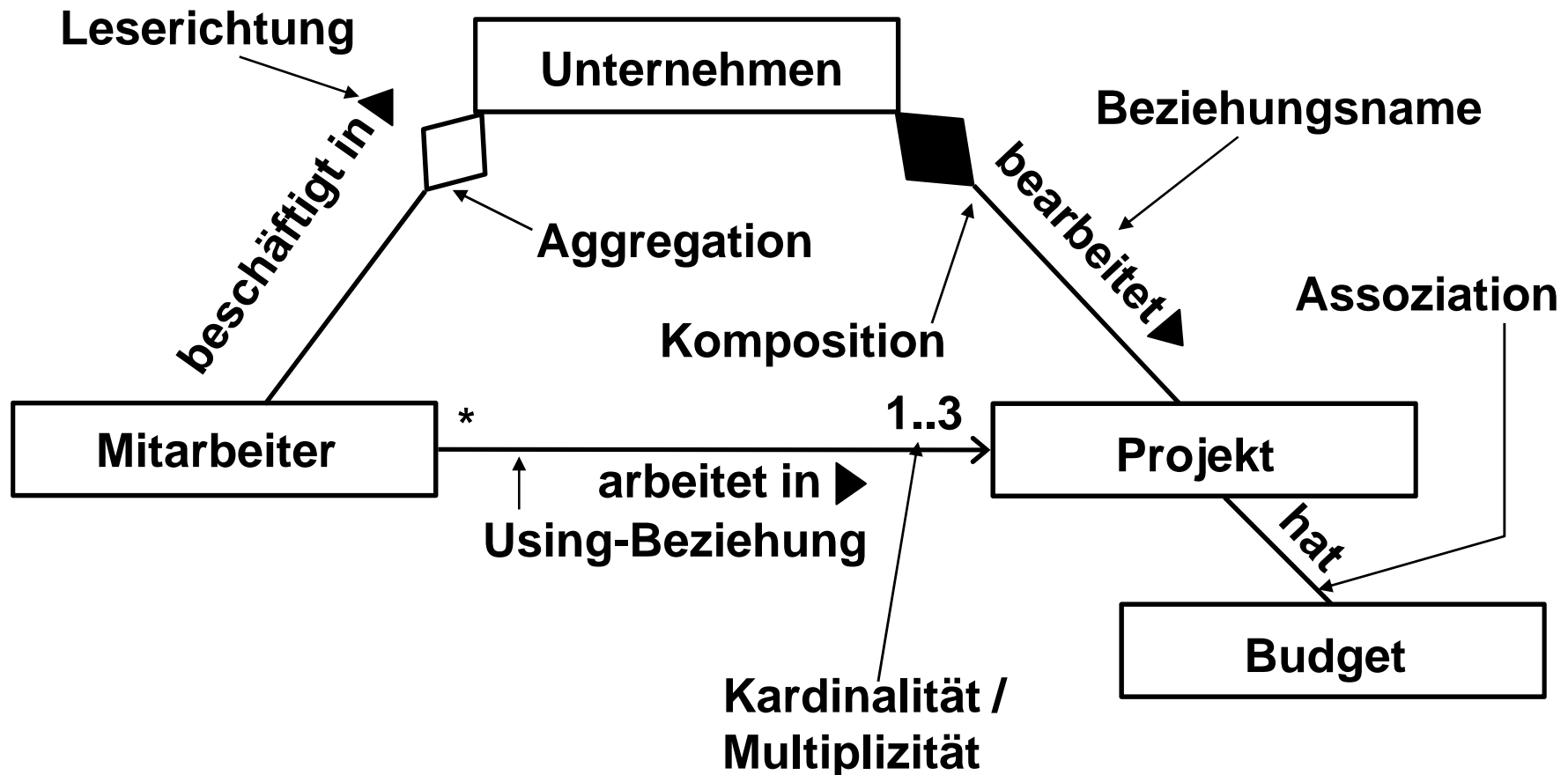
5.8 OO Modellierung mit UML: Klassendiagramm

Kriterium	Komposition	Aggregation	Using
UML			
Besitzt / Hat (mit Hat-Beziehung \subseteq Besitzt- Beziehung)	Whole besitzt Part	Whole hat Part	Client hat nur Zugriff auf Some
Ort der Erzeugung der Part-Instanz bzw. Some-Instanz	Im Whole	Im Whole, evtl. auch außerhalb und Übergabe an Whole	Nur außerhalb von Client
Zeitpunkt der Erzeugung der Part- Instanz bzw. Some- Instanz	Meist im Konstruktor von Whole (selten später)	Oft zusammen mit Whole (evtl. später), evtl. auch Erzeugung außerhalb von Whole	Offen: vor der Nutzung durch einen Client
Eine Part-Instanz bzw. Some-Instanz kann <u>zu einem</u> <u>Zeitpunkt</u> mehrfach in der gleichen Beziehungart sein	Nein	Zwei Ansichten: 1. Nein 2. Ja (meist in Verbindung mit der Bezeichnung „Shared Aggregate“)	Ja

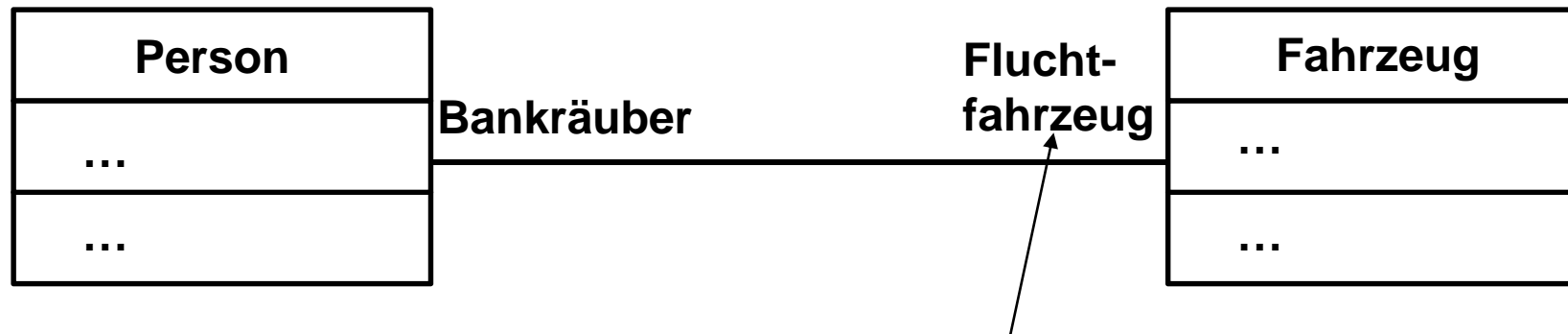
5.8 OO Modellierung mit UML: Klassendiagramm

Kriterium	Komposition	Aggregation	Using
UML			
Eine Part-Instanz bzw. Some-Instanz kann <u>zu verschiedenen Zeitpunkten</u> in der gleichen Beziehungsart sein	Zwei Ansichten: 1. Nein 2. Ja (z.B. erlaubt UML eine verantwortliche Weitergabe der Instanz z.B. vor der Zerstörung der Whole-Instanz)	Ja	Ja
Die Part-Instanz bzw. Some-Instanz hängt mit seiner Existenz von Whole bzw. Client ab	Ja	Nicht zwingend	Nie
Kontrolle über die Part-Instanz bzw. Some-Instanz von Whole bzw. Client aus	Stärkste Kontrolle	Mittlere bis starke Kontrolle	Keine Kontrolle

Verfeinerung der Beziehungen durch mehr Details an den Beziehungen (optional)



Rollen

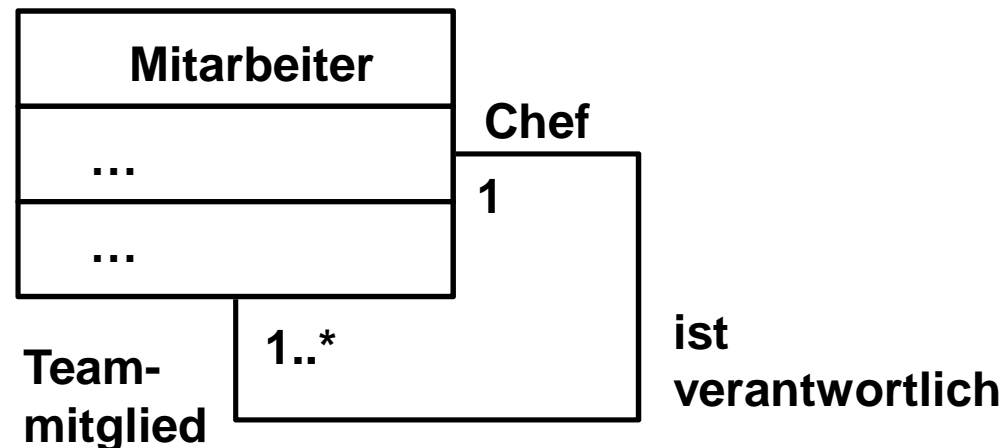


Rollenname:

Information über die Bedeutung einer Klasse (hier Fahrzeug) bzw. ihrer Objekte in der Assoziation

