

ingo

CHRISTIAN HUEMER
MARION SCHOLZ

Objektorientierte Modellierung mit UML

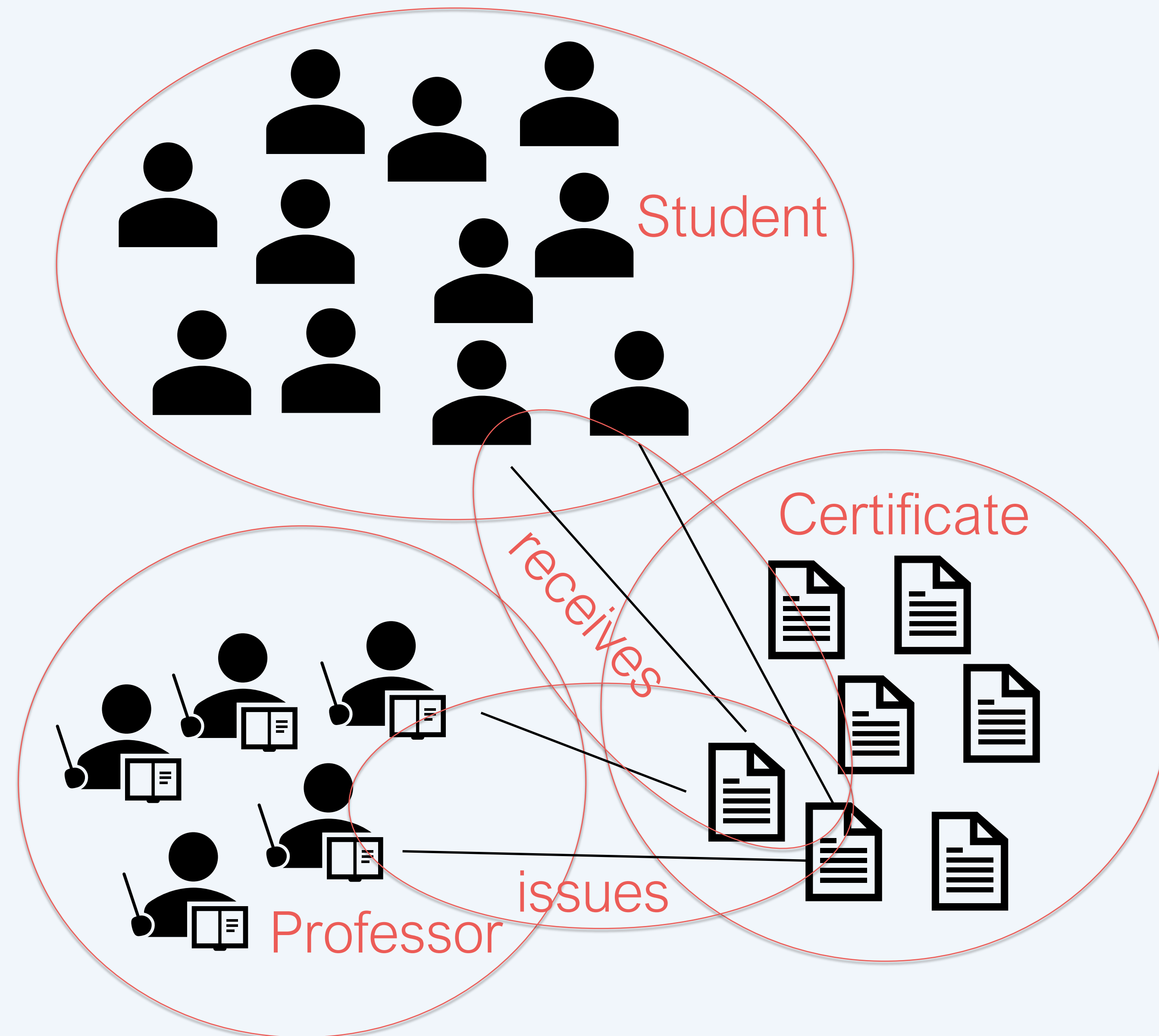
Inhalt	
1	Vom Objekt zur Klasse
2	Die Klasse
3	<i>entfällt</i>
4	Die Assoziation
5	Die Multiplizität und die Rolle
6	Die exklusive Assoziation und die Assoziationsklasse
7	Die n-äre Assoziation
8	Die Aggregation
9	Die Generalisierung
10	Exkurs: Die Ordnung und Eindeutigkeit von Assoziationen
11	Gesamtbeispiel
12	Die Datentypen
13	Die Übersetzung nach Java
15	Forward versus Reverse Engineering
14	Das Paketdiagramm

Strukturmodellierung Vom Objekt zur Klasse

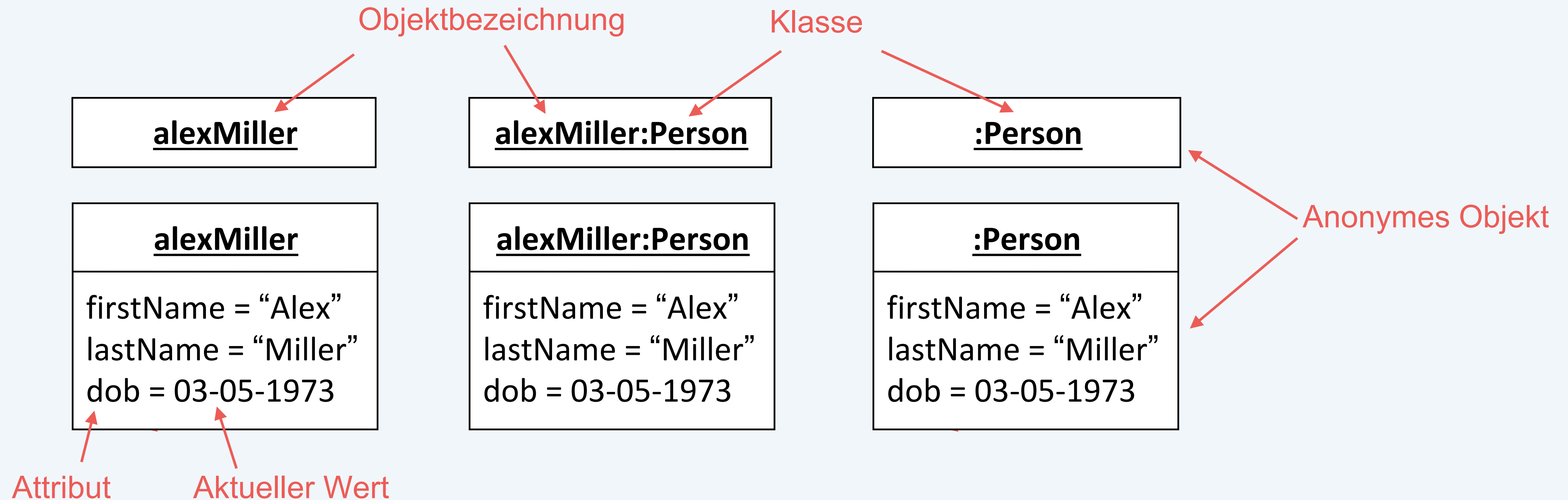


Christian Huemer und Marion Scholz

Objekte/Links und Klassen/Assoziationen

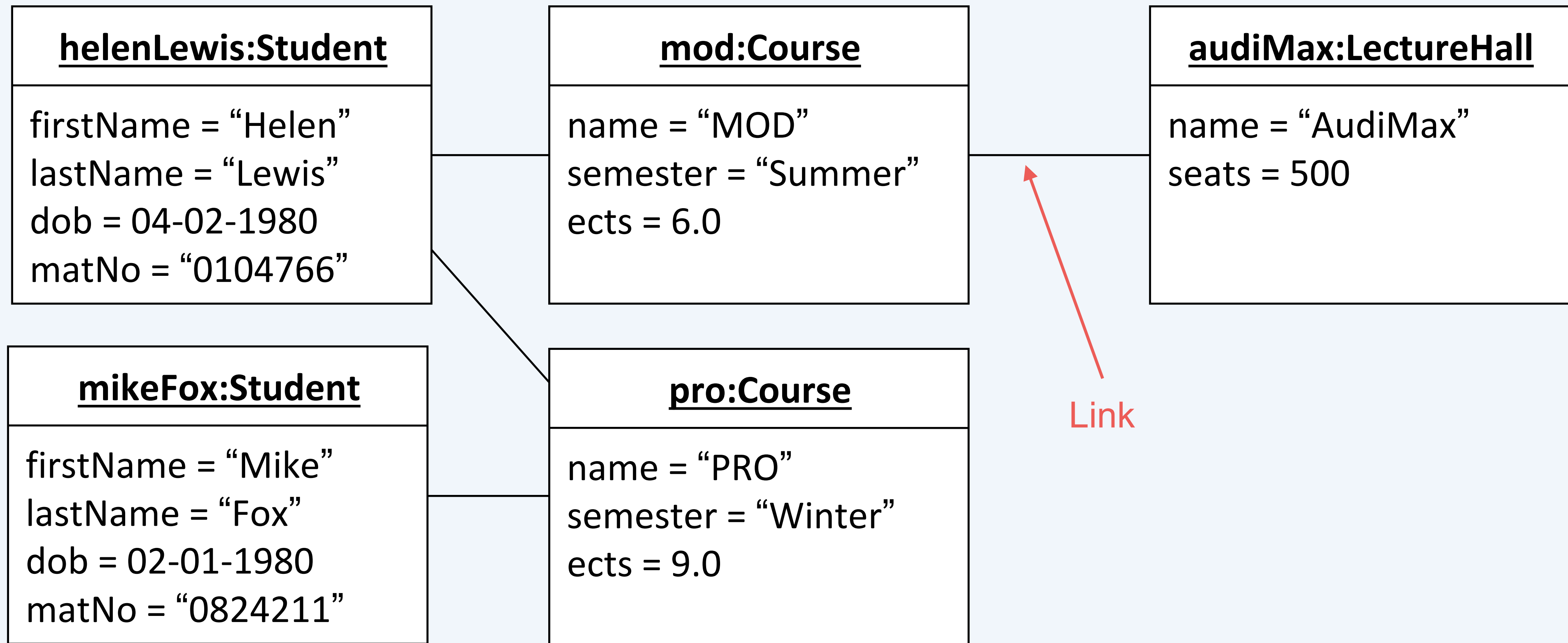


- Instanzen eines Systems
- Notationsvarianten:

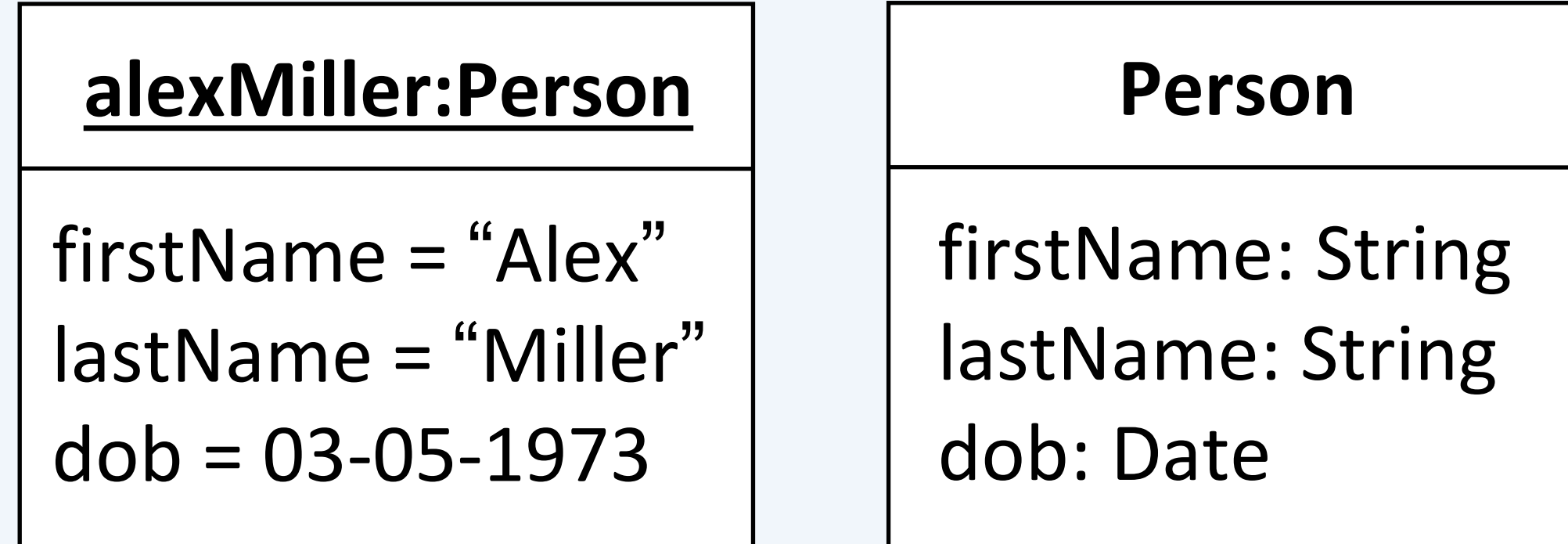


Objektdiagramm

- Beschreibt den strukturellen Aspekt eines Systems auf Instanzebene
- Momentaufnahme (snapshot) des Systems
- Muss nicht vollständig sein



Vom Objekt zur Klasse



Objekt dieser Klasse

Klasse

- Instanzen eines Systems haben oft gleiche charakteristische Merkmale und gleiches Verhalten
- Klasse: Bauplan für eine Menge von ähnlichen Objekten eines Systems
- Objekte: Instanzen von Klassen
- Attribute: strukturelle Merkmale einer Klasse
 - Unterschiedlicher Wert für jede Instanz (= Objekt)
- Operationen: Verhalten einer Klasse
 - Für alle Objekte einer Klasse ident
 - ⇒ Werden im Objektdiagramm nicht abgebildet

Strukturmodellierung

Die Klasse



Christian Huemer und Marion Scholz

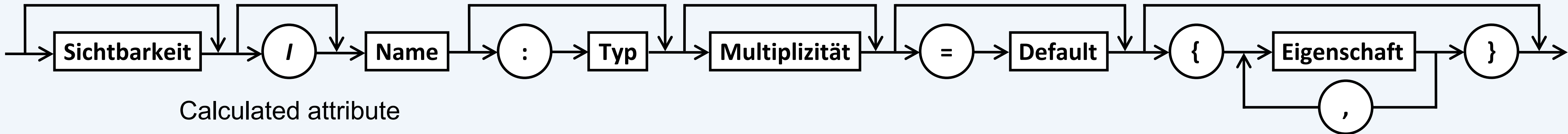
Course
name: String semester: SemesterType hours: float
getHours(): float getLecturer(): Lecturer getGPA(): float

Klassenname

Attribute

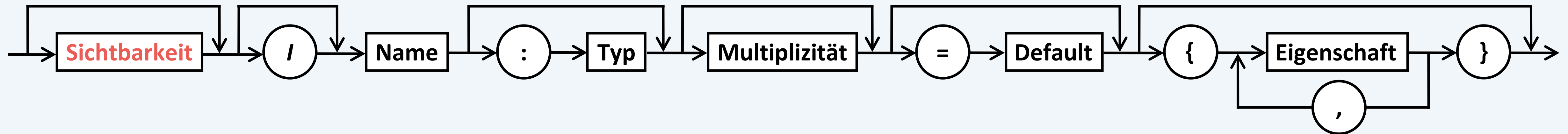
Operationen

Syntax der Attributspezifikation



Person
+ firstName: String + lastName: String - dob: Date # address: String[1..*]{unique,ordered} - ssNo: String {readOnly} - /age: int - password: String = "pw123" - <u>personsNumber</u> : int

Attributsyntax - Sichtbarkeit



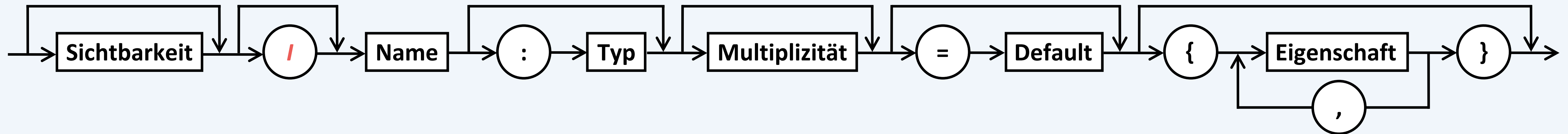
Person

```
+ firstName: String
+ lastName: String
- dob: Date
# address: String[1..*]{unique,ordered}
- ssNo: String {readOnly}
- /age: int
- password: String = "pw123"
- personsNumber: int
```

■ Wer darf auf das Attribut zugreifen

- + ... public
- - ... private
- # ... protected
- ~ ... package

Attributsyntax – Abgeleitetes Attribut

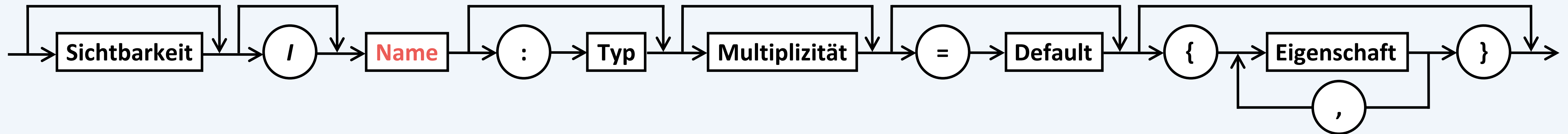


Person

```
+ firstName: String
+ lastName: String
- dob: Date
# address: String[1..*]{unique,ordered}
- ssNo: String {readOnly}
- /age: int
- password: String = "pw123"
- personsNumber: int
```

- Wert des Attributs wird von anderen Attributen abgeleitet
 - **age**: abgeleitet vom Geburtsdatum (**dob**)

Attributsyntax - Name

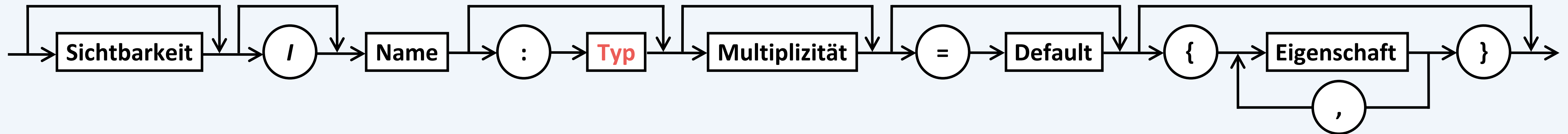


■ Name des Attributs

Person

```
+ firstName: String
+ lastName: String
- dob: Date
# address: String[1..*]{unique,ordered}
- ssNo: String {readOnly}
- /age: int
- password: String = "pw123"
- personsNumber: int
```

Attributsyntax - Typ



Person

```
+ firstName: String
+ lastName: String
- dob: Date
# address: String[1..*]{unique,ordered}
- ssNo: String {readOnly}
- /age: int
- password: String = "pw123"
- personsNumber: int
```

■ Klasse

■ Datentyp

■ Primitive Datentypen

- Vordefiniert: **Boolean**, **Integer**, **UnlimitedNatural**, **String**

- User-definiert:
«**primitive**»

- Zusammengesetzte Datentypen:
«**datatype**»

- Enumerationen:
«**enumeration**»

«**primitive**»

Float

round(): void

«**datatype**»

Date

day

month

year

«**enumeration**»