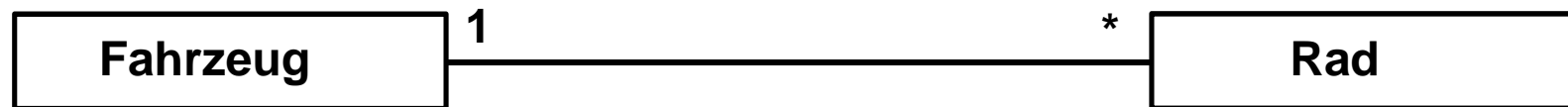
Reflexive Assoziation

- Wird auch zirkuläre oder rekursive Assoziation genannt
- Beziehung zwischen Objekten, die zur gleichen Klasse gehören (Typ verweist auf sich selbst)
- Rollen sind hier besonders wichtig



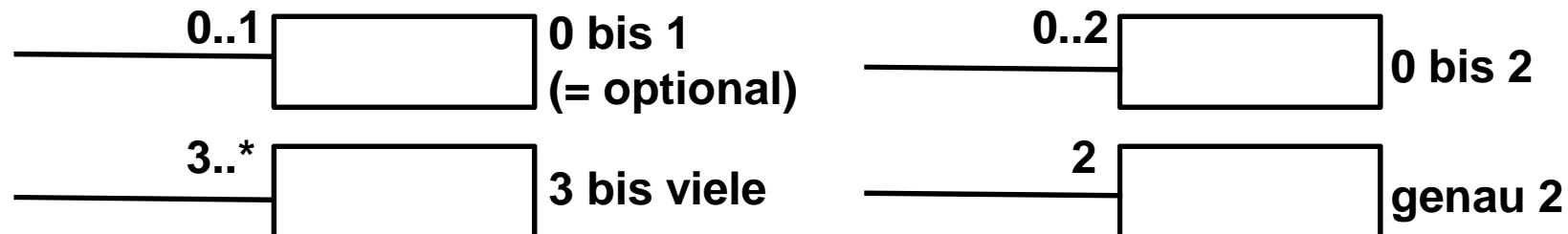
Multiplizität (bzw. Kardinalität, Anzahlangabe, Vielfachheit)

- Verfeinert Assoziationen (allgemeine oder spezifischere)
- Spezifiziert, wie viele Objekte ein bestimmtes Objekt kennen kann (bzw. in Verbindung steht)



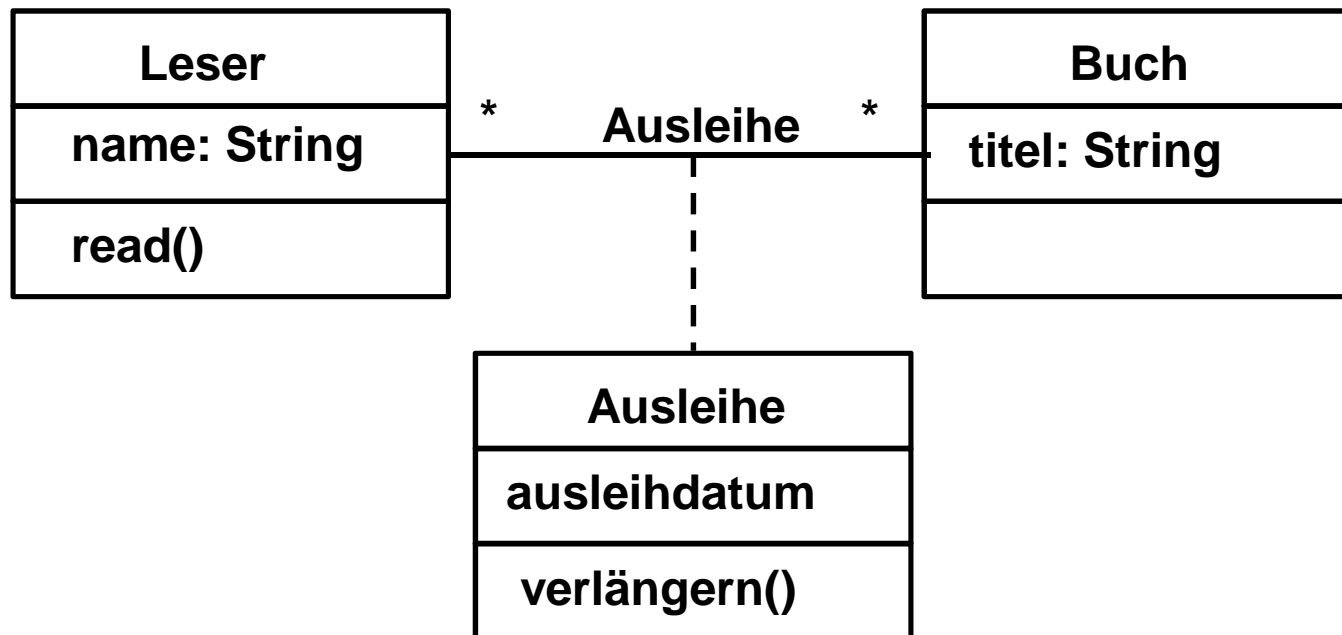
„Ein Fahrzeug hat 0 bis viele Räder“

„Ein Rad gehört zu genau einem Fahrzeug“



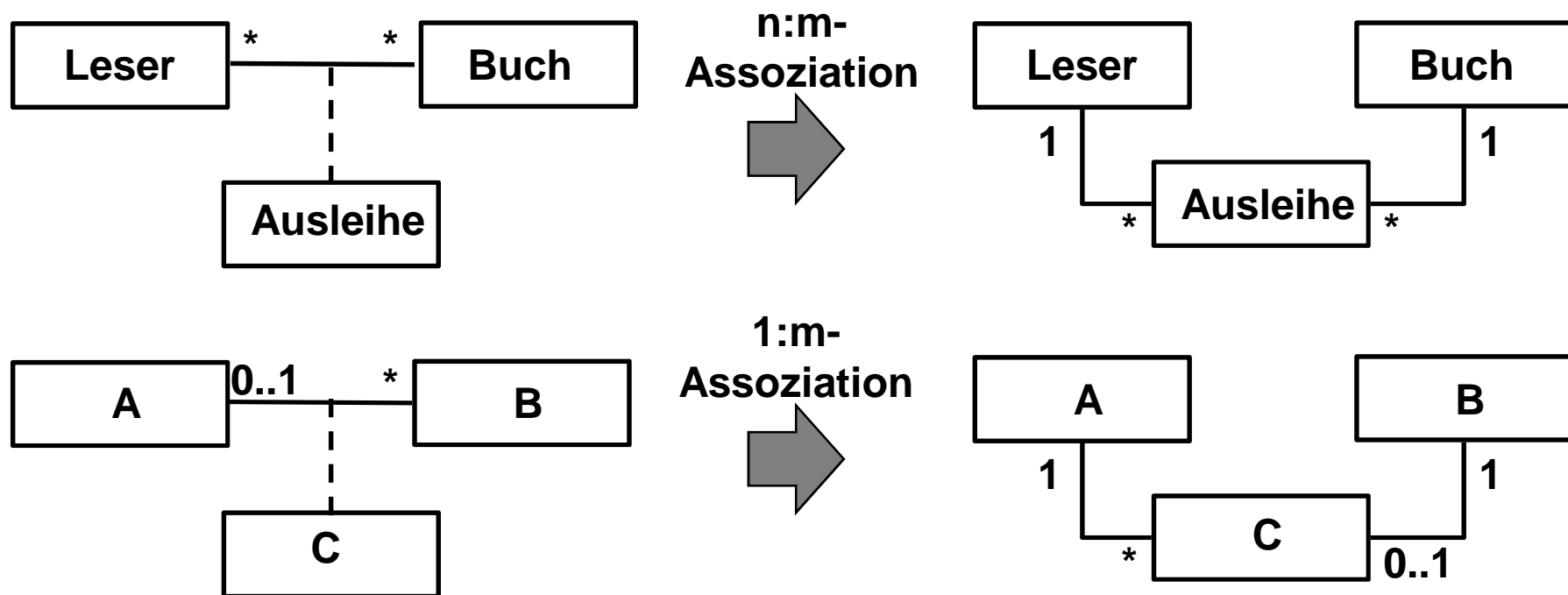
Assoziationsklasse (association class)

- Assoziationen, die zusätzlich Eigenschaften einer Klasse besitzen sollen
- Auch als attributierte Assoziation bezeichnet

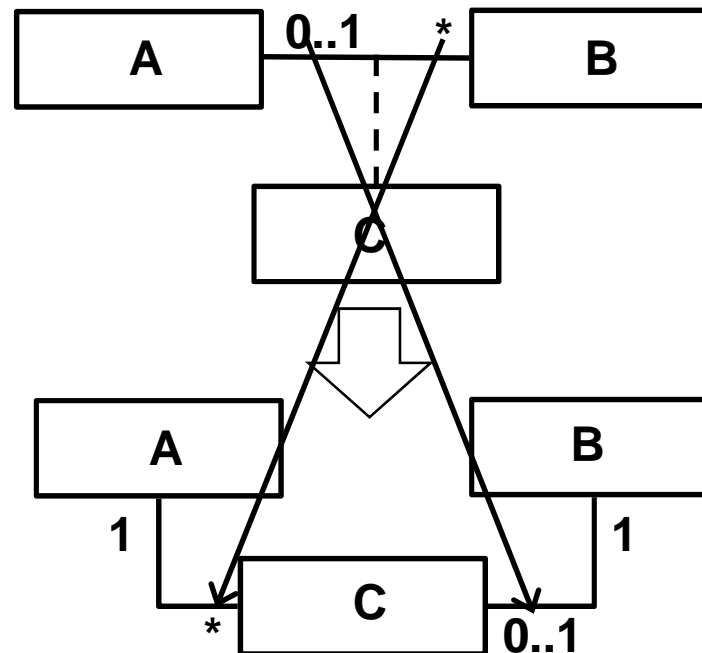


Assoziationsklasse: Transformation

- Immer möglich: Transformation in zwei Assoziationen und eine eigenständige Klasse



Assoziationsklasse: Transformation

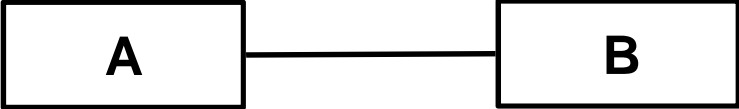
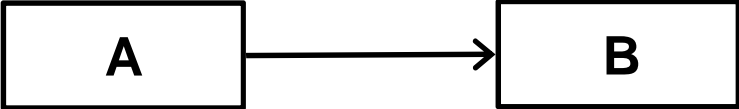
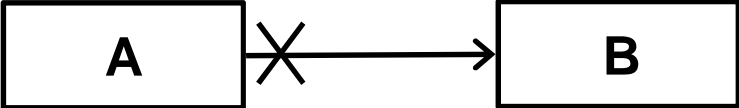
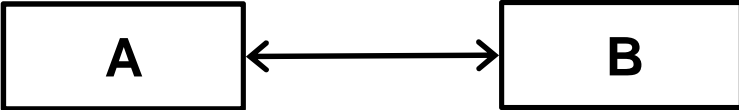


Navigierbarkeit (Navigability)



- “Die Assoziation ist von Mitarbeiter nach Projekt navigierbar” = Objekte von Mitarbeiter können auf Objekte von Projekt zugreifen (aber nicht unbedingt umgekehrt)
- Erweiterung der reinen Benutzt-Beziehung ...

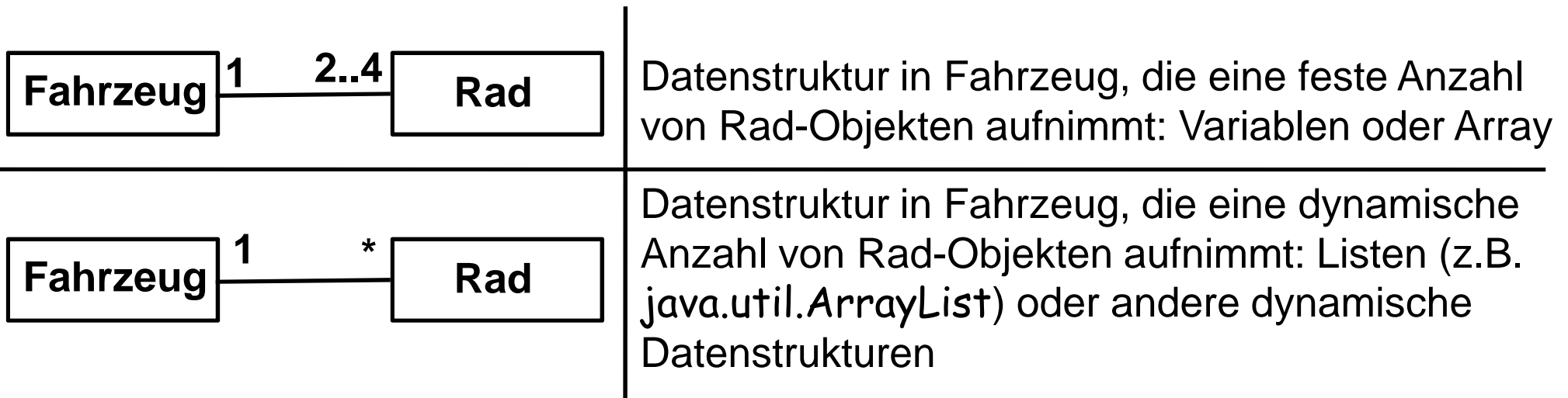
Navigierbarkeit

	unspezifiziert
	unidirektional bzw. gerichtet: A-Objekte haben Referenzen auf B-Objekte (umgekehrt hier undefiniert)
	unidirektional mit explizitem Ausschluss der Navigierbarkeit von B nach A: B-Objekte besitzen keine Referenzen auf A-Objekte
	bidirektional: A-Objekte besitzen Referenzen auf B-Objekte und umgekehrt

5.8 OO Modellierung mit UML: Klassendiagramm

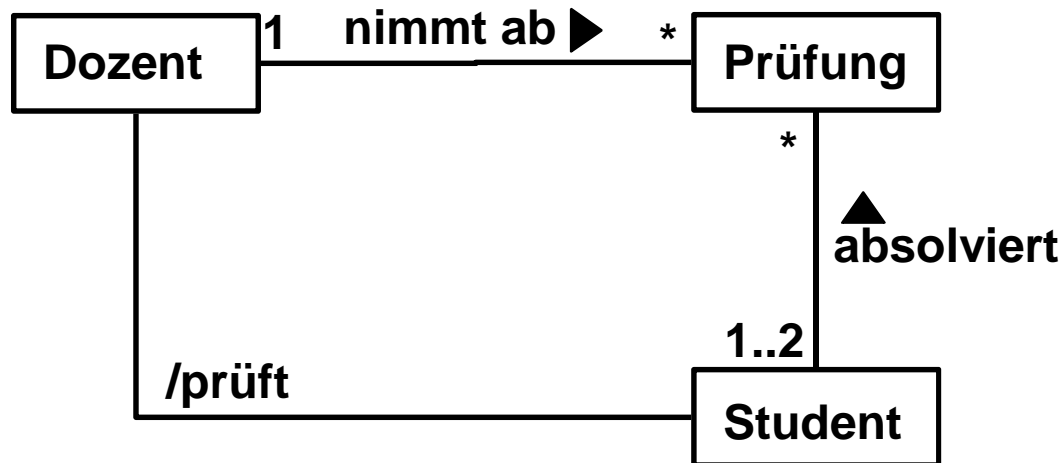
Umsetzung der Beziehungen in die Implementierung:

- Umsetzung der Beziehung in der Regel durch Referenzen
- Konsistenzbedingungen müssen für die jeweils modellierte Beziehungsart kontrolliert werden (z.B. 1:1 Beziehung)
- Für Beziehungen zu mehreren Objekten genügt eine Referenzvariable nicht:

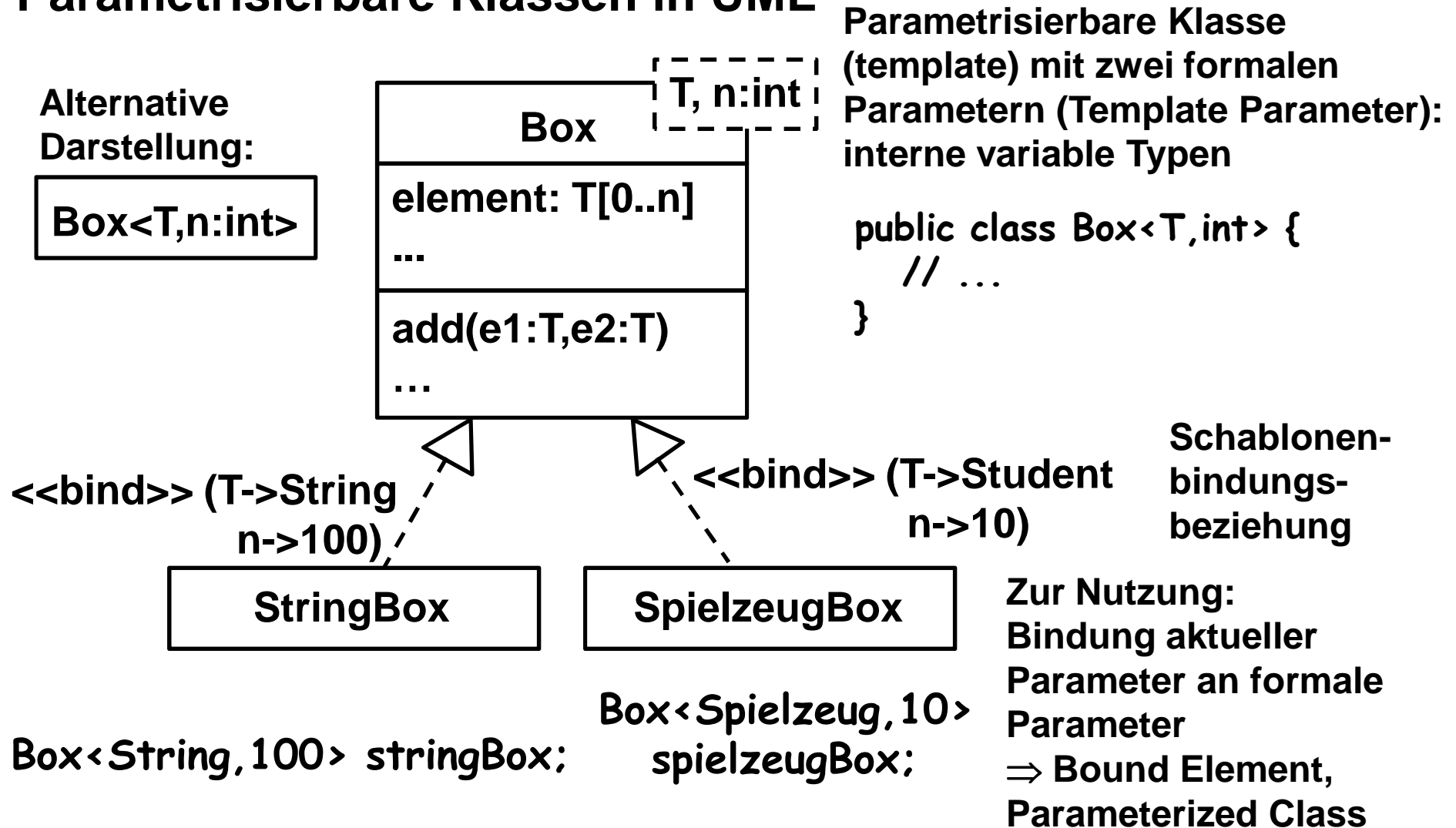


Abgeleitete Beziehung (Derived Association)