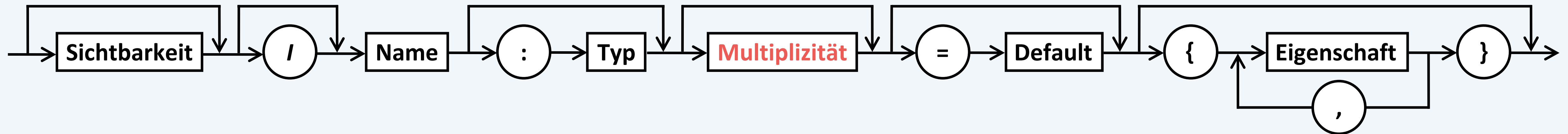
AcademicDegree

bachelor

master

phd

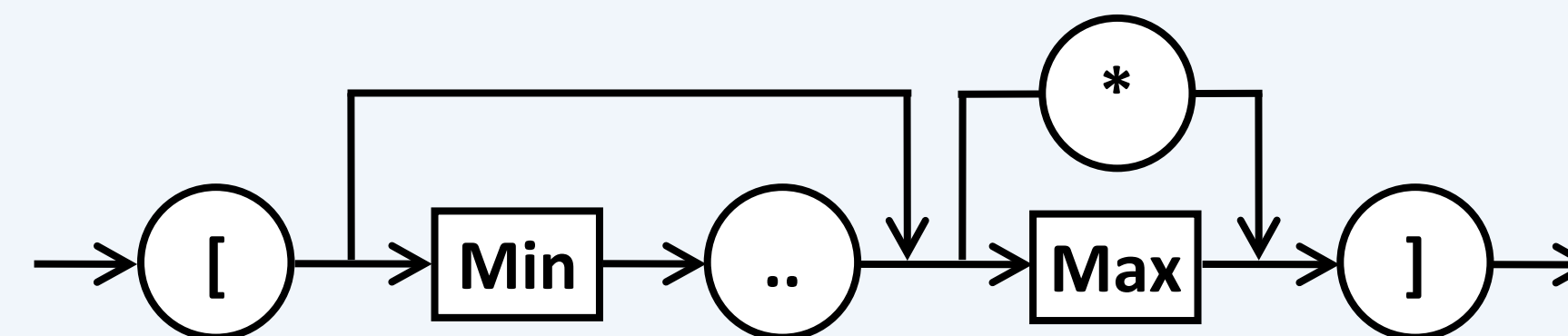
Attributsyntax - Multiplizität



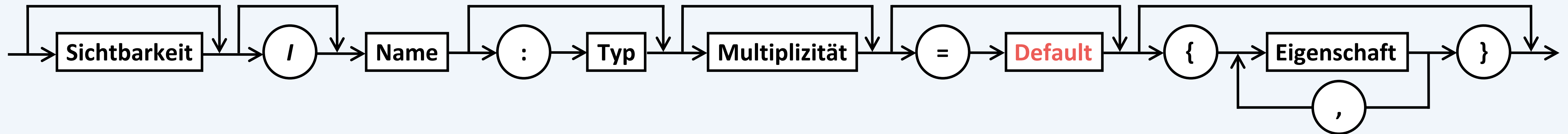
Person

+ firstName: String
+ lastName: String
- dob: Date
address: String[1..*]{unique,ordered}
- ssNo: String {readOnly}
- /age: int
- password: String = "pw123"
- personsNumber: int

- Anzahl an Werten die ein Attribut beinhalten kann
- Default: 1
- Notation: **[min..max]**
 - Keine Obergrenze: [*] oder [0..*]



Attributsyntax - Defaultwert

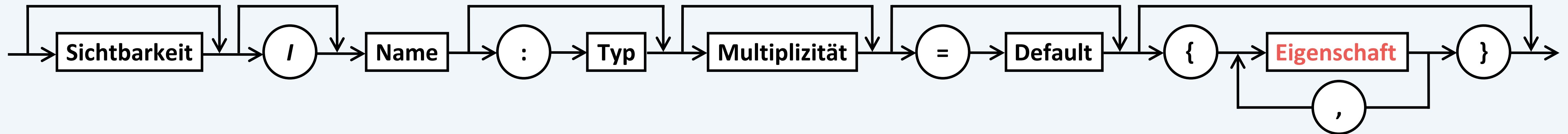


Person

```
+ firstName: String
+ lastName: String
- dob: Date
# address: String[1..*]{unique,ordered}
- ssNo: String {readOnly}
- /age: int
- password: String = "pw123"
- personsNumber: int
```

- Standardwert
- Wird verwendet, wenn der Wert des Attributs nicht explizit gesetzt wird

Attributsyntax - Eigenschaften



Person

```
+ firstName: String
+ lastName: String
- dob: Date
# address: String[1..*]{unique,ordered}
- ssNo: String {readOnly}
- /age: int
- password: String = "pw123"
- personsNumber: int
```

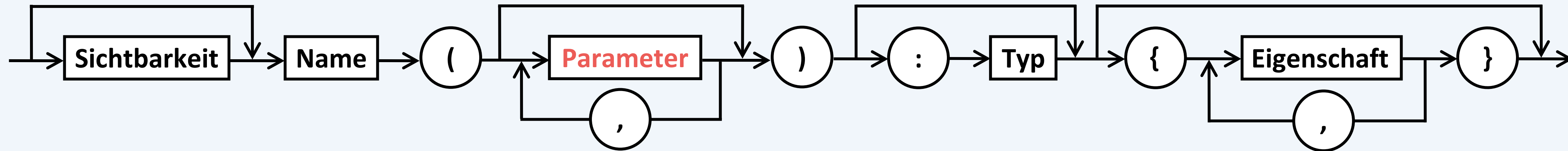
■ Vordefinierte Eigenschaften

- {readOnly}
- {unique}, {non-unique}
- {ordered}, {unordered}

■ Mögliche Kombinationen

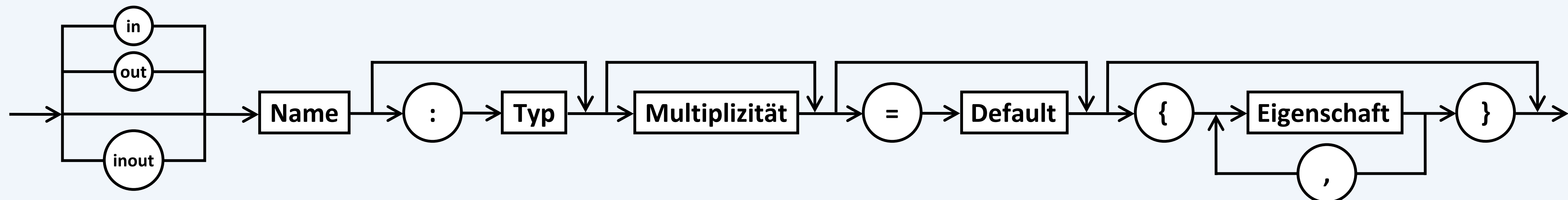
- Menge: {unordered, unique}
- Multimenge: {unordered, non-unique}
- Geordnete Menge: {ordered, unique}
- Liste: {ordered, non-unique}

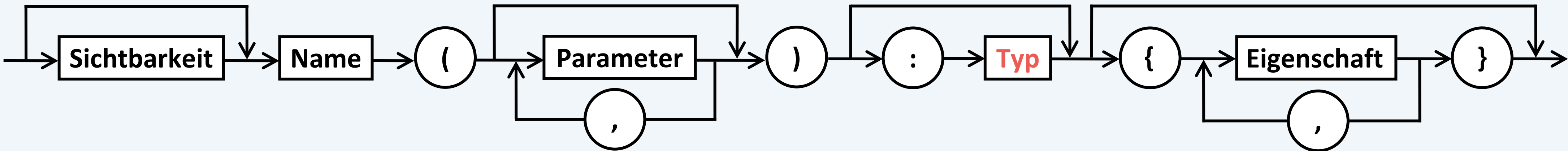
Operationssyntax - Parameter



Person
...
+getName(out fn: String, out ln: String): void + updateLastName(newName: String): boolean <u>+ getPersonsNumber(): int</u>

- Notation ähnlich Attribut
- Richtung des Parameters
 - **in** ... Eingabeparameter
 - **out** ... Ausgabeparameter
 - **inout** ... Kombiniertes Ein-/Ausgabeparameter



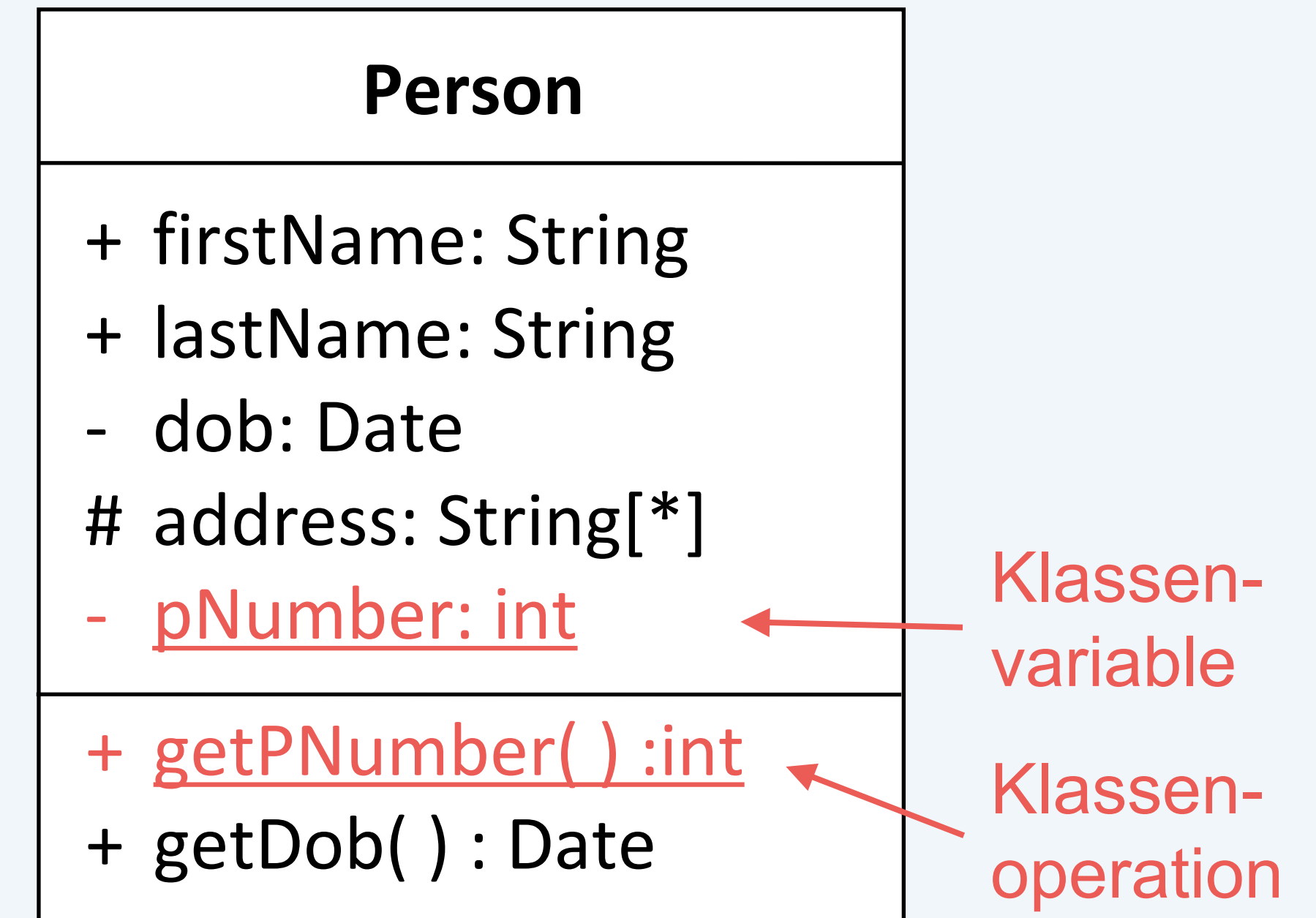


Person
...
+getName(out fn: String, out ln: String): void + updateLastName(newName: String): boolean <u>+ getPersonsNumber(): int</u>

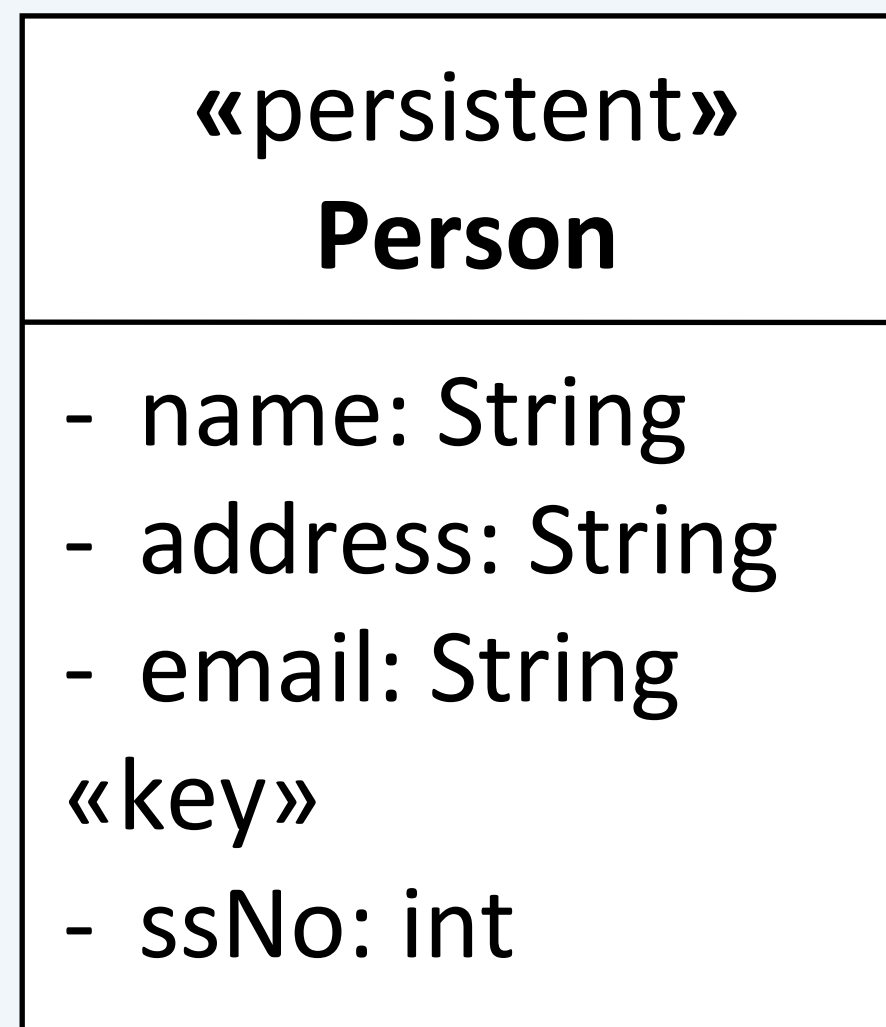
■ Typ des Rückgabewerts

Klassenvariable und Klassenoperation

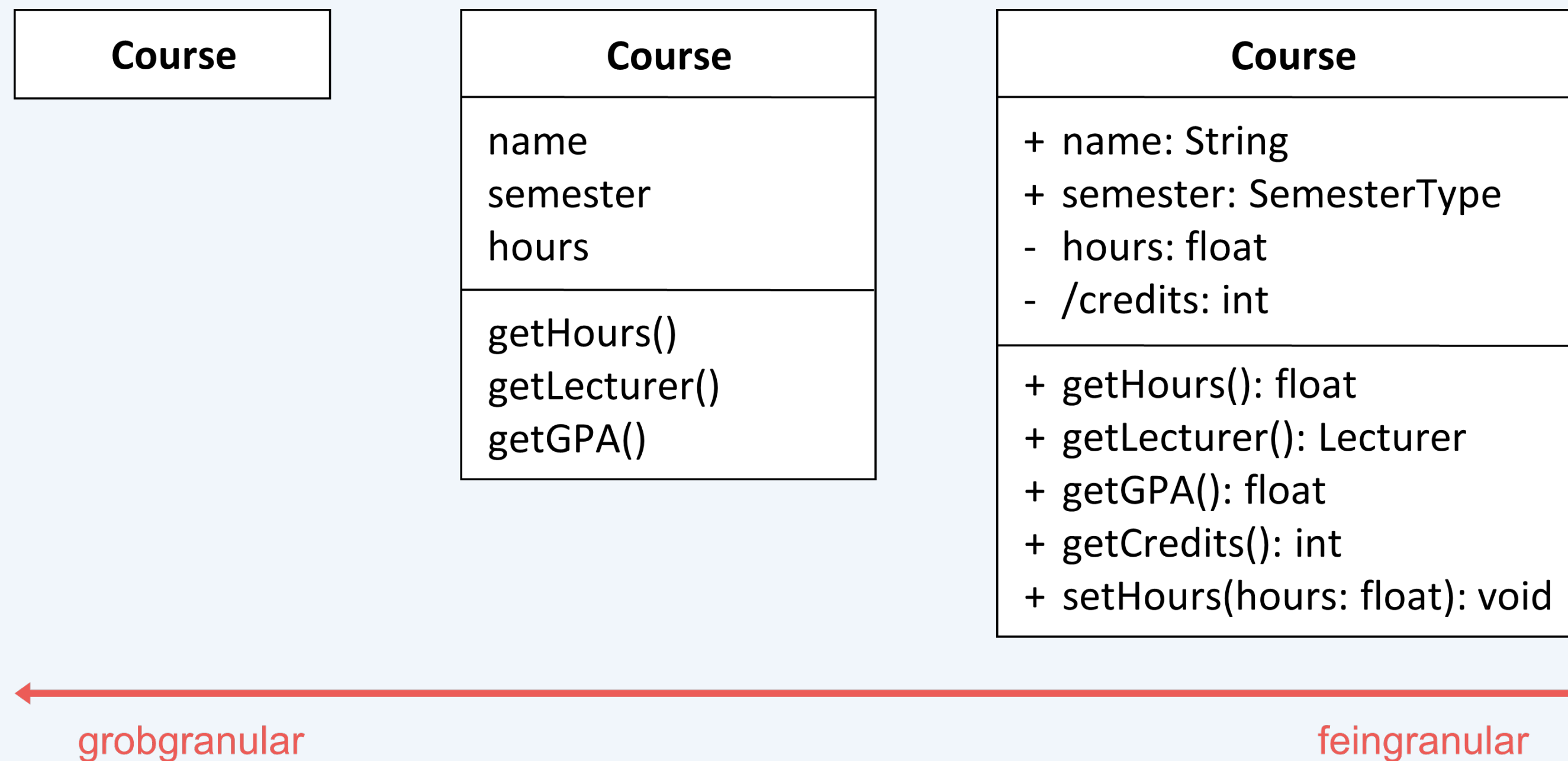
- Instanzvariable (= Instanzattribut)
- Klassenvariable (= Klassenattribut, Statisches Attribut)
 - Werden nur einmal pro Klasse angelegt
 - Z.B. Zähler für die Instanzen einer Klasse, Konstanten, etc.
- Klassenoperation (= Statische Operation)
 - Können verwendet werden, wenn keine Instanz der sie beinhaltenden Klasse erzeugt wurde
 - Z.B. Konstruktoren, mathematische Funktionen, etc.
- Notation: Unterstrichen



```
class Person {
    public String firstName;
    public String lastName;
    private Date dob;
    protected String[] address;
    private static int pNumber;
    public static int getPNumber() {...}
    public Date getDob() {...}
}
```



Spezifikation einer Klasse: Unterschiedlicher Detailgrad



Strukturmodellierung

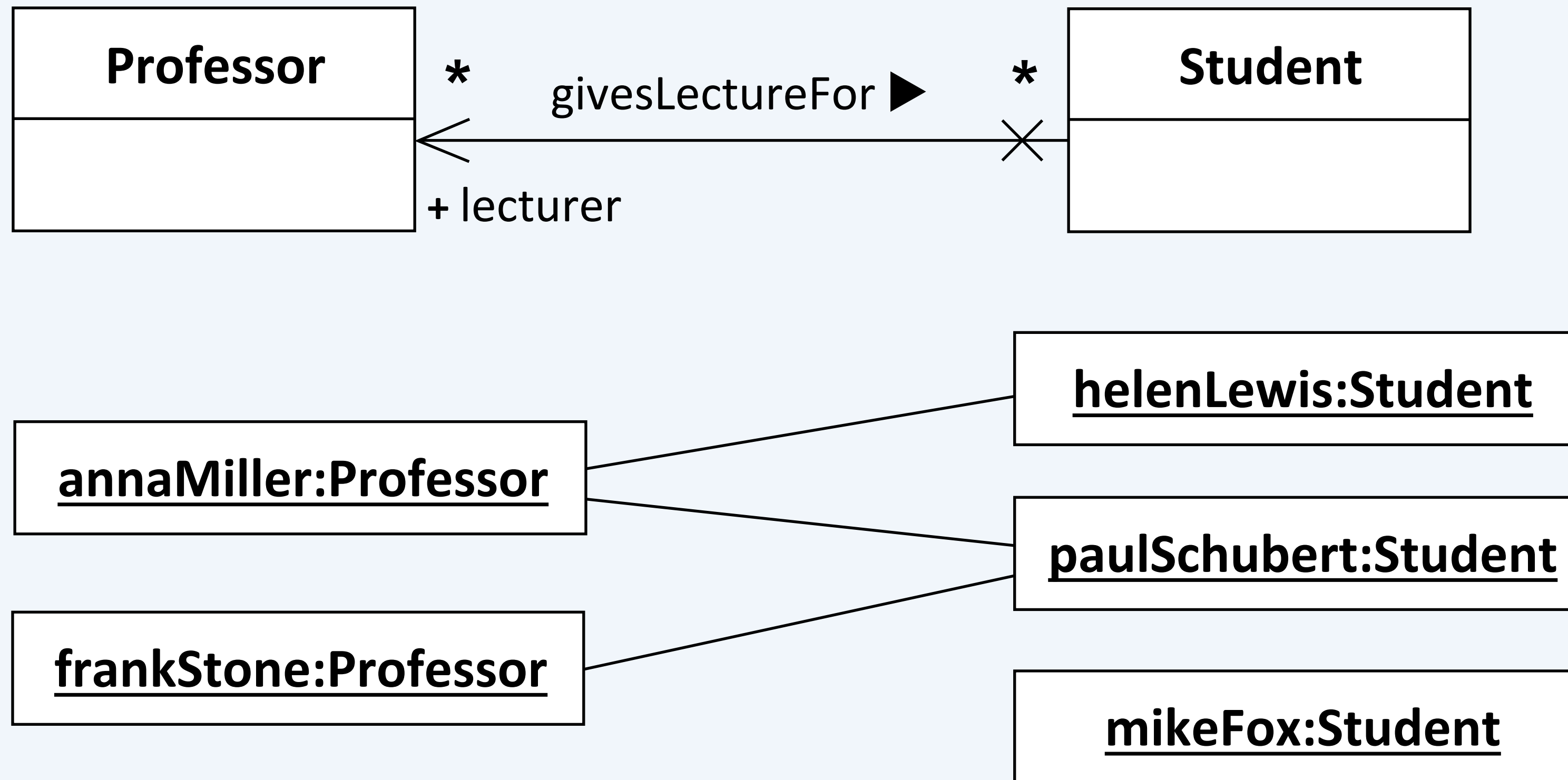
Die Assoziation



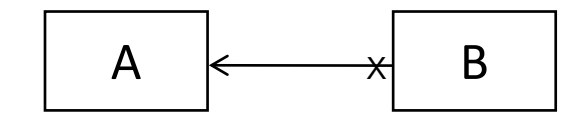
Christian Huemer und Marion Scholz

Assoziation

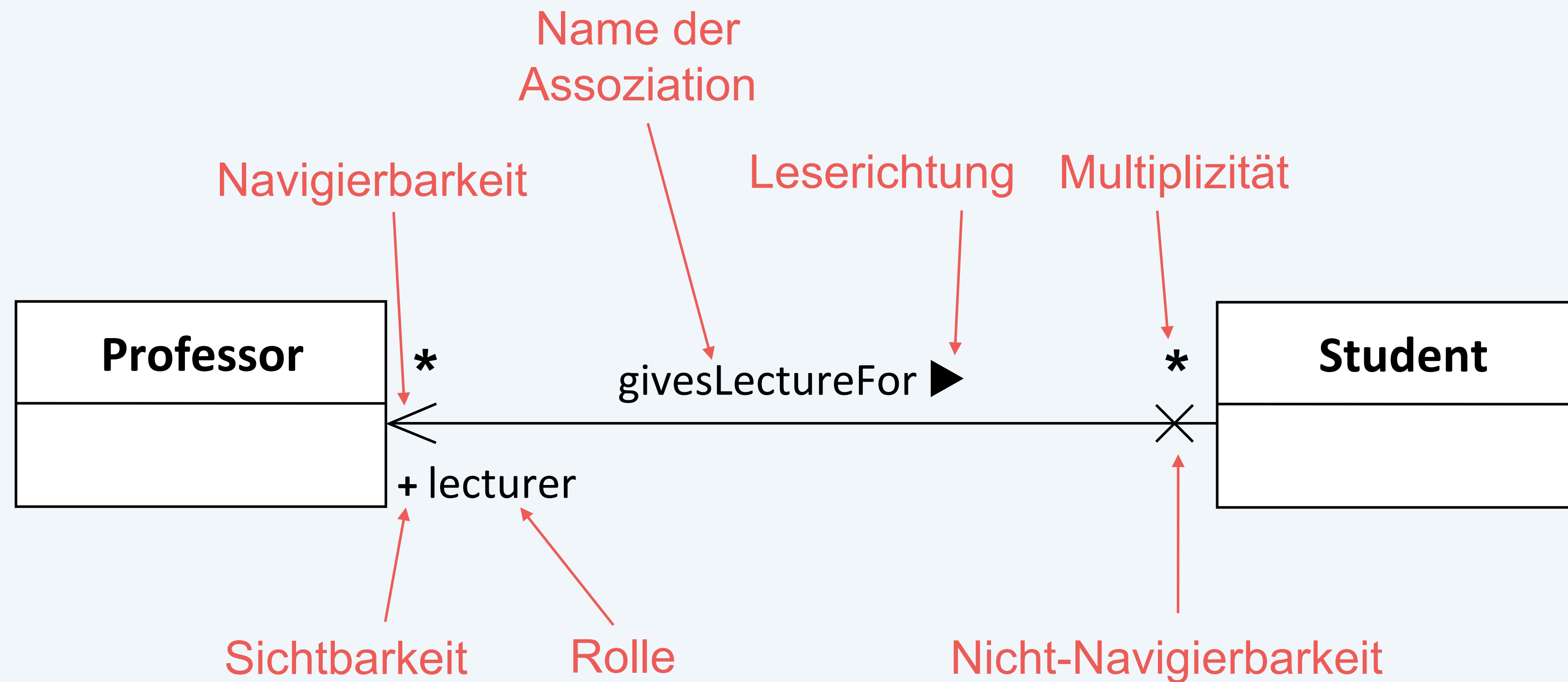
- Assoziationen zwischen Klassen modellieren mögliche **Links** zwischen **den Instanzen der Klassen**



Binäre Assoziation



- Verbindet die Instanzen von zwei Klassen miteinander

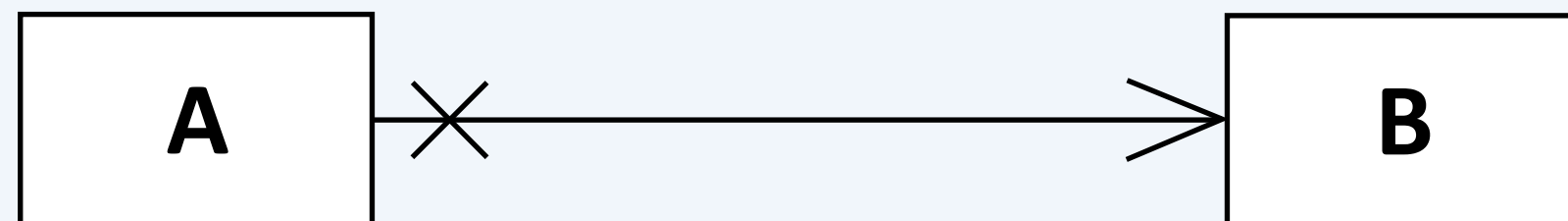


Binäre Assoziation: Navigierbarkeit

- Navigationsrichtungen sind entscheidend für die spätere Entwicklung
- **Navigierbares Assoziationsende: Pfeil**



- **Nicht-navigierbares Assoziationsende: Kreuz**
 - **A** kann auf die sichtbaren Attribute und Operationen von **B** zugreifen
 - **B** kann auf keinerlei Attribute und Operationen von **A** zugreifen