- Die gleiche Abhängigkeit ist bereits durch andere Assoziationen beschrieben



Parametrisierbare Klassen in UML



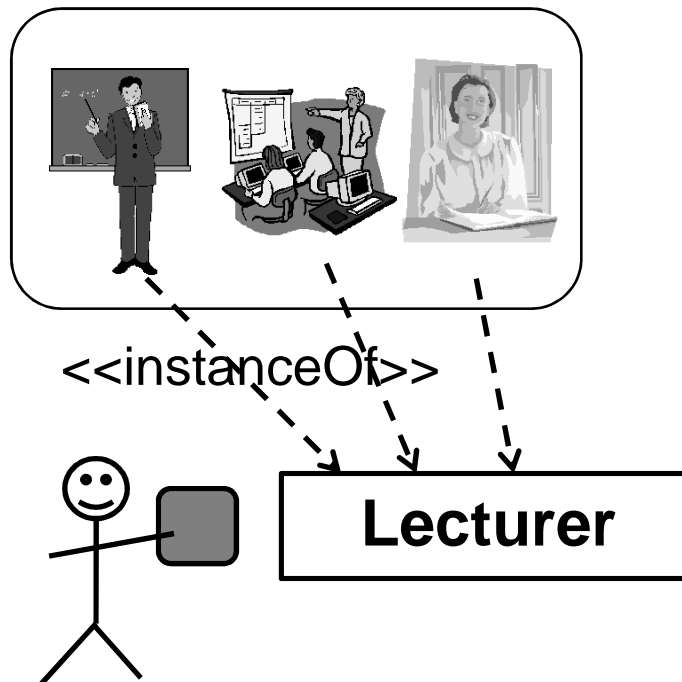
Schablonenklasse in Kurzform: (Notationsvariante)

Box<T - >String>

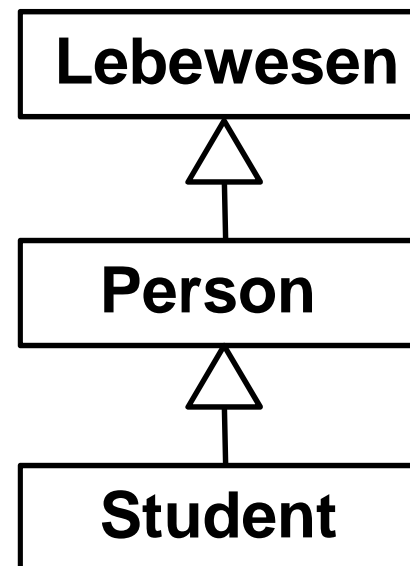
- Anonym gebundene Klasse (ohne eigenen Klassennamen)
- In Java gibt es nur anonym gebundene Klassen

Abstraktion und Hierarchie in OO

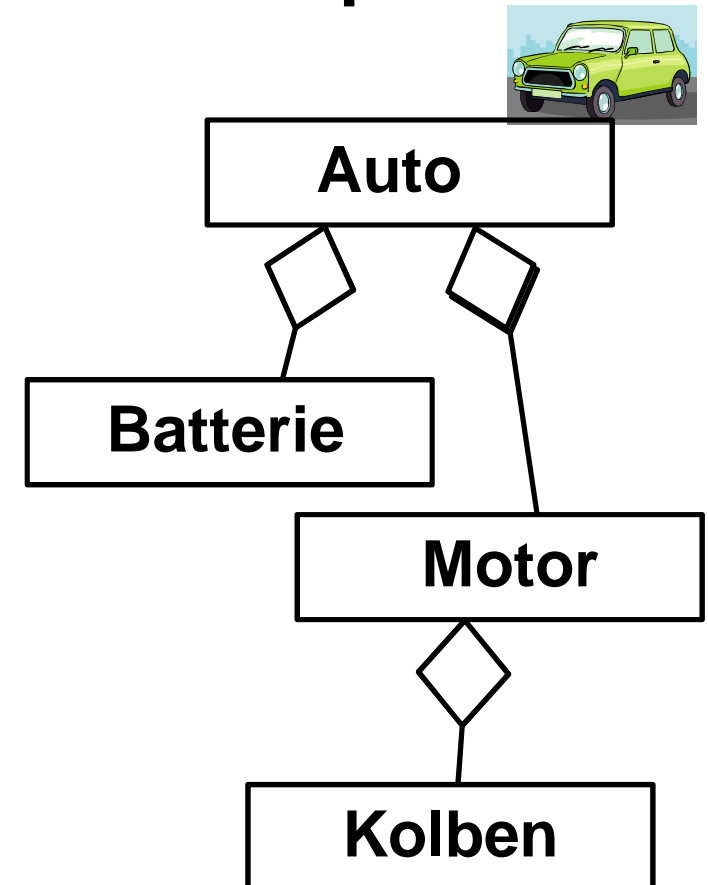
1. Klassenbildung



2. Vererbung

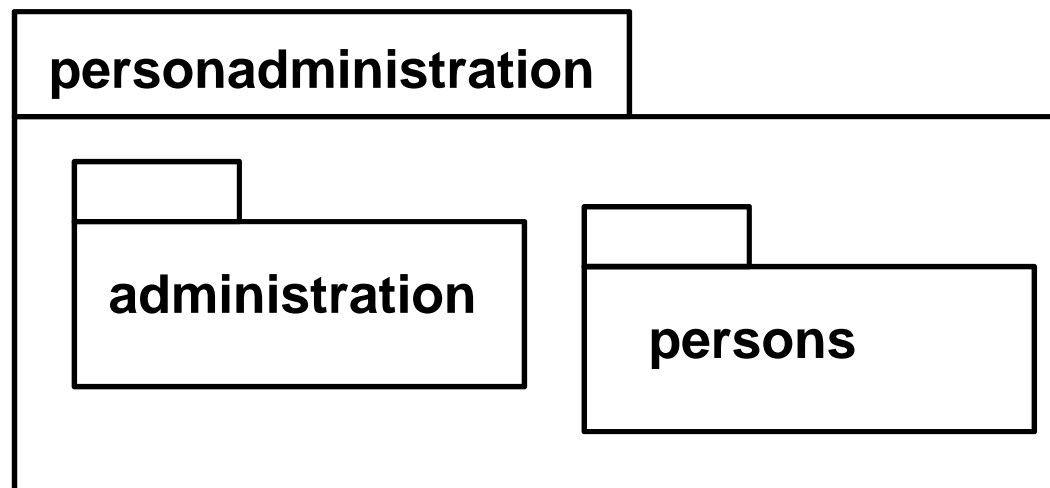


3. Aggregation/ Komposition



Abstraktion und Hierarchie in OO (Fortsetzung)

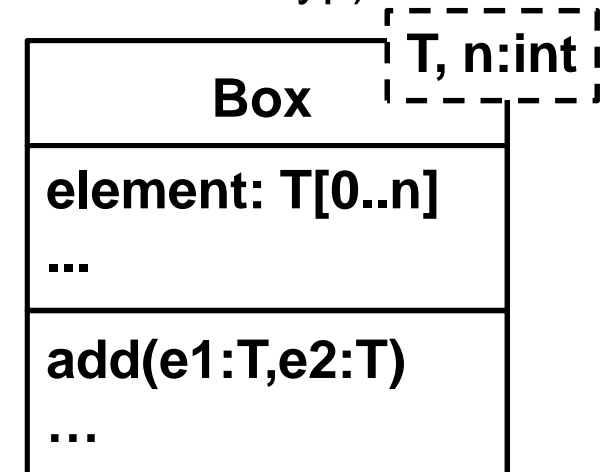
4. Pakete / Subsysteme



- Gruppen von Klassen zu einem gemeinsamen Aufgabenbereich bündeln
- Hierarchie von Systemteilen

5. Generizität

- Höheres Abstraktionsniveau durch generische Typen (Implementierung unabhängig zu einem konkreten Typ)



- Zerlegung der grösseren Funktionalität in kleinere (Dekomposition)
- Aufruf der „kleineren“ Funktionalität aus der grösseren heraus
- Delegation, Consultation, Forwarding

zunehmende Abstraktion



Sequence Diagrams (Sequenzdiagramme)

- **Klassendiagramme und Paketdiagramme: Modellierung statischer Strukturen**
Sequenzdiagramme: Modellierung dynamischer Abläufe
- **Sind eine Adaption der aus der Telekommunikationsindustrie stammenden MSC's (Message Sequence Charts).**
- **Sequenzdiagramme gehören (wie Kommunikationsdiagramme) zu den UML Interaktionsdiagrammen.**
- **Stellen exemplarisch den zeitlichen Ablauf gesendeter Nachrichten zwischen den beteiligten Objekten dar (exemplarisch konkrete aber unvollständige Ablaufspezifikation).**
- **Besitzen zwei Dimensionen:**
 - **Von links nach rechts sind die Kommunikationspartner aufgereiht.**
 - **Von oben nach unten verläuft eine (imaginäre) Zeitachse.**

Elemente eines Sequenzdiagramms:

- Kommunikationspartner
- Lebenslinien
- Nachrichten
- Sprachmittel zur Ablaufkontrolle.