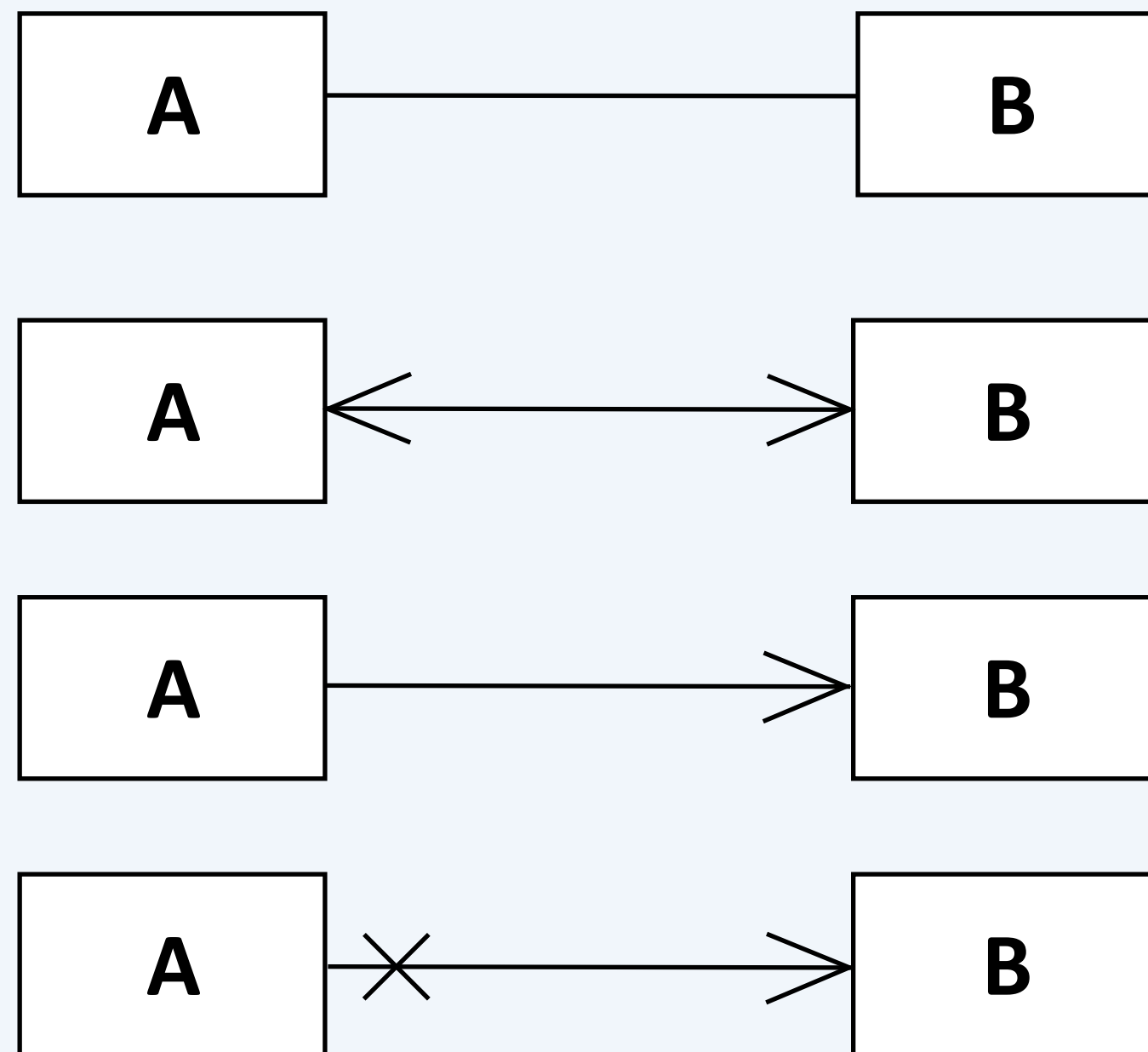


- **Weder Pfeil noch Kreuz: "keine Angabe"**

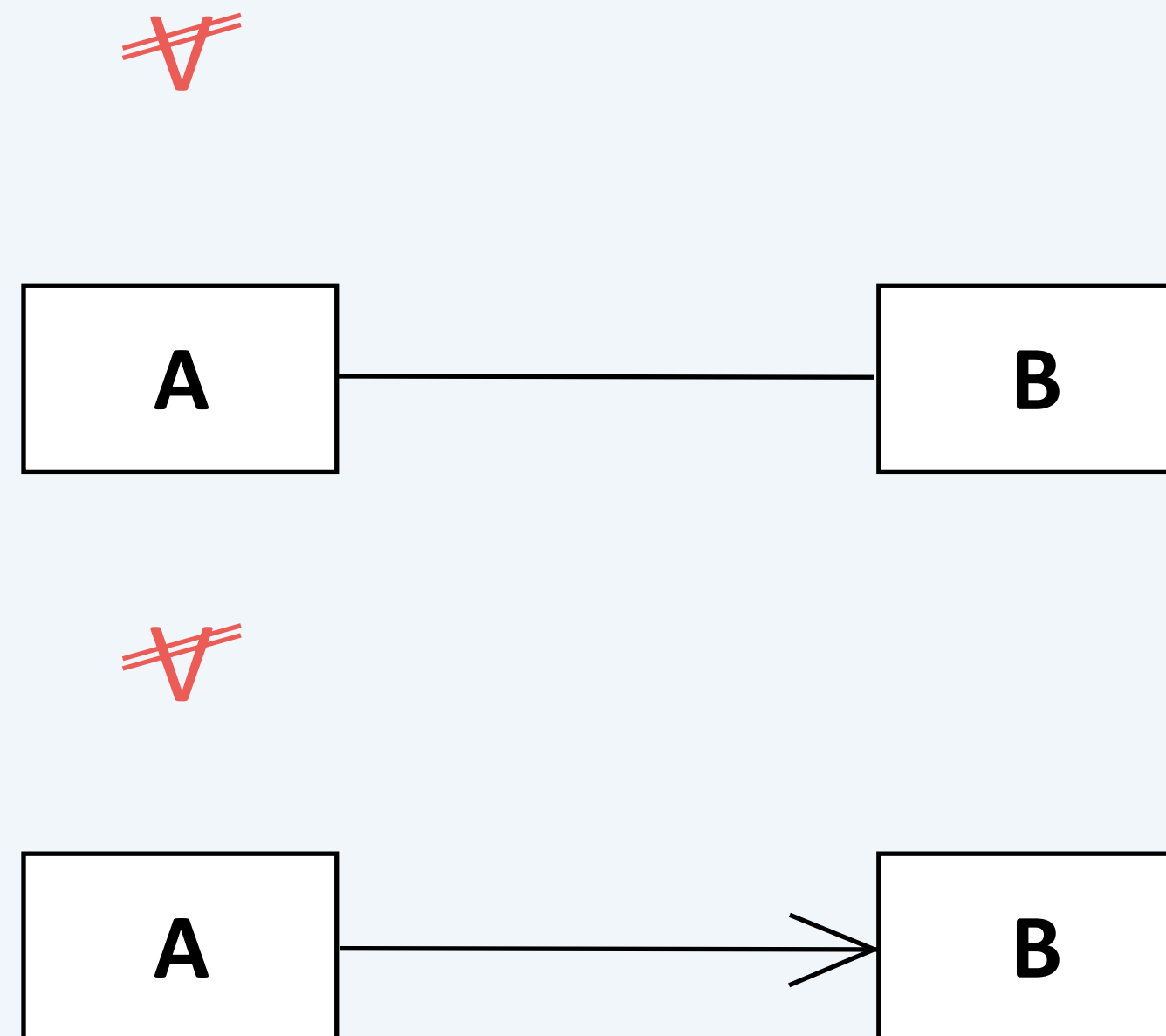


Navigierbarkeit – UML Standard vs. Best Practice

UML Standard



Best Practice

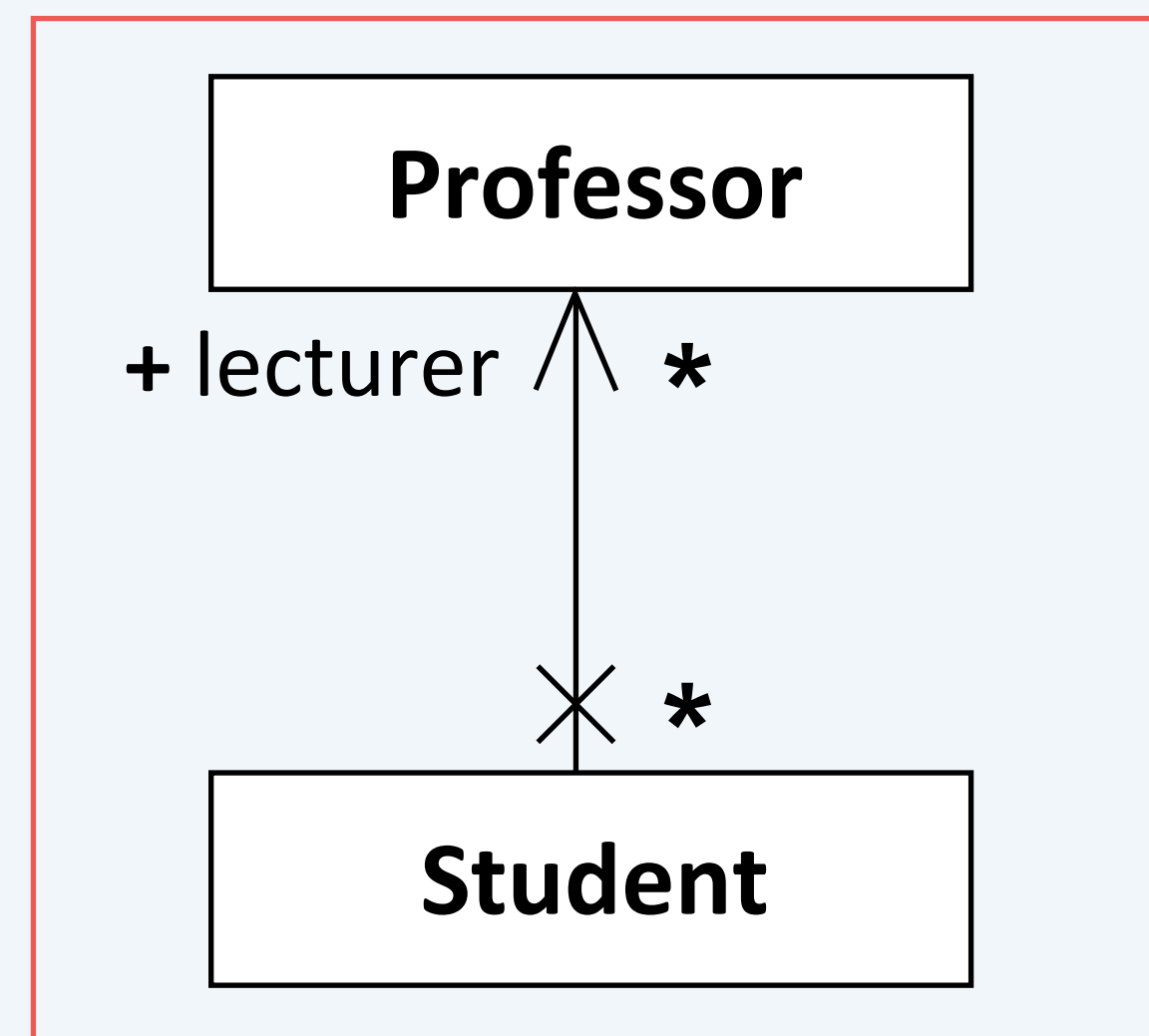


Binäre Assoziation als Attribut

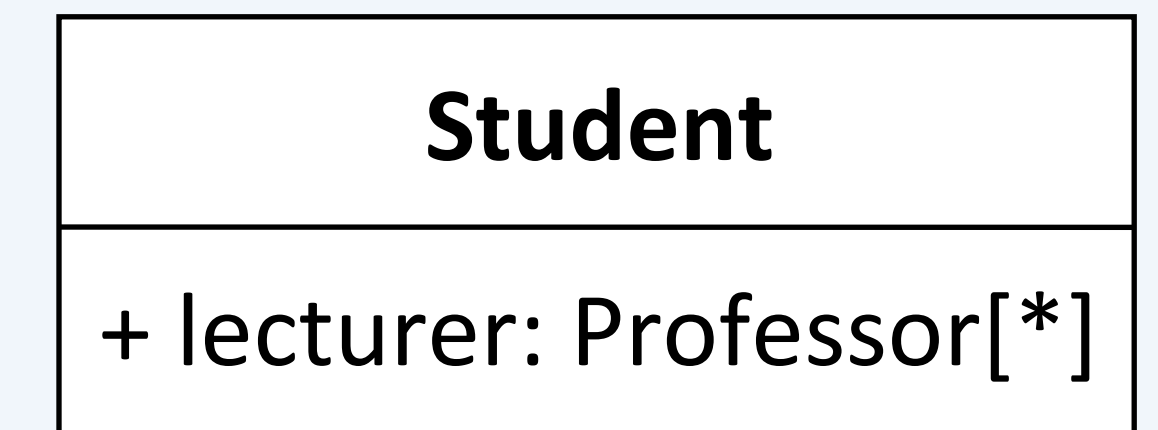
■ Ein navigierbares Assoziationsende

- hat die gleiche Semantik, wie ein Attribut der Klasse am gegenüberliegenden Assoziationsende
- kann daher **anstatt** mit einer **gerichteten Kante** auch als **Attribut** modelliert werden
 - Die mit dem Assoziationsende verbundene Klasse muss dem **Typ** des Attributs entsprechen
 - Die **Multiplizitäten** müssen gleich sein
- Für ein navigierbares Assoziationsende sind somit alle Eigenschaften und Notationen von Attributen anwendbar

```
class Professor{...}  
  
class Student{  
    public Professor[] lecturer;  
    ...  
}
```



besser



Strukturmodellierung

Die Multiplizität und die Rolle



Christian Huemer und Marion Scholz

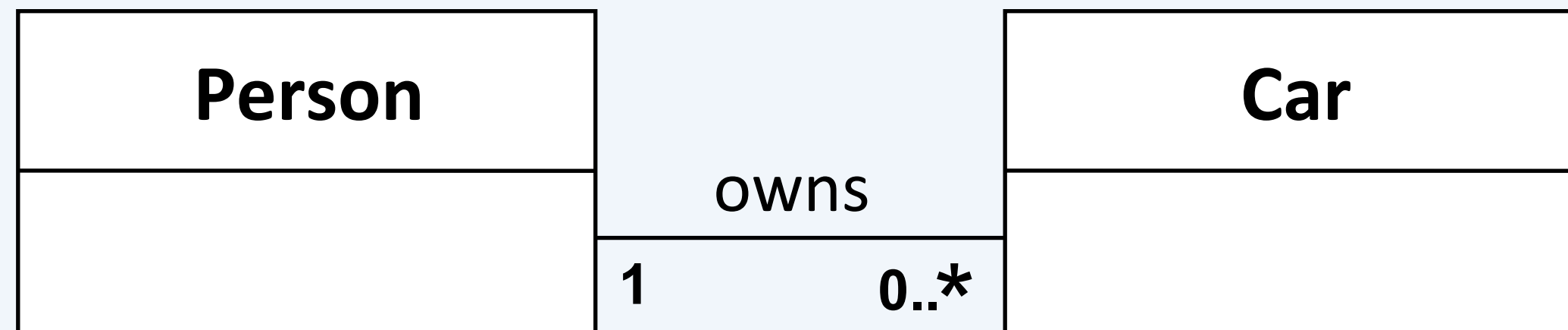
Assoziation: Multiplizität



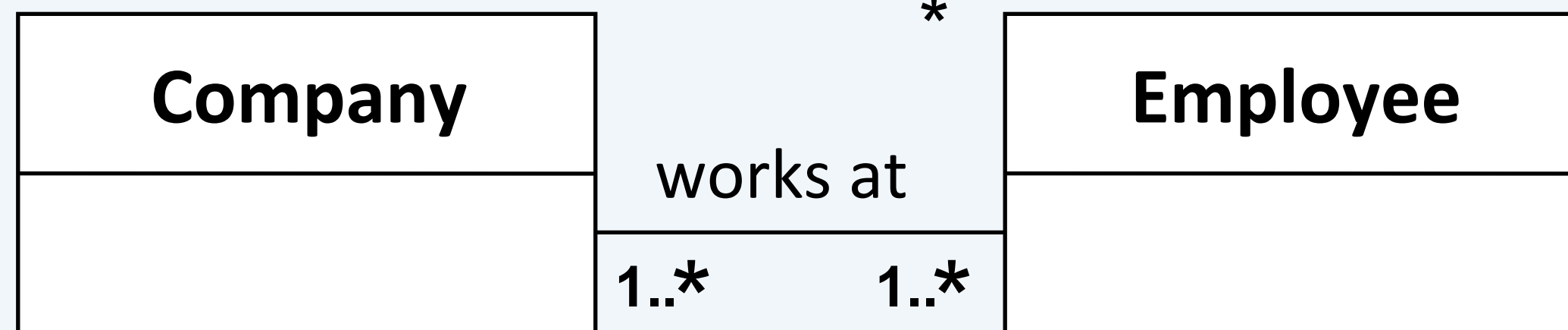
- Bereich: "min .. max"
- Beliebige Anzahl: "*"
- Aufzählung möglicher Kardinalitäten (x, y, z)

genau 1:	1
uneingeschränkt:	* oder 0..*
0 oder 1:	0..1 oder 0, 1
fixe Anzahl (z.B. 3):	3
Bereich (z.B. ≥ 3):	3..*
Bereich (z.B. 3-6):	3..6
Aufzählung:	3, 6, 7, 8, 9 oder 3, 6..9

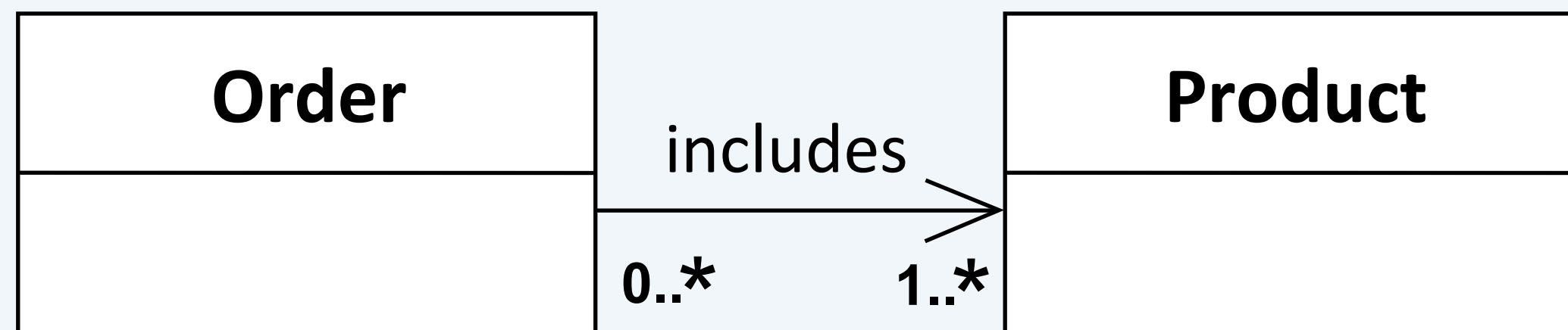
Assoziation: Beispiele



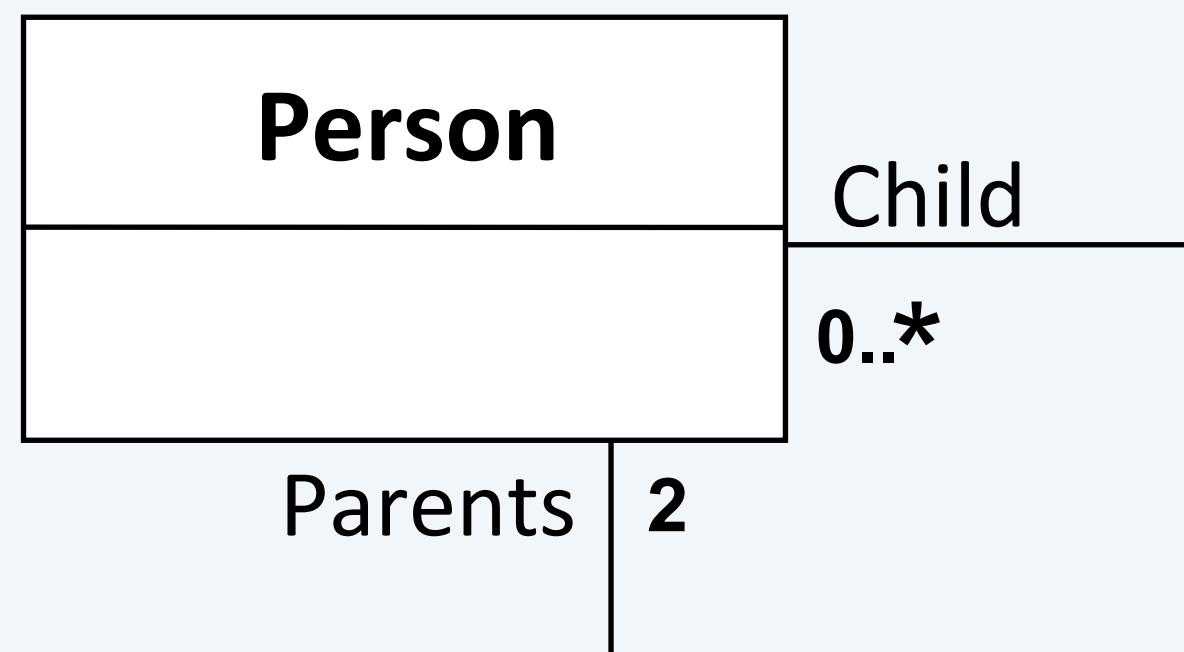
Ein **Car** hat genau einen Besitzer, eine **Person** kann aber mehrere **Cars** besitzen (oder keine).



In einer **Company** arbeitet mind. ein **Employee**, ein **Employee** arbeitet mind. in einer **Company**



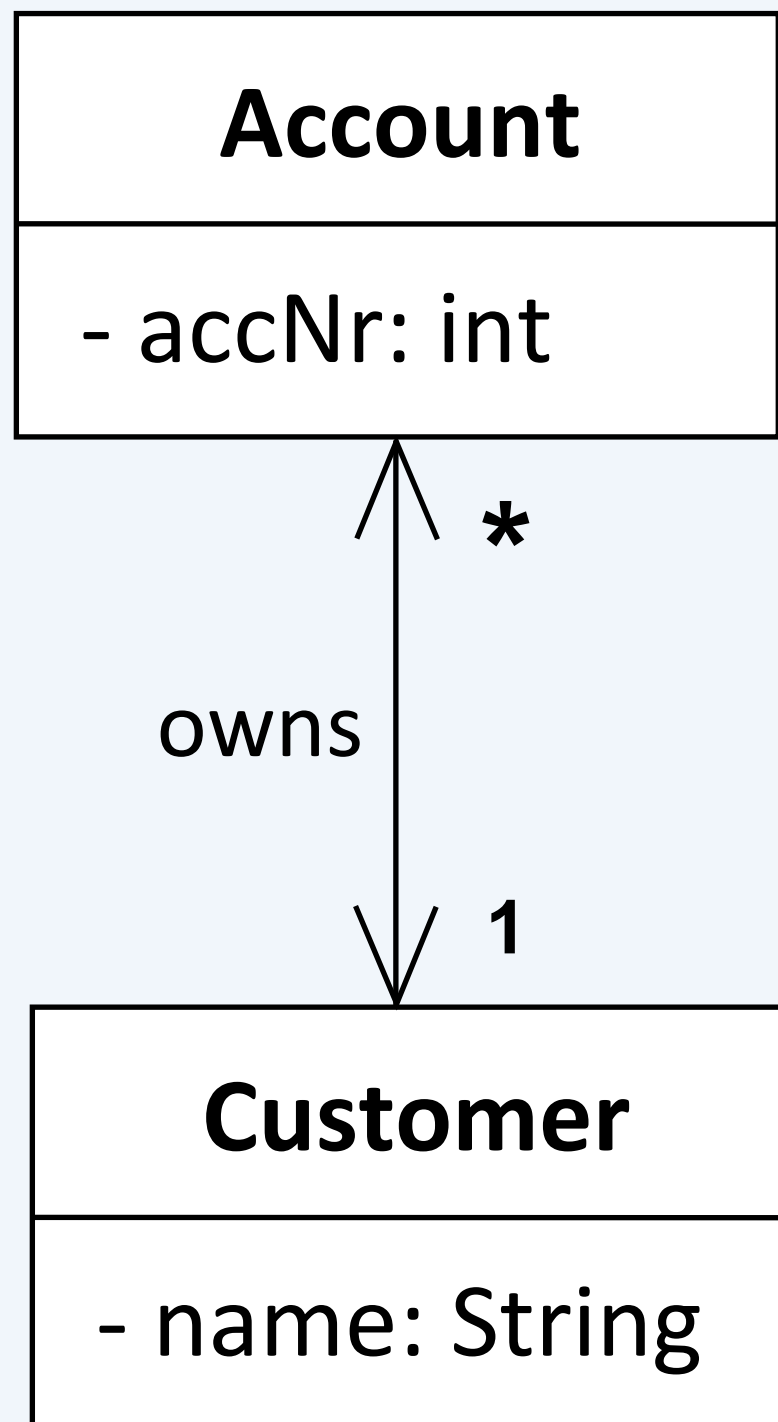
Eine **Order** besteht aus 1-n **Products**, **Products** können beliebig oft bestellt werden. Von einer **Order** kann festgestellt werden, welche **Products** sie beinhaltet.



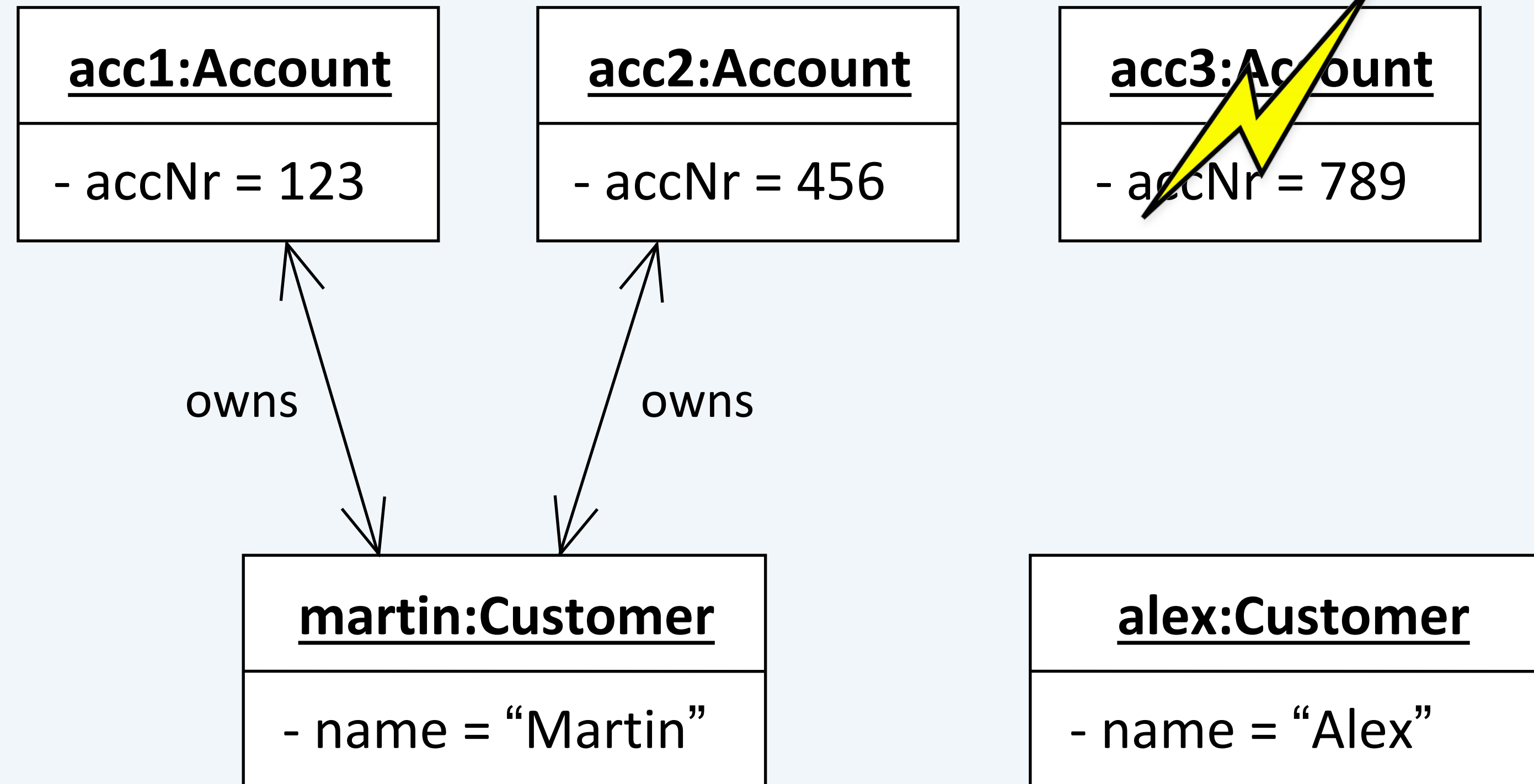
Eine **Person** hat 2 biologische **Parents**, die **Persons** sind, und 0 bis beliebig viele **Children**.
*Ist durch dieses Modell ausgeschlossen, dass eine **Person** **Child** von sich selbst ist?*

Objektdiagramm: Beispiel

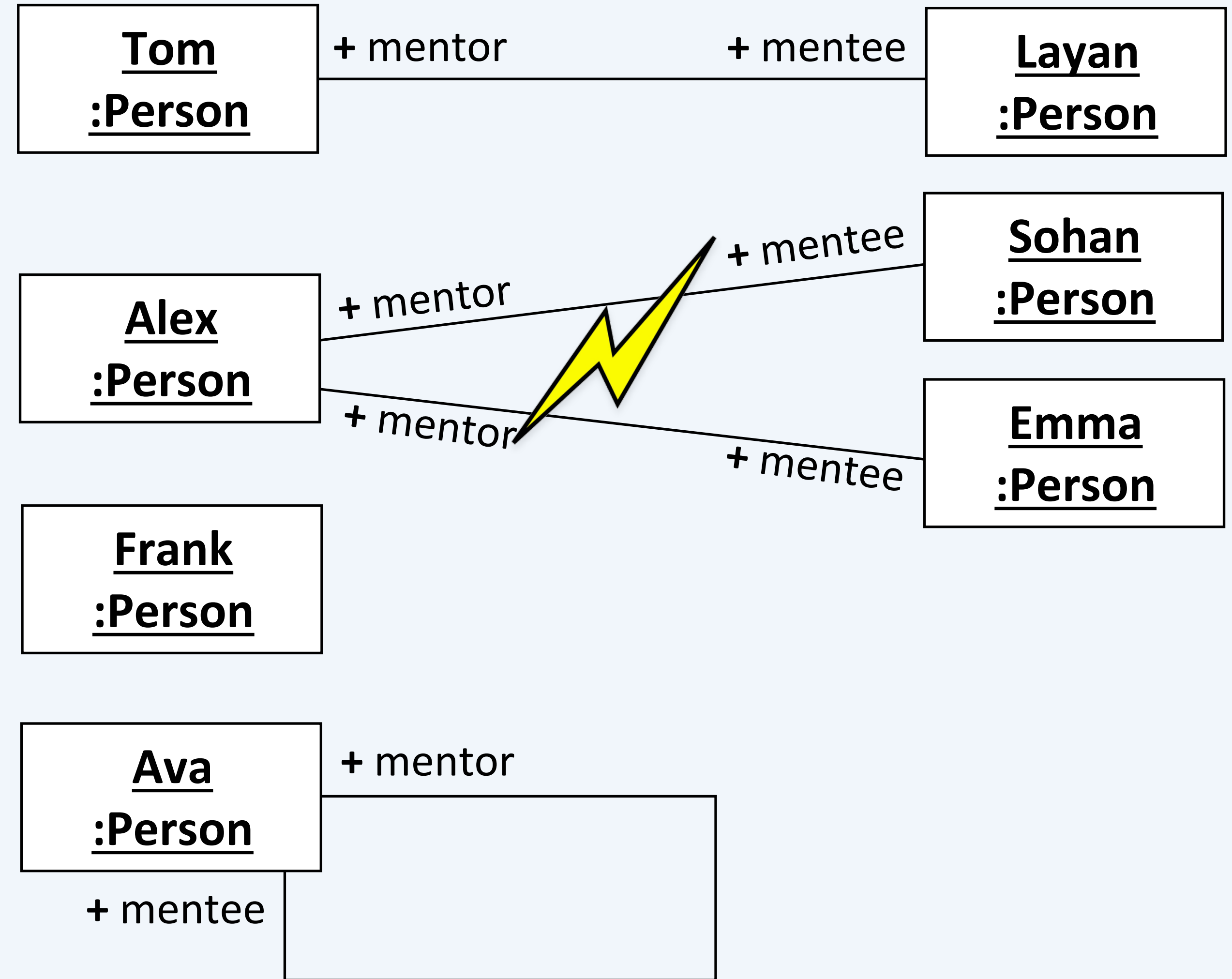
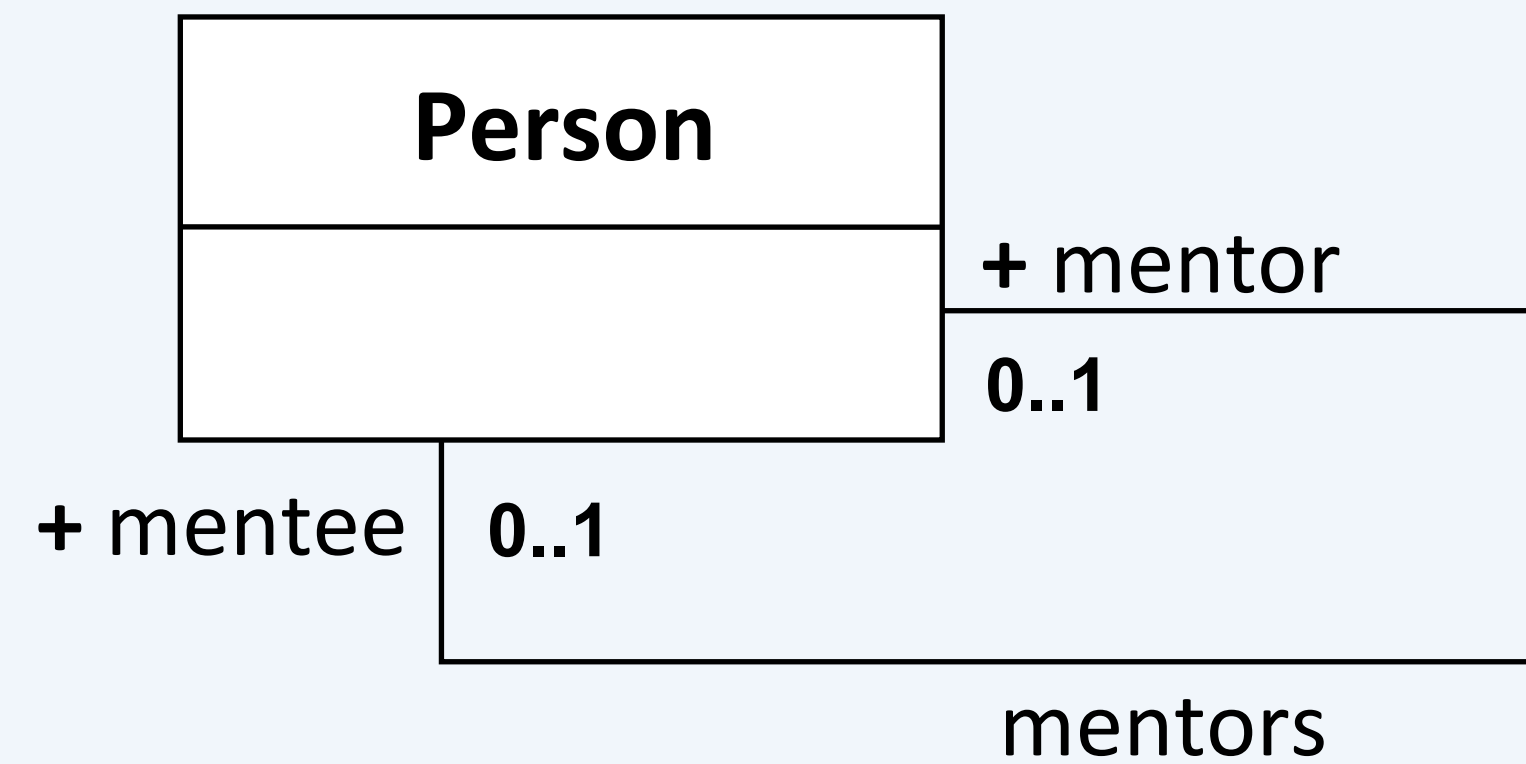
Klassendiagramm



Objektdiagramm

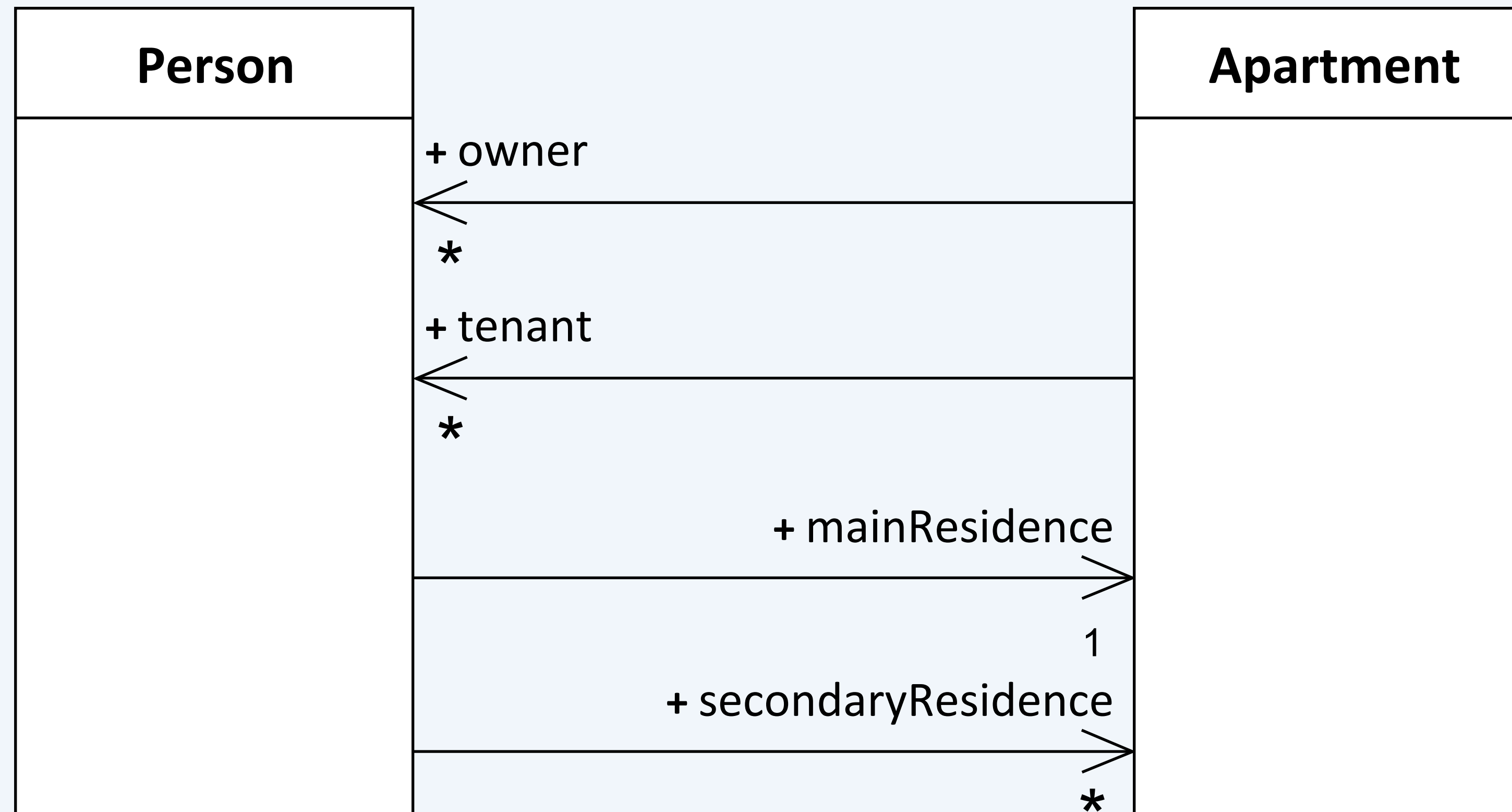


Objektdiagramm: Beispiel bei unärer Assoziation



Assoziation: Rollen

- Es können die Rollen festgelegt werden, die von den einzelnen Objekten in den Objektbeziehungen gespielt werden



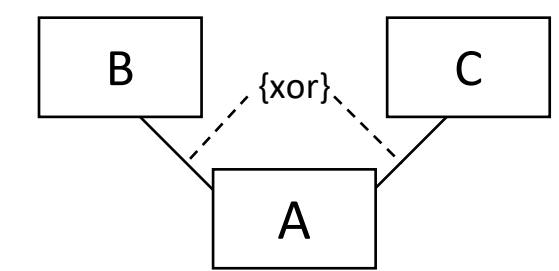
Strukturmodellierung

Die exklusive Assoziation und die Assoziationsklasse

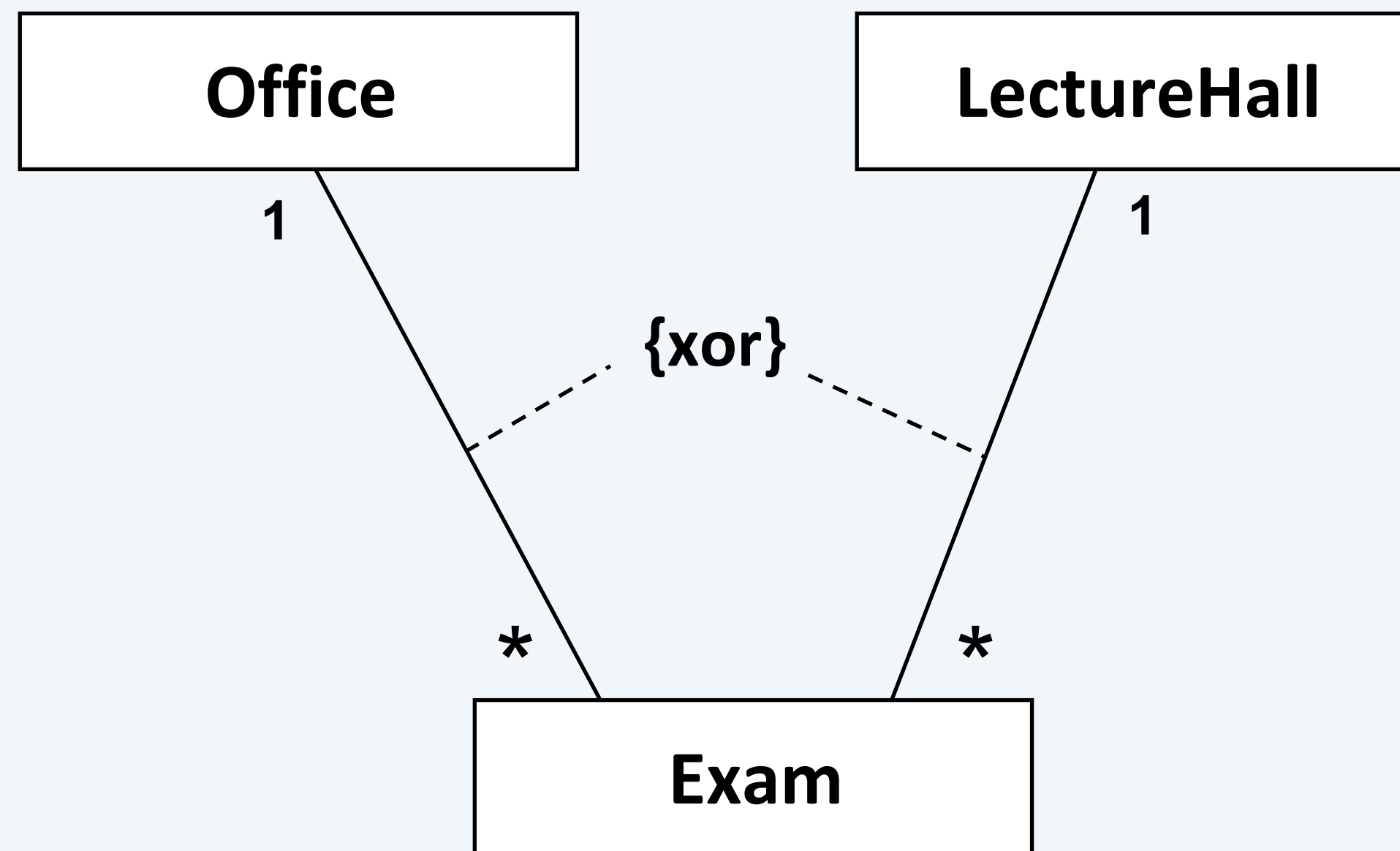


Christian Huemer und Marion Scholz

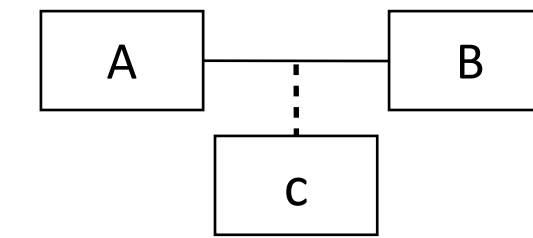
Exklusive Assoziation



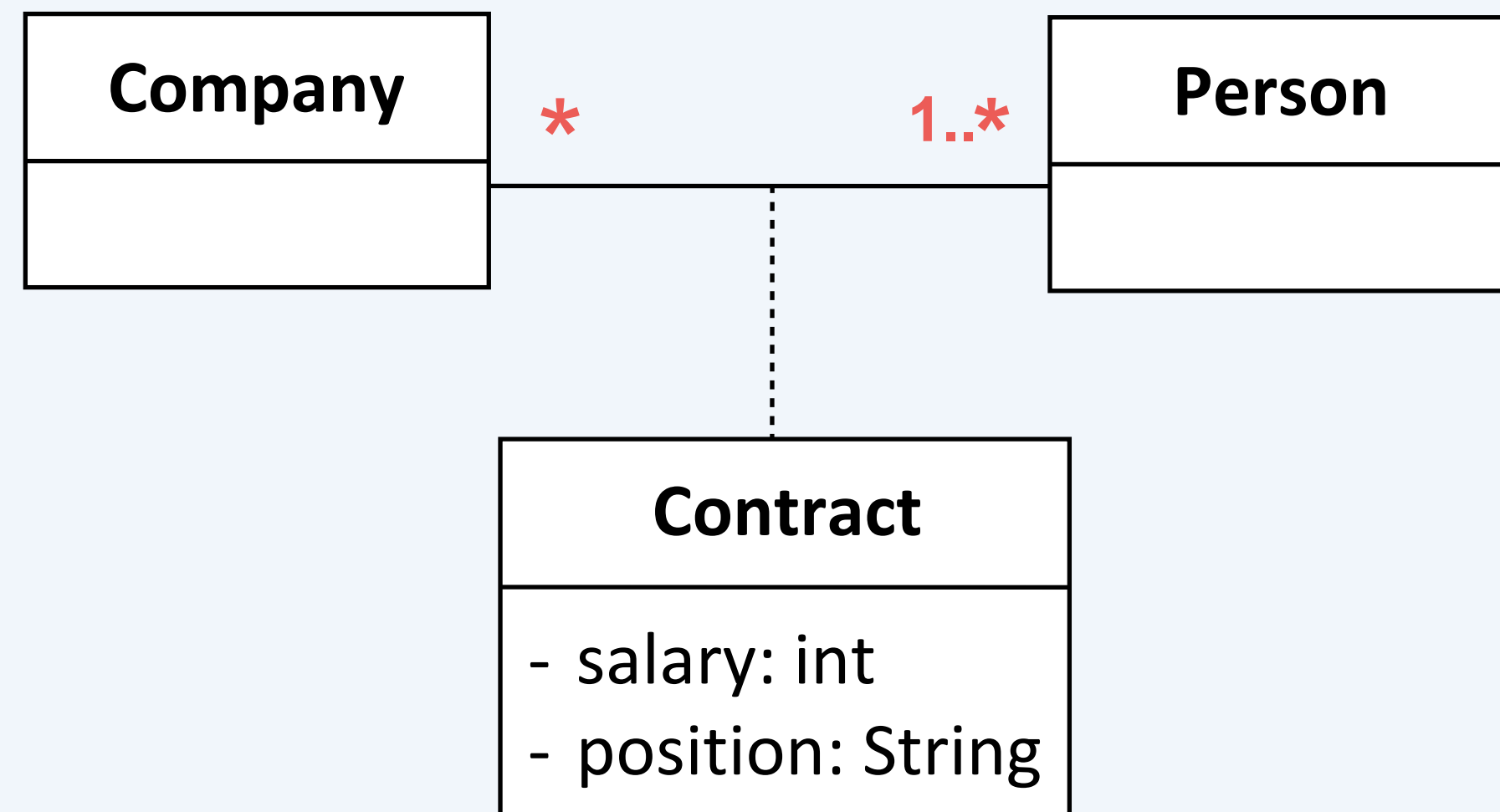
- Für ein bestimmtes Objekt kann zu einem bestimmten Zeitpunkt **nur eine von verschiedenen möglichen Assoziationen** instanziiert werden: **{xor}**