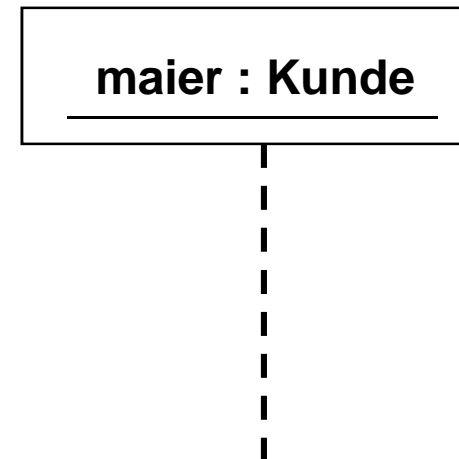
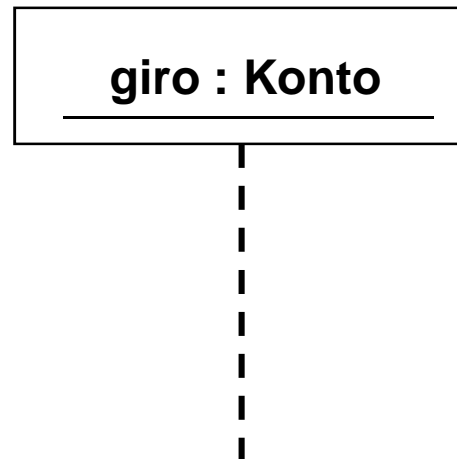
5.8 OO Modellierung mit UML: Sequenzdiagramm

Kommunikationspartner und Lebenslinien

- Als Kommunikationspartner kann jedes Objekt des Systems auftreten.



- Eine Lebenslinie (life line) wird als gestrichelte Linie dargestellt und repräsentiert die Lebenszeit eines Kommunikationspartners.



5.8 OO Modellierung mit UML: Sequenzdiagramm

Nachrichten

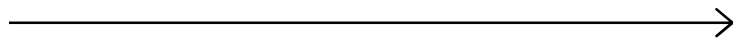
- **Kommunikationspartner versenden Nachrichten untereinander.**
- **Nachrichten werden als Pfeile zwischen den Lebenslinien der Kommunikationspartner dargestellt.**

Nachrichtenarten

Es werden zwei Arten von Nachrichten unterschieden:

- **Synchrone Nachrichten**
Aufrufer wartet bis der aufgerufene Kommunikationspartner die Verarbeitung beendet hat und eine Rückantwort geliefert hat.
- **Asynchrone Nachrichten**
Aufrufer der Nachricht wartet mit der Verarbeitung nicht auf den aufgerufenene Kommunikationspartner.
Asynchrone Nachrichten werden verwendet, um mehrere Prozesse zu modellieren.

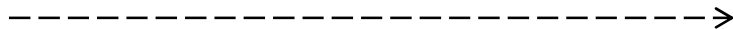
Grafische Darstellung der Nachrichtenarten:



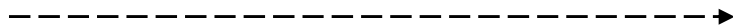
Asynchrone Nachricht



Synchrone Nachricht



Antwortnachricht (Alternativen)



5.8 OO Modellierung mit UML: Sequenzdiagramm

- Beschriftung der Nachrichtenpfeile (vereinfacht)

Nachricht ::= [NameDerNachricht ["(" Argumentliste ")"]
Argumentliste ::= {(NameDesParameters ["=" Argumentwert]) | "-"} }

Vor der Nachricht kann eine Bedingung stehen: "[Bedingung]"

- Beschriftung der Rückantwort (vereinfacht)

Antwortnachricht ::= [Attribut "="] NameDerNachricht ["(" Rückgabeliste ")"]
[":" Rückgabewert]
Rückgabeliste ::= {(Attribut ["=" NameDesParameters] ":" Rückgabewert) | "-"} }

- **Konsistenz:** alle im Sequenzdiagramm verwendeten Nachrichten müssen im Klassendiagramm mit entsprechender Signatur definiert sein

Beispiele für Nachrichtenbezeichnungen

Im Klassendiagramm ist eine Operation

`findeNamen(in telNummer:String, out name:String): boolean`
deklariert.

Aufruf der Operation (Variante1)

`findeNamen(nummer,-)`

Aufruf der Operation (Variante2)

`findeNamen(telNummer=nummer)`

Rückantwort zu der Operation (Variante1)

`findeNamen(-,x:'Sophie'):true`

Rückantwort zu der Operation (Variante2)

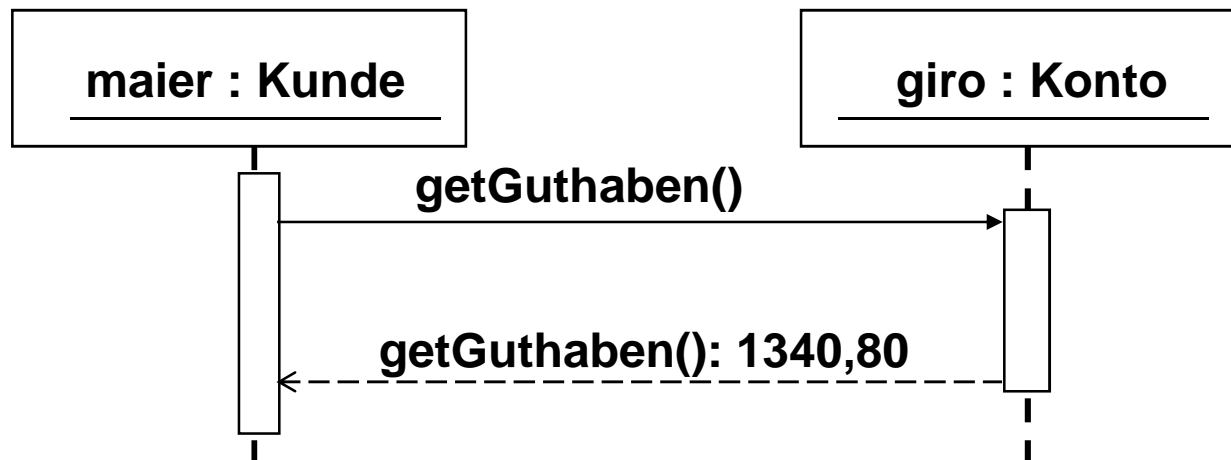
`findeNamen(x=name:'Sophie'):true`

Bedingte Operation

`[nummer<>0] findeNamen(nummer,-)`

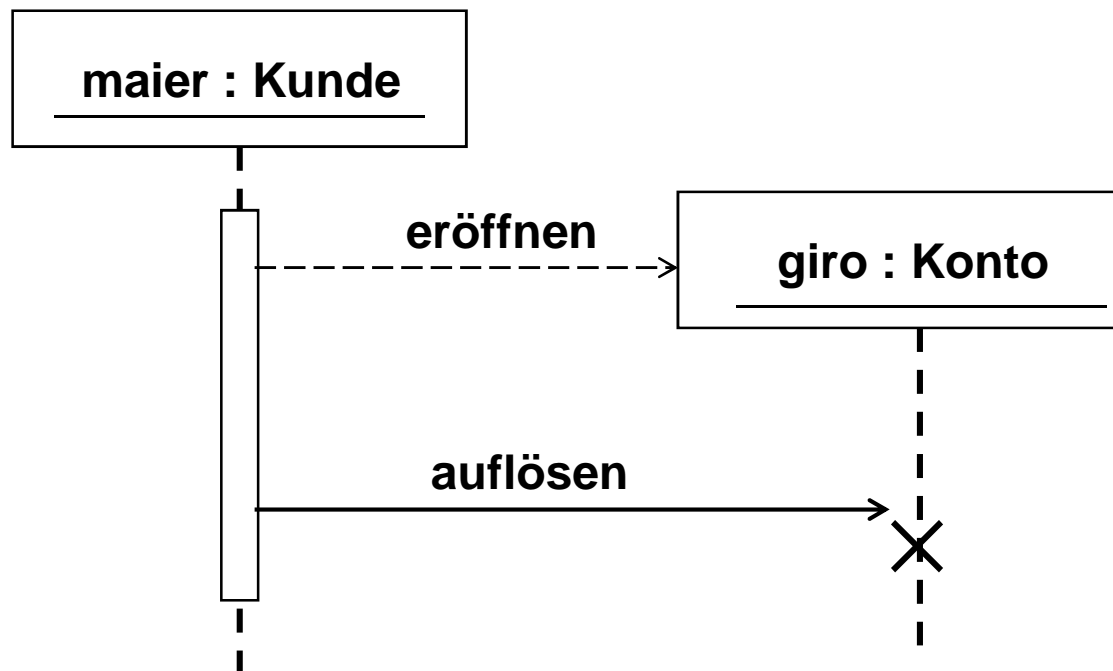
Aktionssequenz (Aktivierung)

- Ein Kommunikationspartner kann neben dem Senden und Empfangen von Nachrichten verschiedene Tätigkeiten ausführen.
- Aktionssequenzen, in denen ein Kommunikationspartner etwas durchführt, werden als senkrechte Balken auf der Lebenslinie, die auch geschachtelt auftreten können, dargestellt (Activation Bar, Aktivierungsbalken)
- Eine Aktionssequenz ist durch ein Start- und ein Endereignis eingeschlossen.
- Die Ausführung der Aktionssequenz benötigt Zeit.