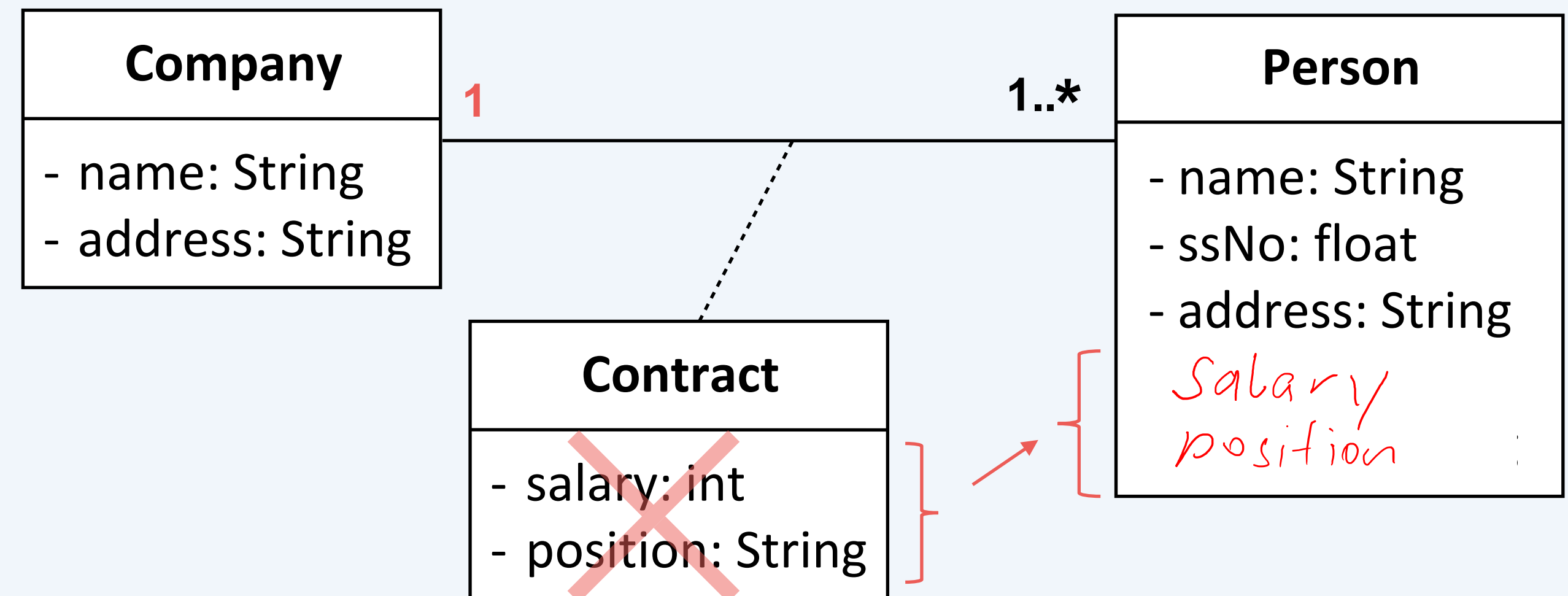
Assoziationsklasse (1/2)



- Kann Attribute der Assoziation enthalten
 - Bei m:n-Assoziationen mit Attributen notwendig
 - Bei 1:1 und 1:n-Assoziationen sinnvoll aus Flexibilitätsgründen

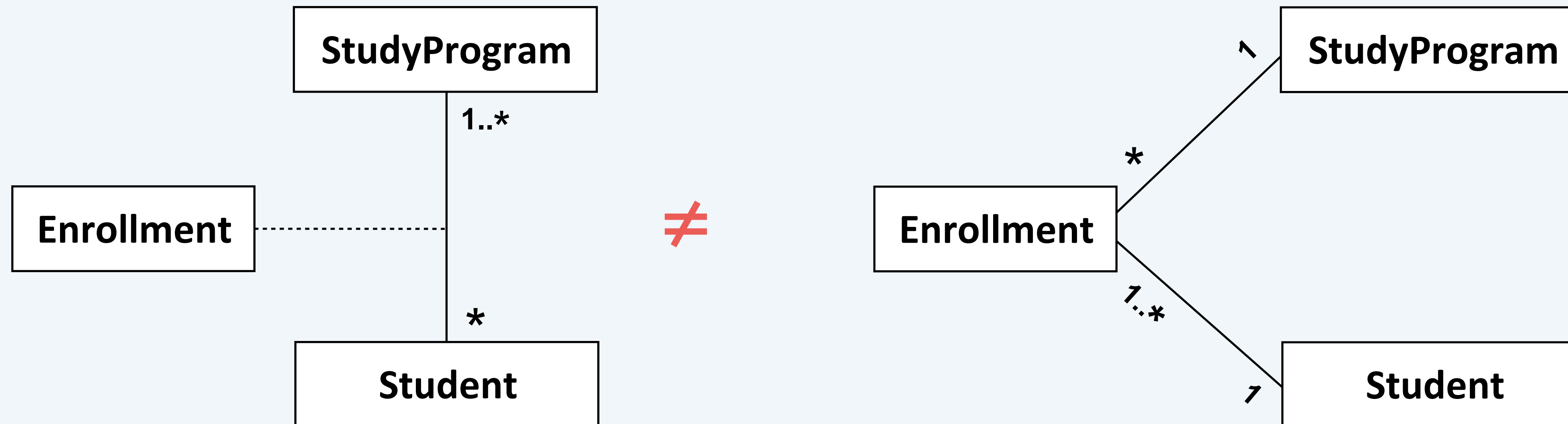


Assoziationsklasse



Assoziationsklasse (2/2)

- Normale Klasse ungleich Assoziationsklasse



Ein **Student** kann sich für ein bestimmtes *StudyProgram* nur **einmal** einschreiben

Ein **Student** kann **mehrere** *Enrollments* für ein und dasselbe *StudyProgram* haben

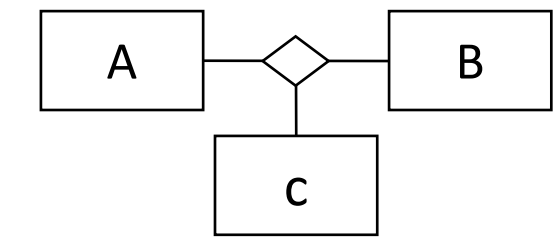
Strukturmodellierung

Die n-äre Assoziation



Christian Huemer und Marion Scholz

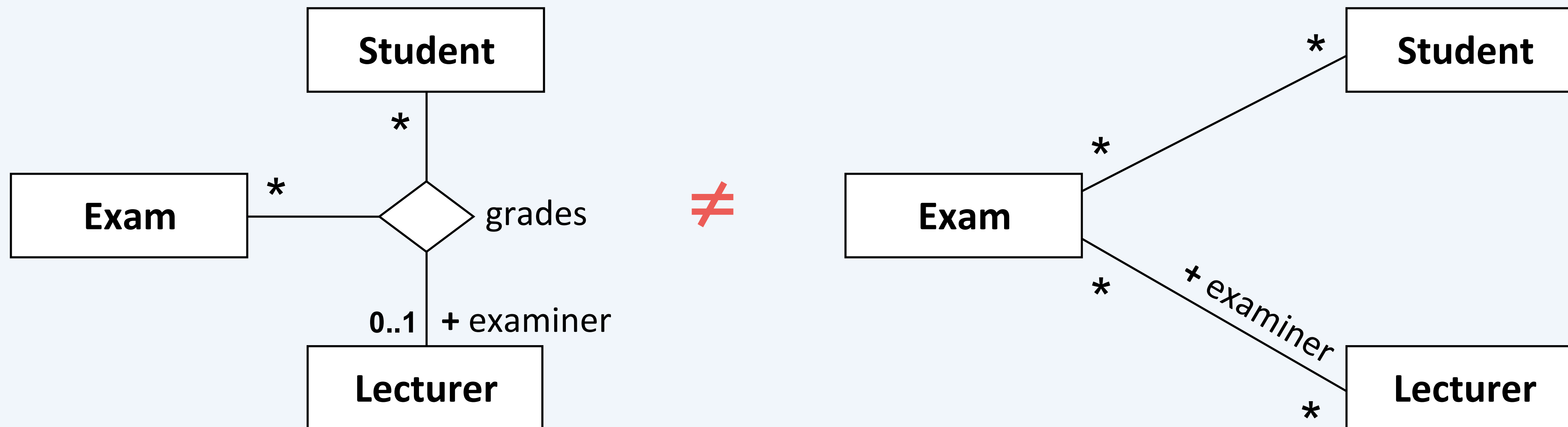
n-äre Assoziation



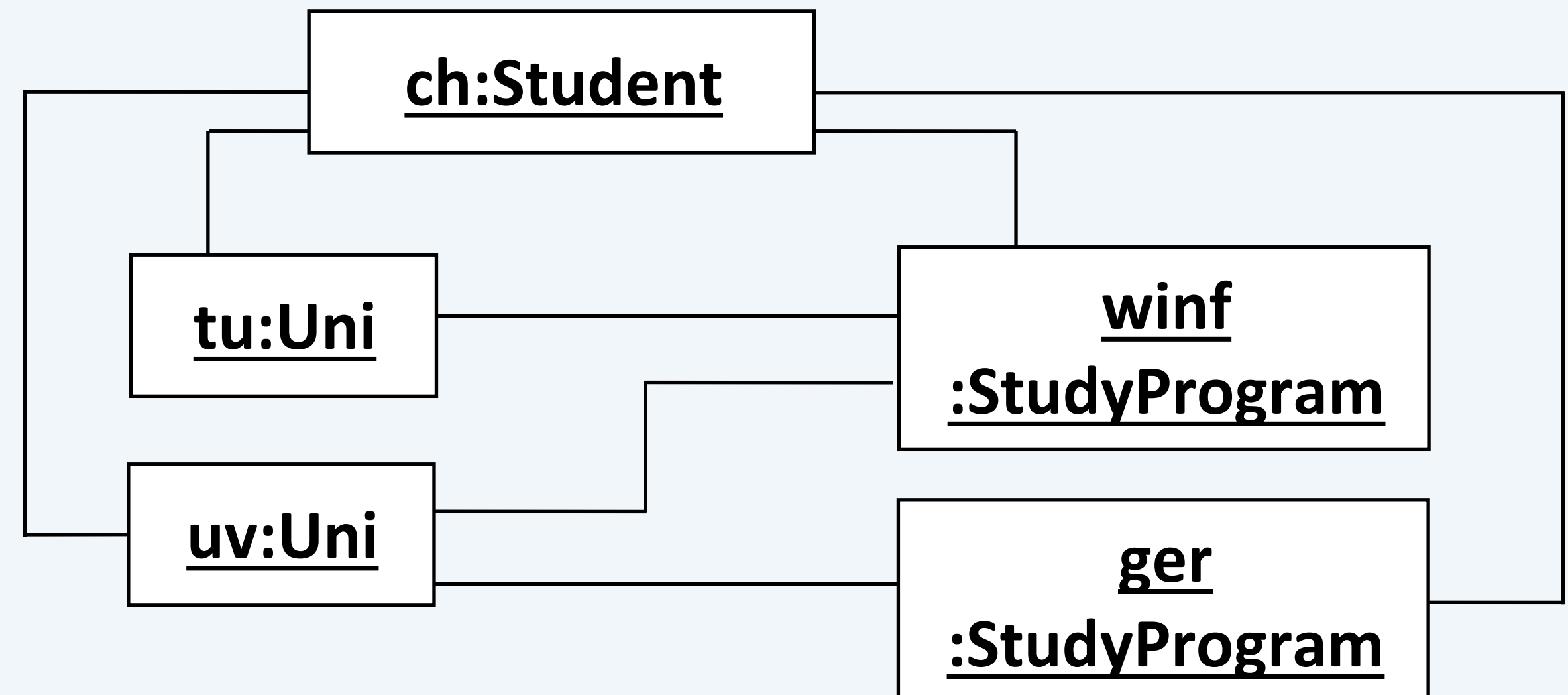
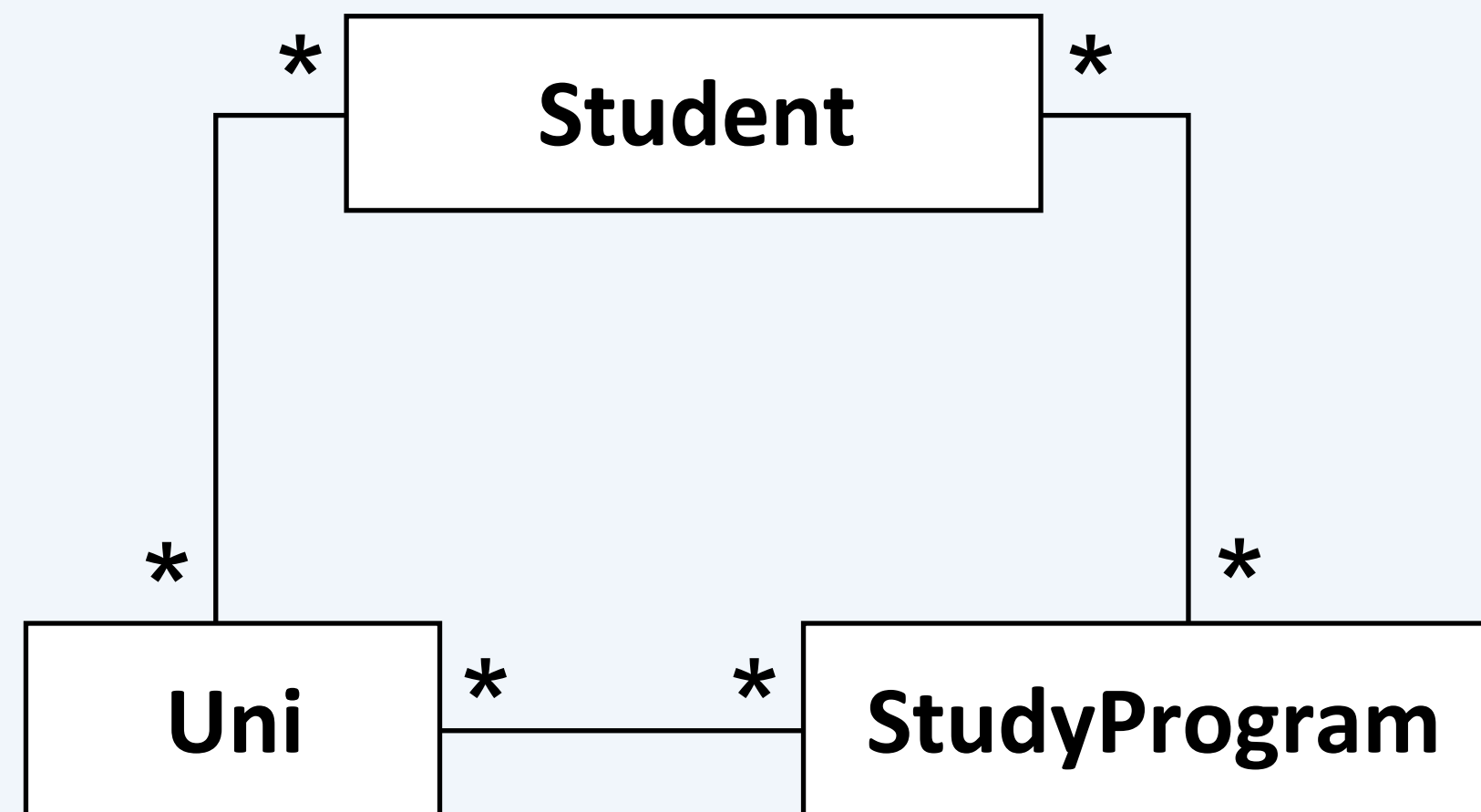
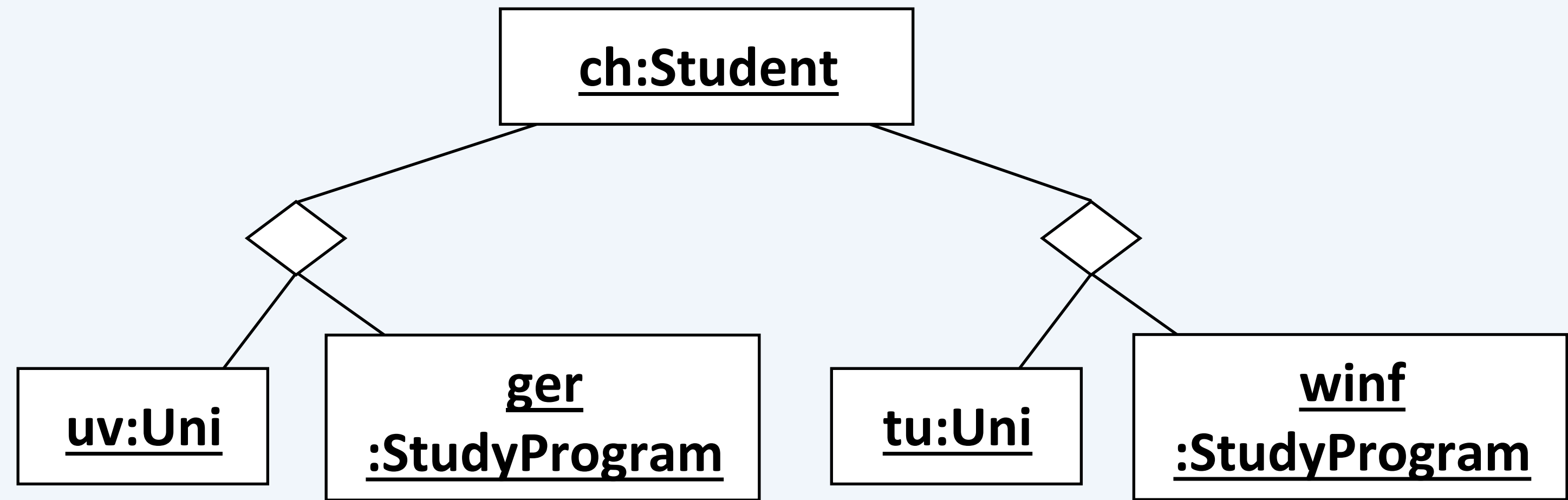
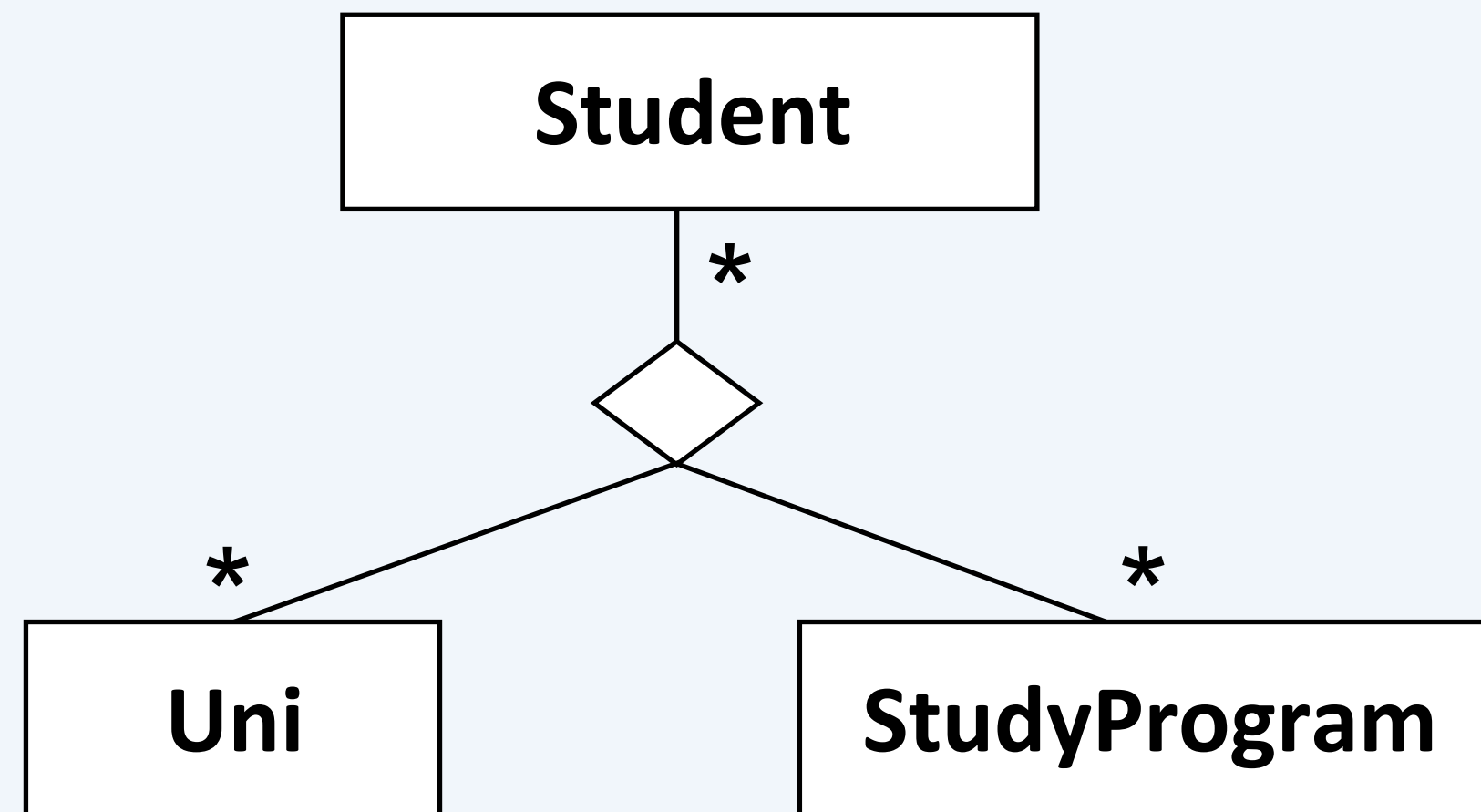
- **Beziehung zwischen mehr als zwei Klassen**
 - Navigationsrichtung kann nicht angegeben werden
 - N Sätze zu Bestimmung der Multiplizitäten
 - Eine bestimmte Kombination von Objekten aller anderen Klassen stehen in Beziehung zu wie vielen Objekten dieser Klasse.
- **Multiplizitäten implizieren Einschränkungen,**
in einem bestimmten Fall funktionale Abhängigkeiten
- Ist bei einer ternären Assoziation die Multiplizität bei der Klasse C mit 1 angegeben, so existiert eine funktionale Abhängigkeit $(A, B) \rightarrow (C)$

n-äre Assoziation: Beispiel

- $(\text{Student}, \text{Exam}) \rightarrow (\text{Lecturer})$
 - Ein **Student** absolviert ein **Exam** bei einem oder keinem **Lecturer**
- $(\text{Exam}, \text{Lecturer}) \rightarrow (\text{Student})$
 - Ein **Exam** kann mit einem **Lecturer** von mehreren **Students** absolviert werden
- $(\text{Student}, \text{Lecturer}) \rightarrow (\text{Exam})$
 - Ein **Student** kann von einem **Lecturer** für mehrere **Exams** bewertet werden



ternäre Assoziation versus binäre Assoziation



Strukturmodellierung

Die Aggregation



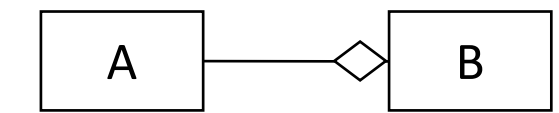
Christian Huemer und Marion Scholz

- Aggregation ist eine spezielle Form der Assoziation für Ganzes-Teil-Beziehungen

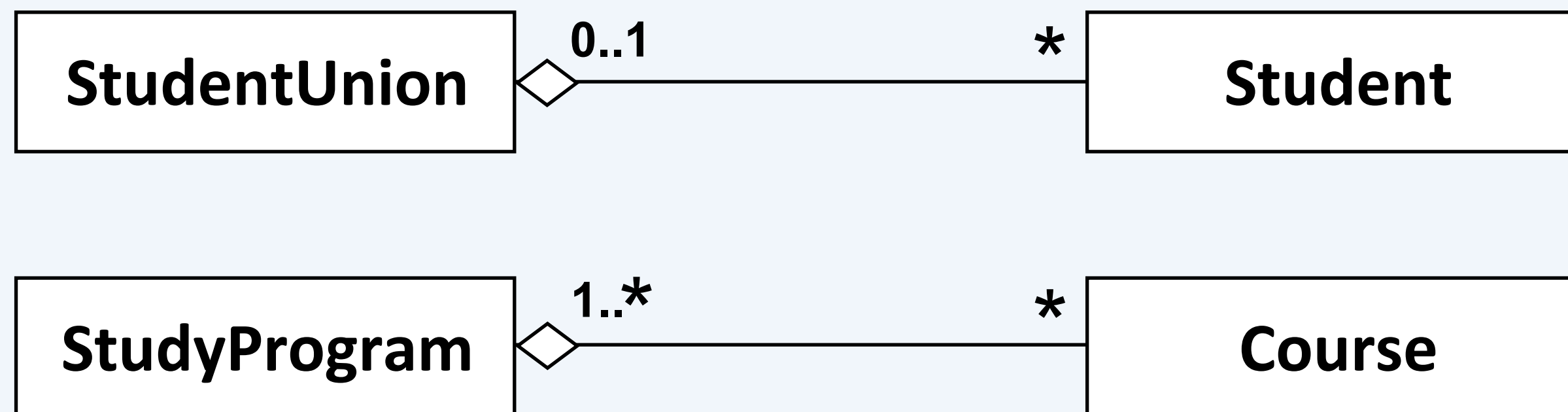
- Zwei Arten von Aggregationen:
 - Schwache Aggregation (shared aggregation)
 - Starke Aggregation – Komposition (composite aggregation)

- Für beide gelten folgenden Eigenschaften:
 - Transitivität:
C ist Teil von B u. B ist Teil von A \Rightarrow C ist Teil von A
 - Anti-Symmetrie:
B ist Teil von A \Rightarrow A ist nicht Teil von B

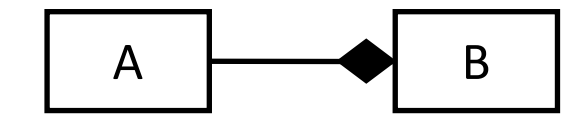
Schwache Aggregation



- Wird als weiße Raute dargestellt
- Schwache Zugehörigkeit der Teile, d.h. Teile sind **unabhängig** von ihrem Ganzen
- Die Multiplizität auf Seite des Ganzen kann > 1 sein
- Es gilt nur eingeschränkte Propagierungssemantik
- Die zusammengesetzten Objekte bilden einen **gerichteten, azyklischen Graphen**



Starke Aggregation (= Komposition)



ingo

- Ein bestimmter Teil darf zu einem bestimmten Zeitpunkt in **maximal einem zusammengesetzten Objekt enthalten** sein
- Die **Multiplizität** des aggregierenden Endes der Beziehung kann (maximal) 1 sein
- **Abhängigkeit** der Teile vom zusammengesetzten Objekt
- **Propagierungssemantik**
- Die zusammengesetzten Objekte bilden einen **Baum**
- Es kann eine Hierarchie von "Teil-von" - Beziehungen dargestellt werden (Transitivität!)

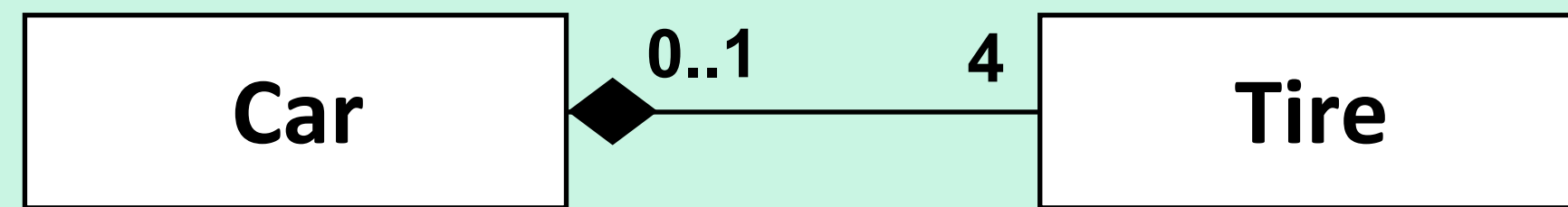


Wenn das `Building` gelöscht wird, wird die `LectureHall` auch gelöscht.

Der `Beamer` kann ohne `LectureHall` existieren, aber wenn er in der `LectureHall` enthalten ist während diese gelöscht wird, dann wird der `Beamer` auch gelöscht.

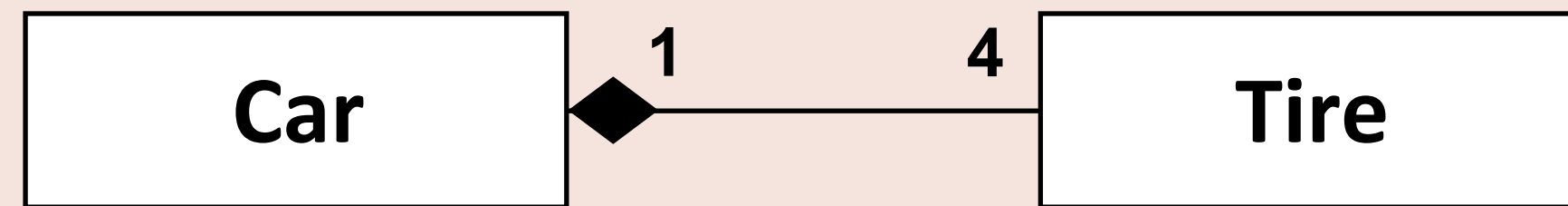
Komposition und Aggregation

- Welche der folgenden Beziehungen trifft zu?



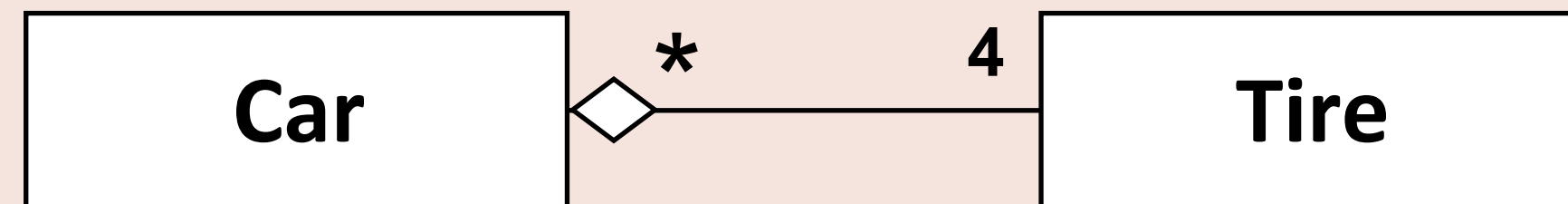
Ein Reifen kann auch ohne Auto existieren. Ein Reifen gehört zu maximal einem Auto.

JA



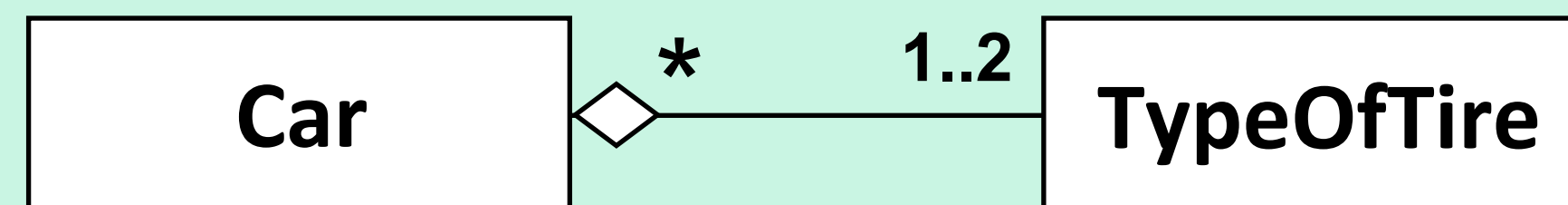
Ein Reifen kann nicht ohne Auto existieren.

NEIN



Ein Reifen kann Bestandteil mehrerer Autos sein.

NEIN



Ein Auto hat 1 oder 2 Typen von Reifen. Mehrere Autos können den selben Reifentyp haben.

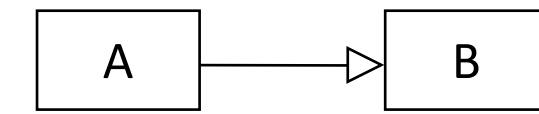
JA

Strukturmodellierung Die Generalisierung

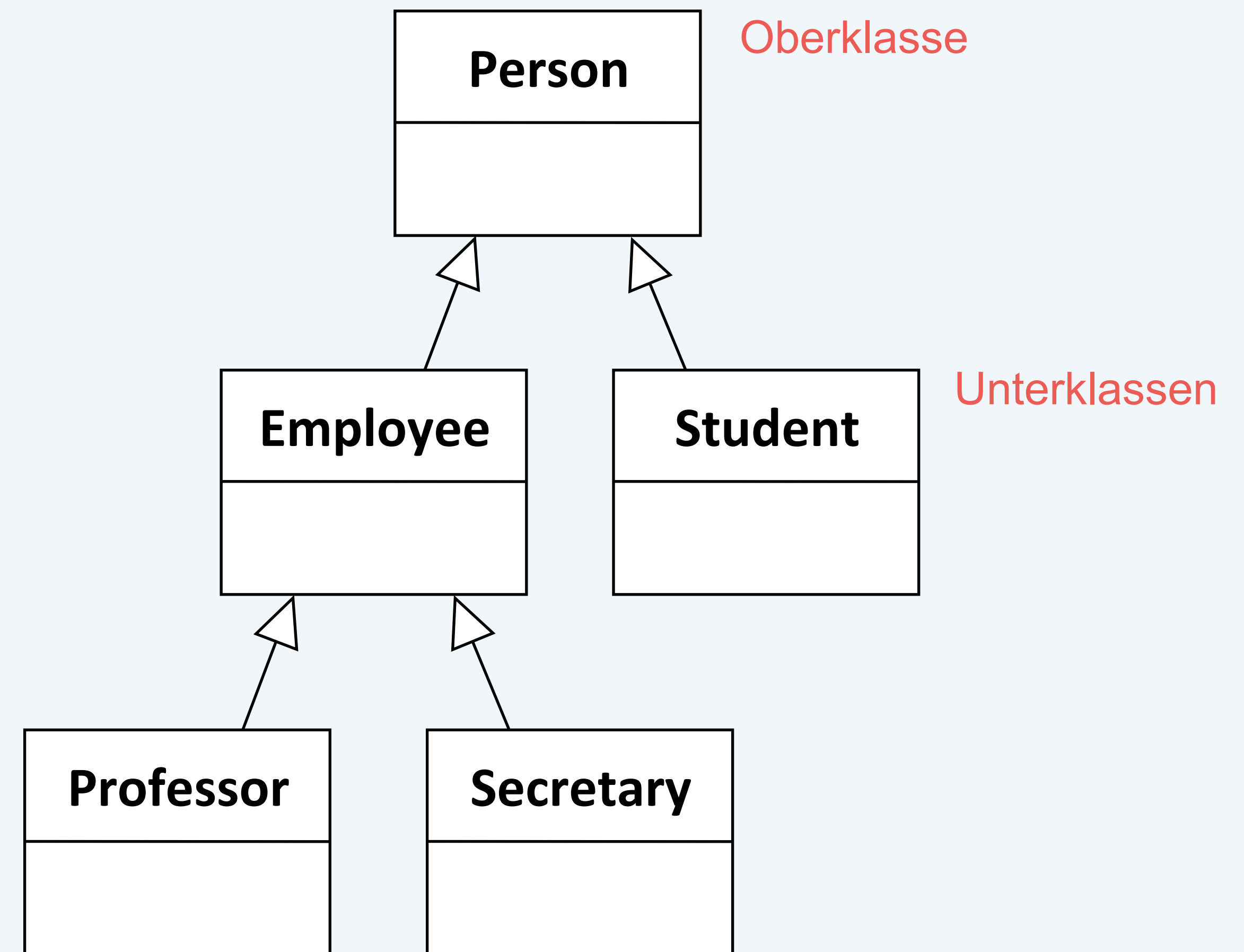


Christian Huemer und Marion Scholz

Generalisierung



- **Taxonomische Beziehung** zwischen einer allgemeineren Klasse und einer spezielleren Klasse
 - Die Unterklasse **erbt** die Eigenschaften der Oberklasse
 - Kann **weitere Eigenschaften** hinzufügen
 - Eine Instanz der Unterklasse kann überall dort verwendet werden, wo eine Instanz der Oberklasse erlaubt ist (zumindest syntaktisch)
- Mittels Generalisierung wird eine Hierarchie von "ist-ein" Beziehungen dargestellt (Transitivität!)