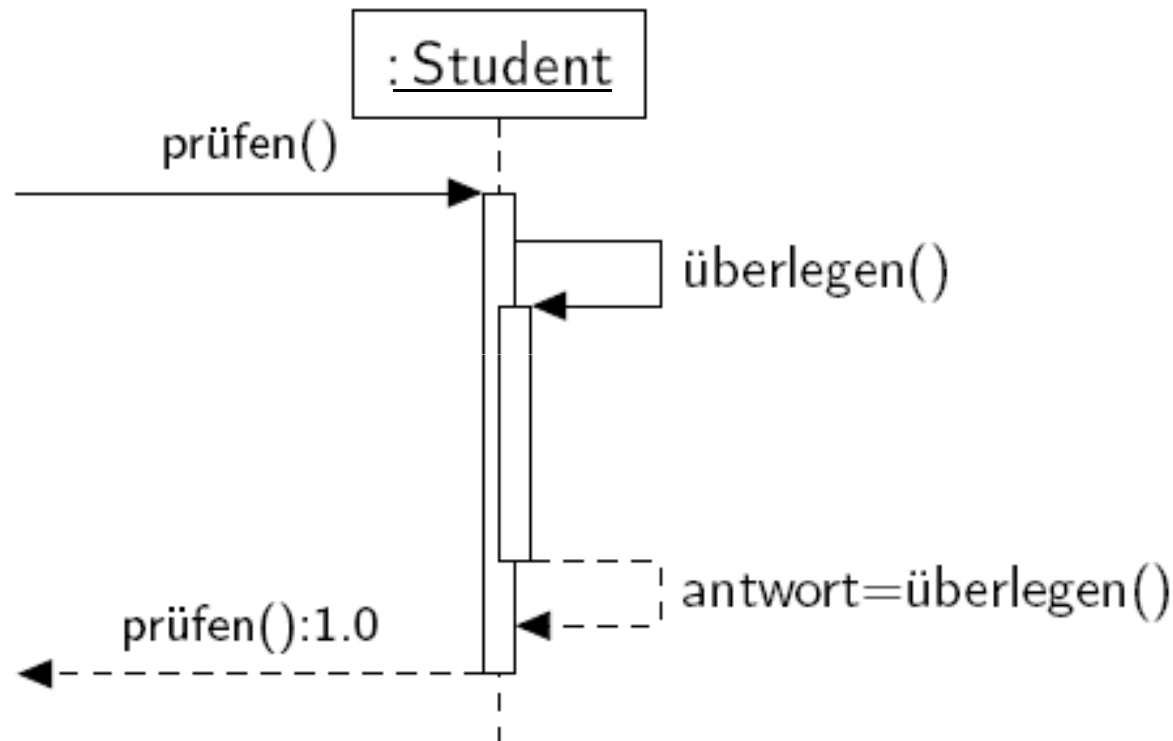
Erzeugen und Zerstören von Objekten

- Erzeugungsnachrichten erschaffen Kommunikationspartner und damit verbundene Lebenslinien
- Die Zerstörung eines Objektes wird durch den Abbruch der Lebenslinie und einem Kreuz an dessen Ende dargestellt.



Selbstdelegation

- Aufruf einer Operation des gleichen Objekts (auch rekursive Aufrufe).
- Selbstdelegation wird u.a. verwendet, um den Aufruf von privaten Methoden, die nur innerhalb des Objekts sichtbar sind, darzustellen.



Ausblick: Sprachmittel zur Ablaufkontrolle (ab UML 2.0)

- **In Sequenzdiagrammen soll eigentlich keine komplexe Ablauflogik dargestellt werden (dafür sind Aktivitätsdiagramme besser).**
- **Ab UML 2.0 sind trotzdem sogenannte kombinierte Fragmente eingeführt worden, die eine bessere Flusskontrolle ermöglichen.**
- **Die wesentlichen Fragmente sind**
 - **Alternatives Fragment (alternative Ablaufmöglichkeiten),**
 - **Optionales Fragment (optionale Interaktionsteile),**
 - **Paralleles Fragment (nebenläufige Interaktionen),**
 - **Schleife,**
 - **Negation (ungültige Interaktionssequenz),**
 - **... (weitere)**

5.8 OO Modellierung mit UML: Sequenzdiagramm

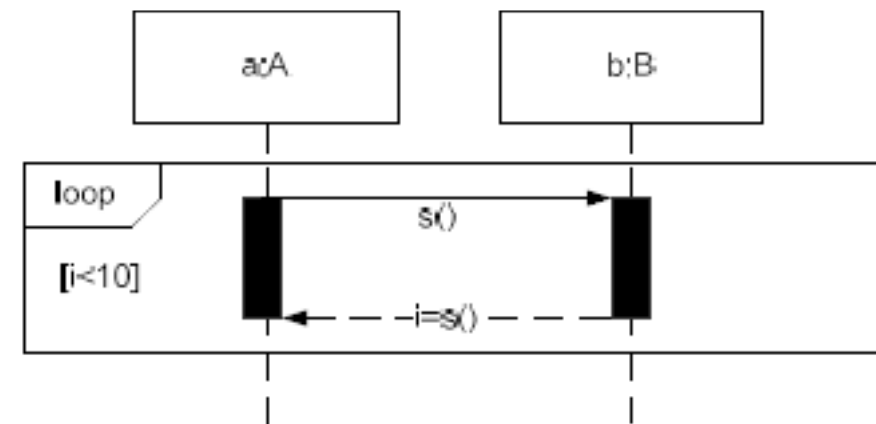
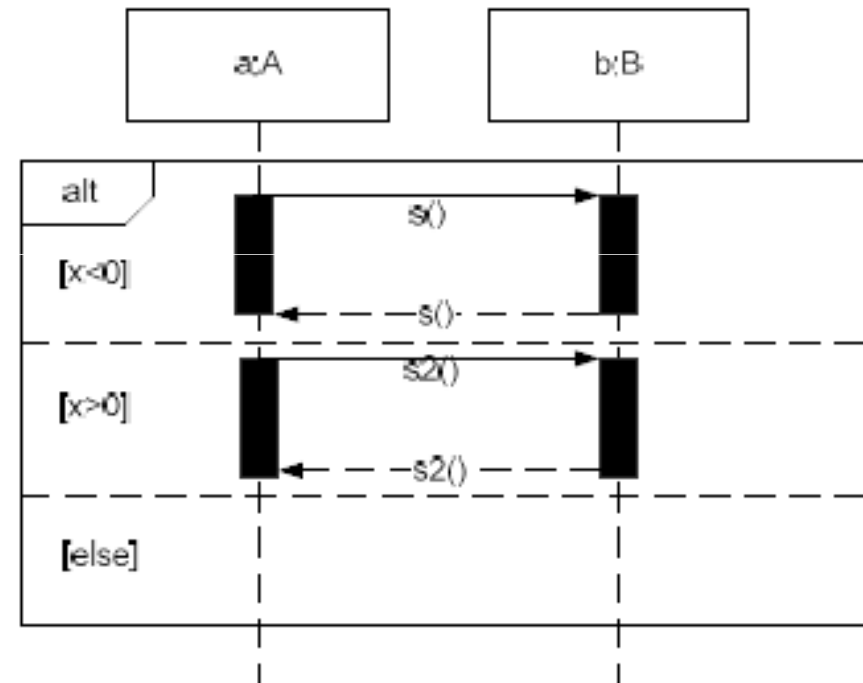
Ab UML 2.0 weitere Sprachmittel zur besseren Ablaufkontrolle

- **Combined Fragment:** Sequenz mit einem Rahmen, für die je nach Operatorangabe im Rahmen eine Ablauflogik definiert ist.

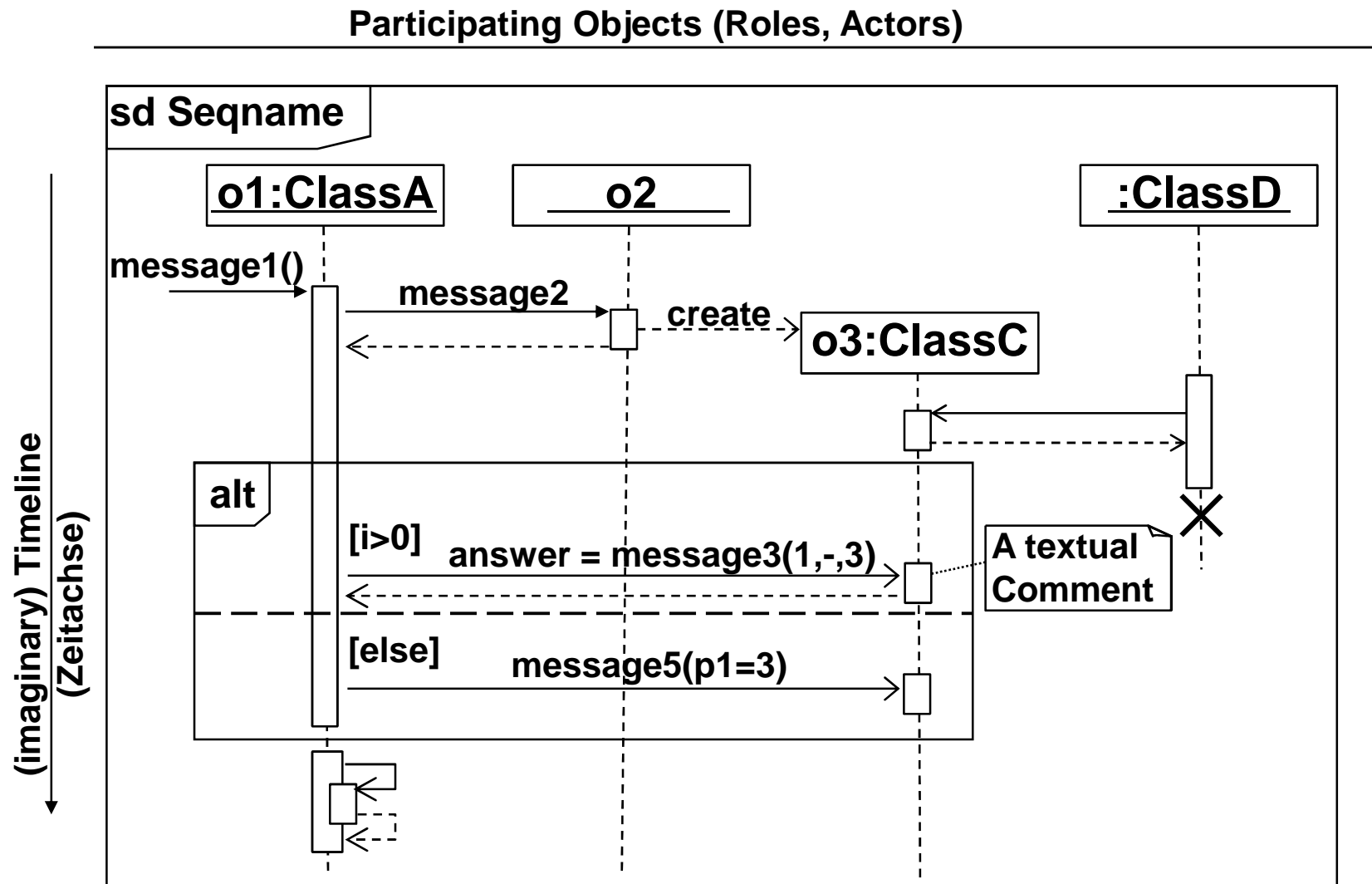
- **Beispiele für Fragment-Operatoren (Fragment Operators):**

Fragment	Interaktionsoperator
Optionale Fragmente	opt
Alternative Fragmente	alt
Parallele Fragmente	par
Schleife	loop
Negation	neg

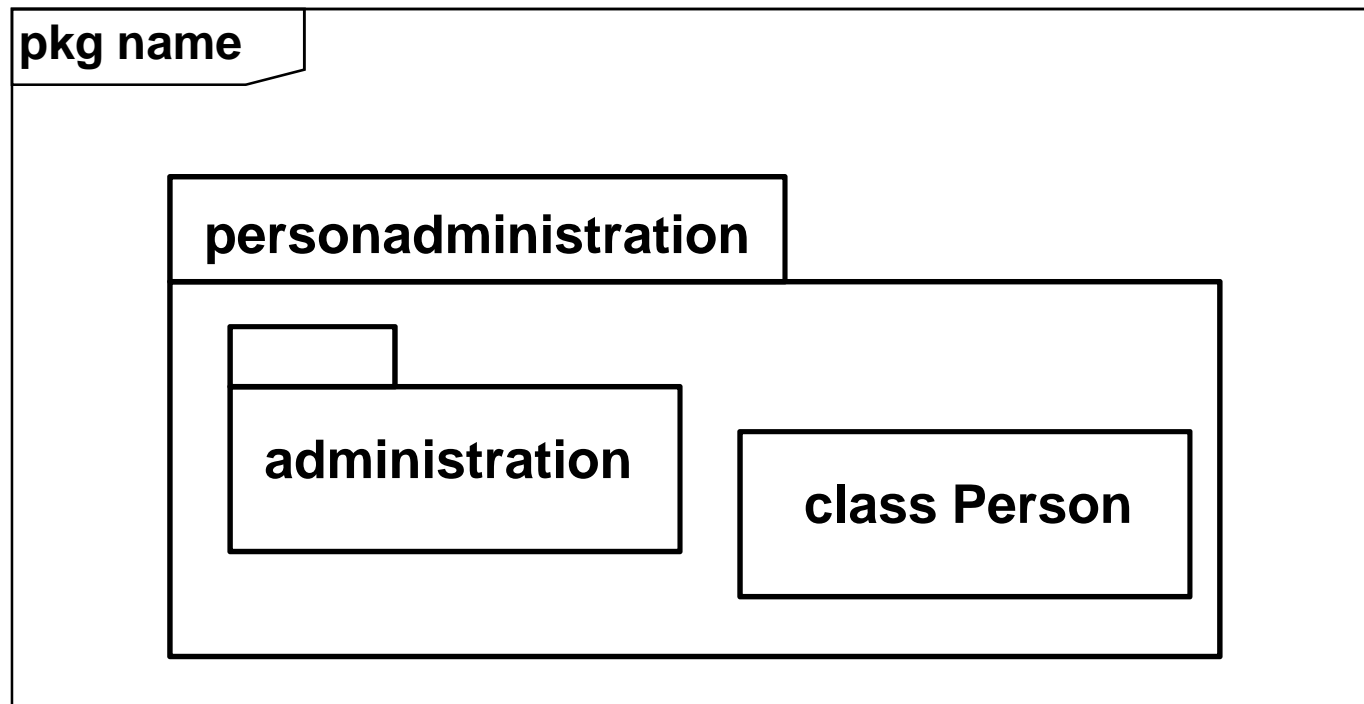
- **Fragment mit Symbol ref** verweist auf ein anderes Sequenzdiagramm (**Detaillierung**).



Beispiel eines Sequenzdiagramms mit Namen (eingeleitet mit Schlüsselwort sd):



Prinzipaufbau eines Paketdiagramms am Beispiel:

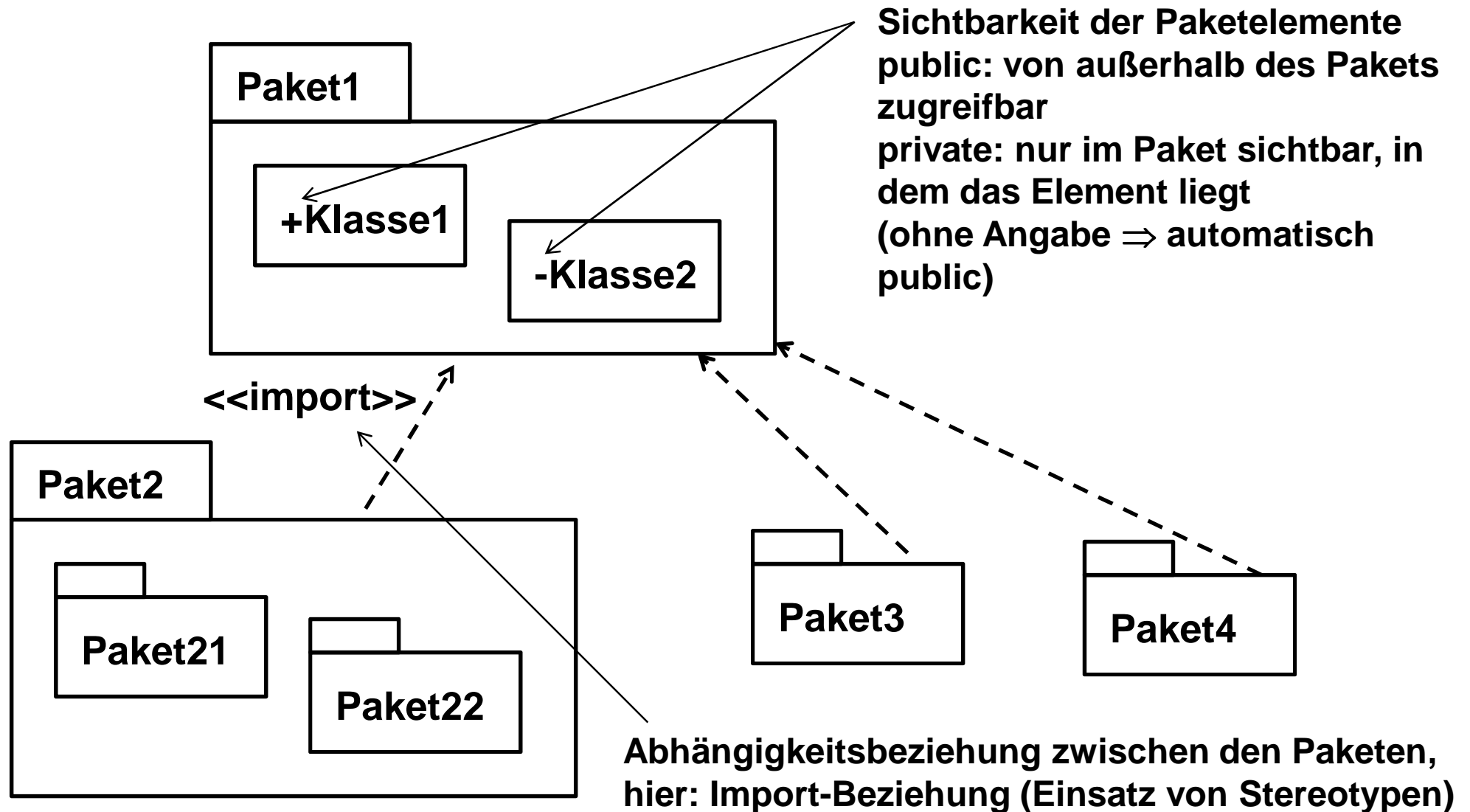


Pakete enthalten (geschachtelt) Pakete und Klassen

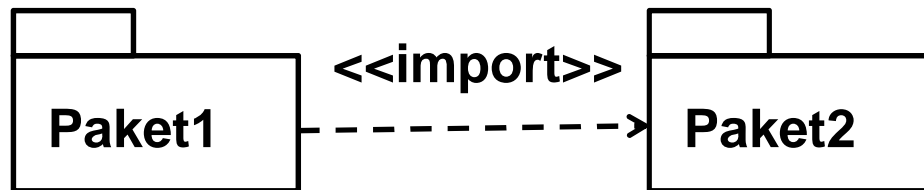
5.8 OO Modellierung mit UML: Paketdiagramm

- Paketdiagramme legen durch die Einteilung von Elementen (z.B. Klassen) auf Pakete die Gliederung eines Systems in Teilsysteme fest.
- Pakete: Gruppierung von Modellelementen zur Darstellung von Systemen oder Subsystemen.
Mit Paketen können Systeme auf höherer Abstraktionsebene modelliert werden (Strukturierung).
- Keine Verhaltensmodellierung.
- Format für qualifizierte Namen: `Paket1::Paket11::Paket111::Klasse`

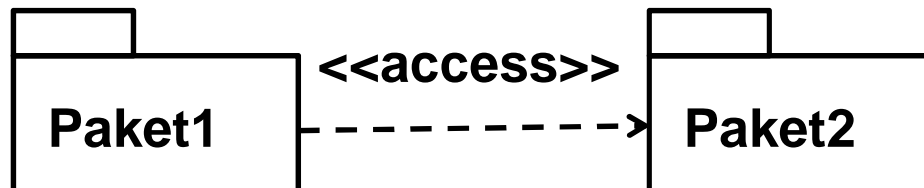
5.8 OO Modellierung mit UML: Paketdiagramm



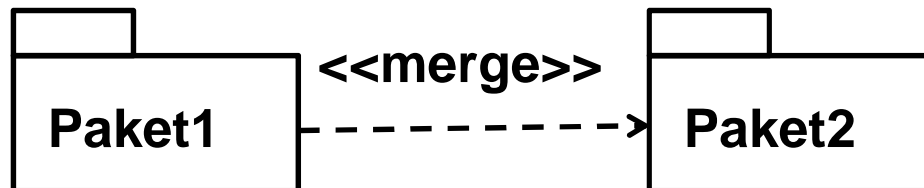
Abhängigkeitsbeziehungen:



- Paket1 importiert Paket2
- Elemente von Paket2 können in Paket1 ohne qualifizierenden Namen verwendet werden
- Public-import: die Sichtbarkeiten bleiben erhalten

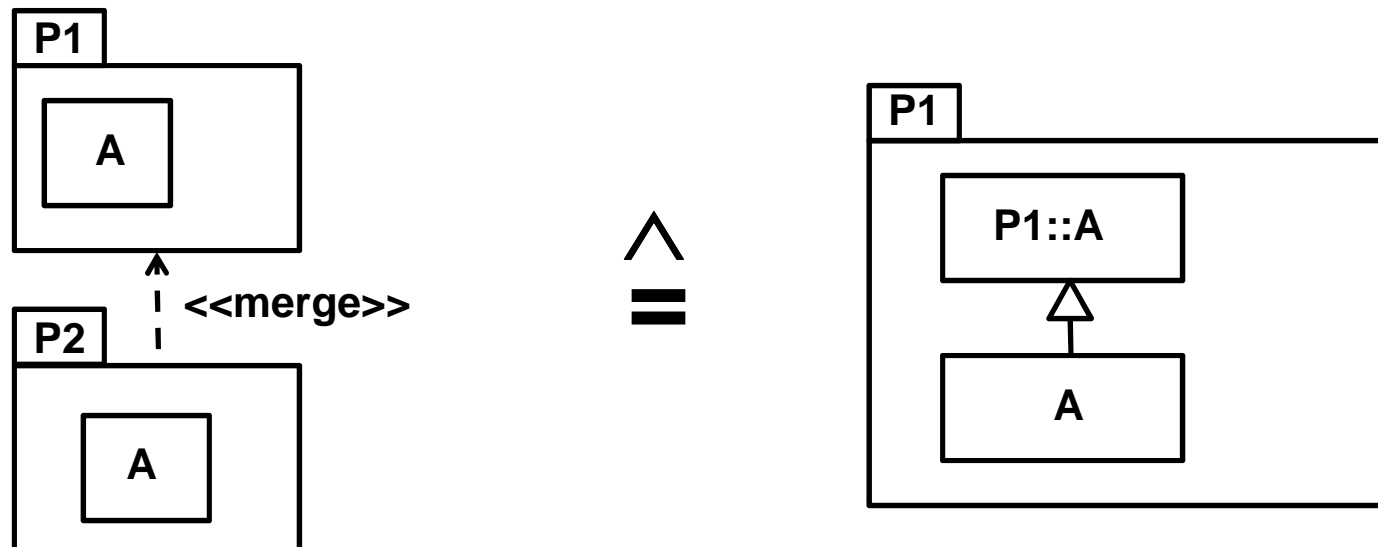
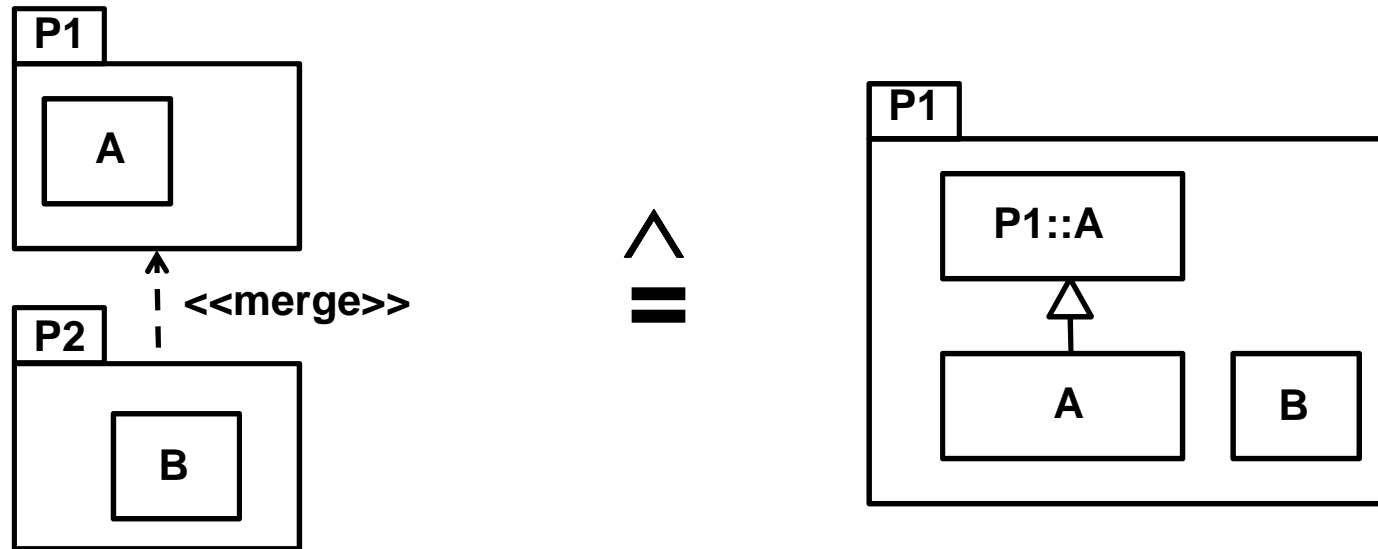


- Paket1 importiert Paket2
- Elemente von Paket2 können in Paket1 ohne qualifizierenden Namen verwendet werden
- Private-import: die Sichtbarkeiten der importierten Elemente wird in Paket1 privat



- Elemente aus Paket2 werden mit allen Importbeziehungen in Paket1 kopiert und durch Generalisierung und Redefinition vermischt
- Rekursive Vermischung gleichnamiger Unterpakete in Paket1 und Paket2

Regel zur Vermischung (Merge-Beziehung):

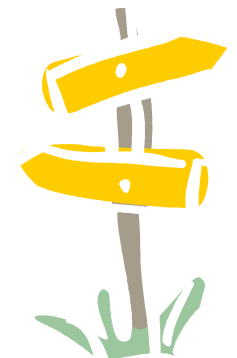


Zusammenfassung und Ausblick

- 1 **Software-Krise und Software Engineering**
- 2 **Grundlagen des Software Engineering**
- 3 **Projektmanagement**
- 4 **Konfigurationsmanagement**
- 5 **Software-Modelle**
- 6 **Software-Entwicklungsphasen, -prozesse, -vorgehensmodelle**
- 7 **Qualität**
- 8 **... Fortgeschrittene Techniken**

- 5.1 Grundlagen und Modelltypen
(Modellbegriff, Modellarten/Sichten, Einsatz, Modellvielfalt, Abstraktionsebenen)
- 5.2 Programmablaufplan
- 5.3 Struktogramm
- 5.4 Funktionsbaum
- 5.5 Structured Analysis
- 5.6 EBNF, Syntaxdiagramm
- 5.7 ERM
- 5.8 OO-Modelle mit UML

**bekannte
Modelle bzw.
Modellierungs-
sprachen**



→ Wege im Umgang mit der Software-Krise und
Umsetzung der Grundlagen und Prinzipien:
Einsatz von Modellen
**Mehr zu UML + weitere bekannte Modelle für
verschiedene Sichten und Einsatzzwecke**