Secretary ist ein
Employee und eine
Person

Abstrakte Klasse (1/2)

{abstract}
A



- Klasse, die **nicht instanziiert** werden kann
- **Nur in Generalisierungshierarchien** sinnvoll
- Dient zum "**Herausheben**" **gemeinsamer Merkmale** einer Reihe von Unterklassen
- Notation: Schlüsselwort **{abstract}** oder Klassenname in kursiver Schrift

{abstract}
Person

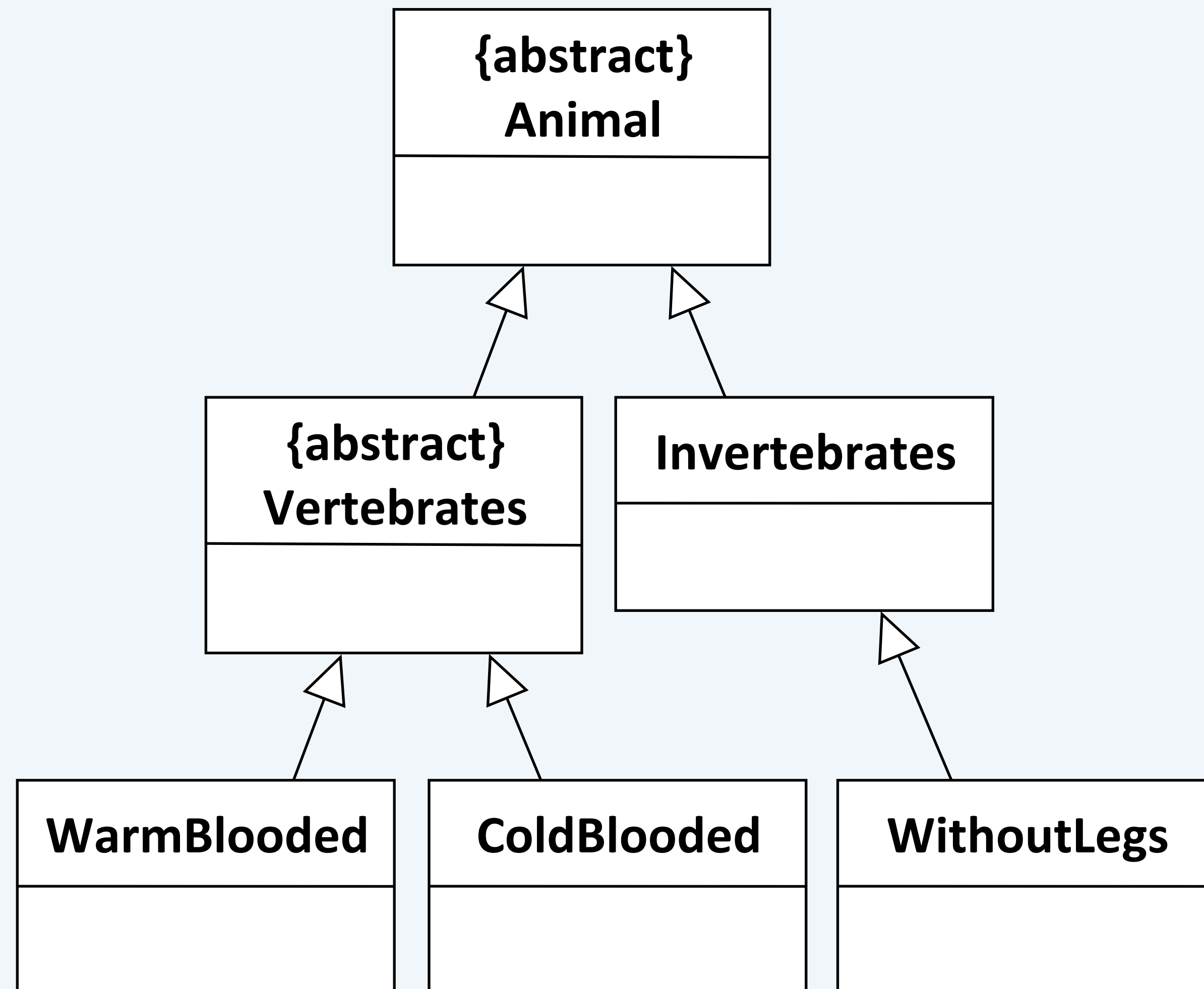
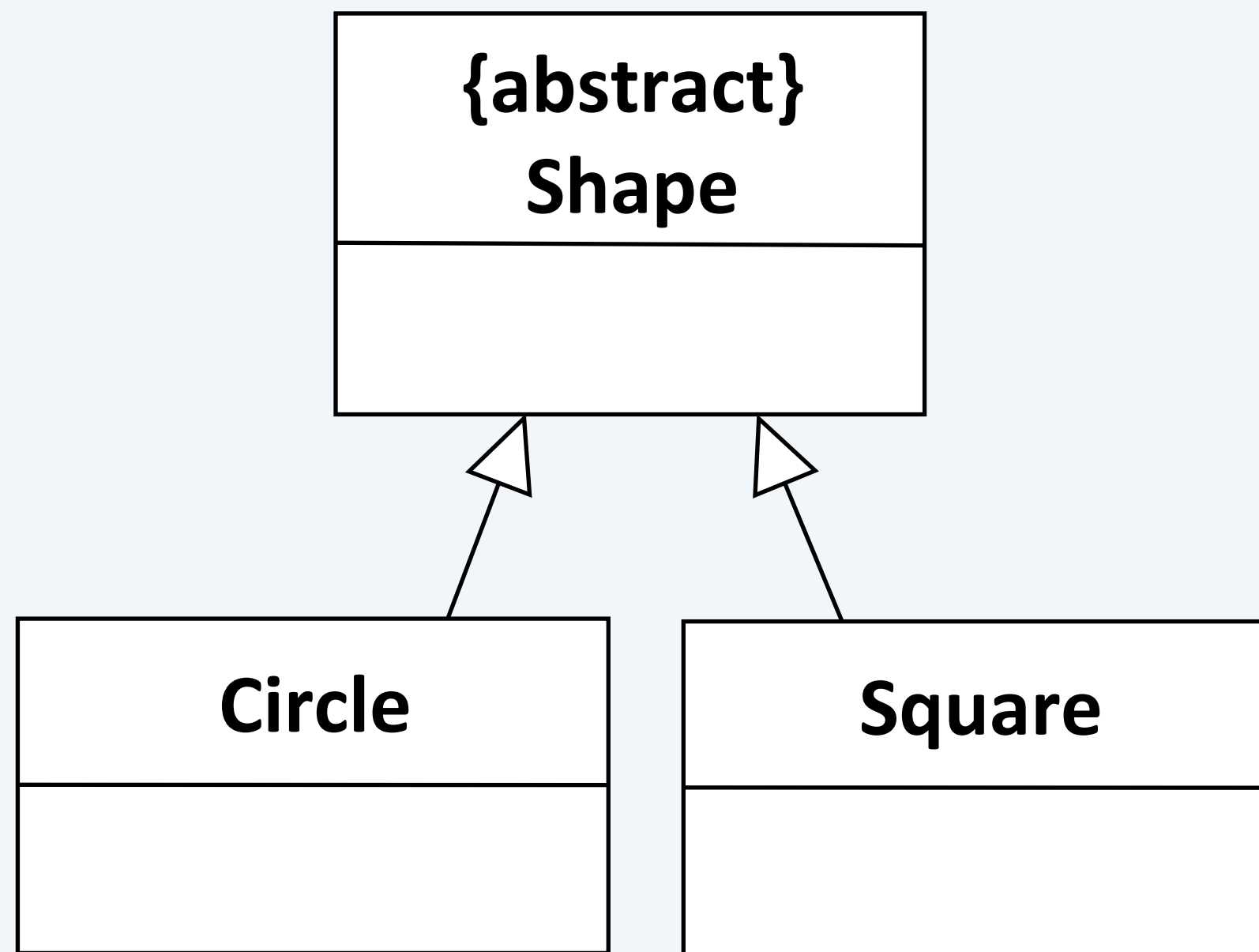
oder

Person

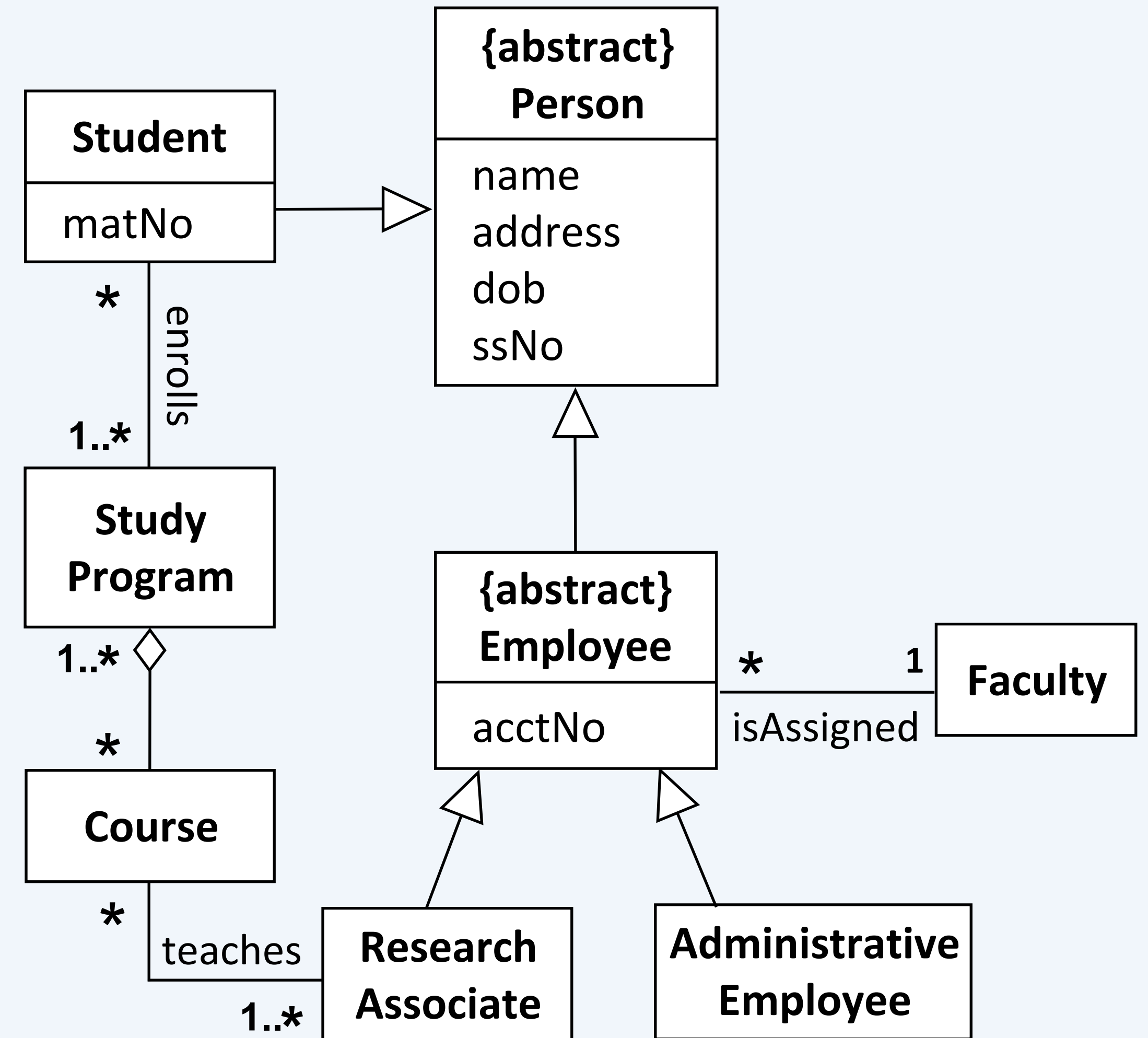
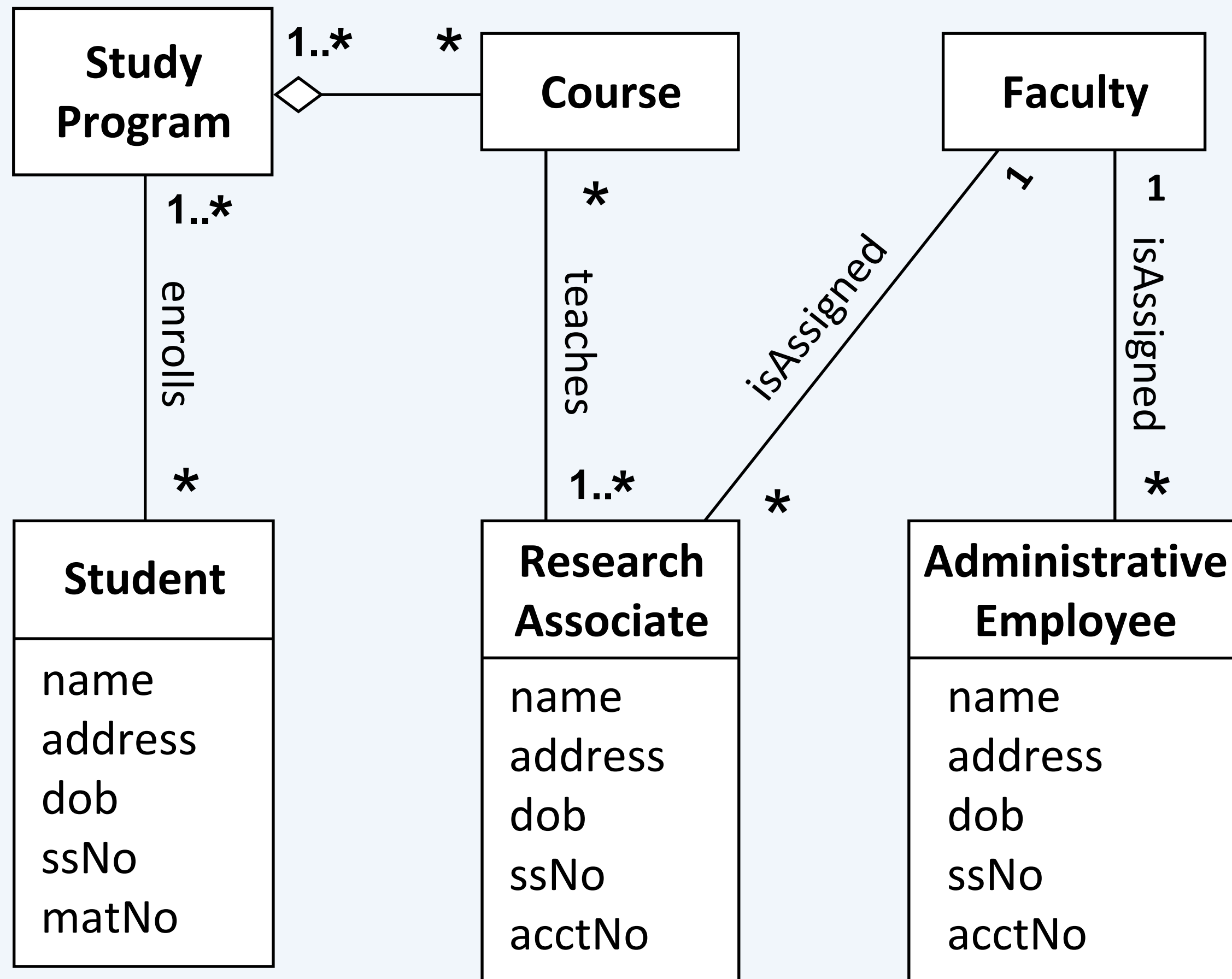
- Unterscheidung zwischen konkreten und abstrakten Operationen einer Klasse funktioniert analog

Abstrakte Klasse (2/2)

■ Beispiele:

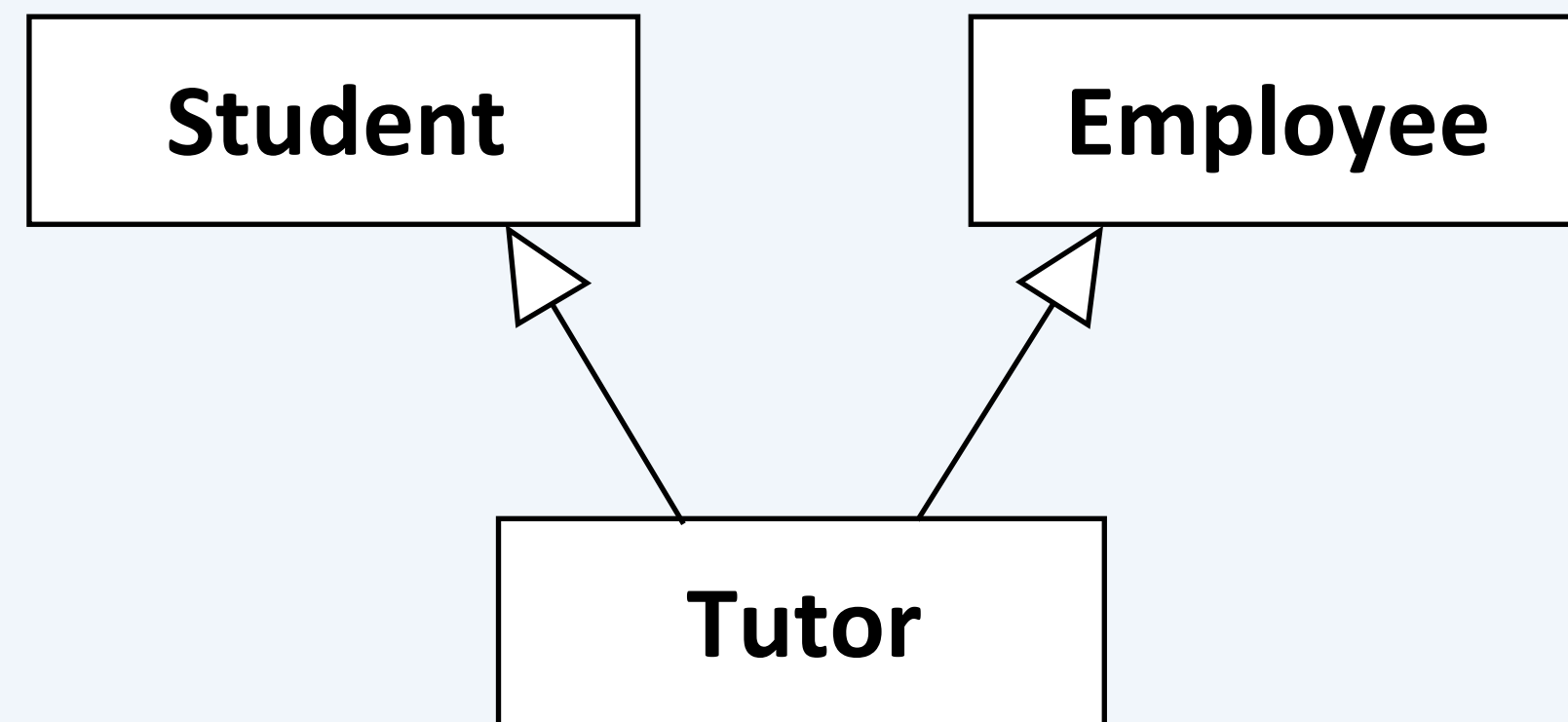


Mit und ohne Generalisierung



Mehrfachvererbung

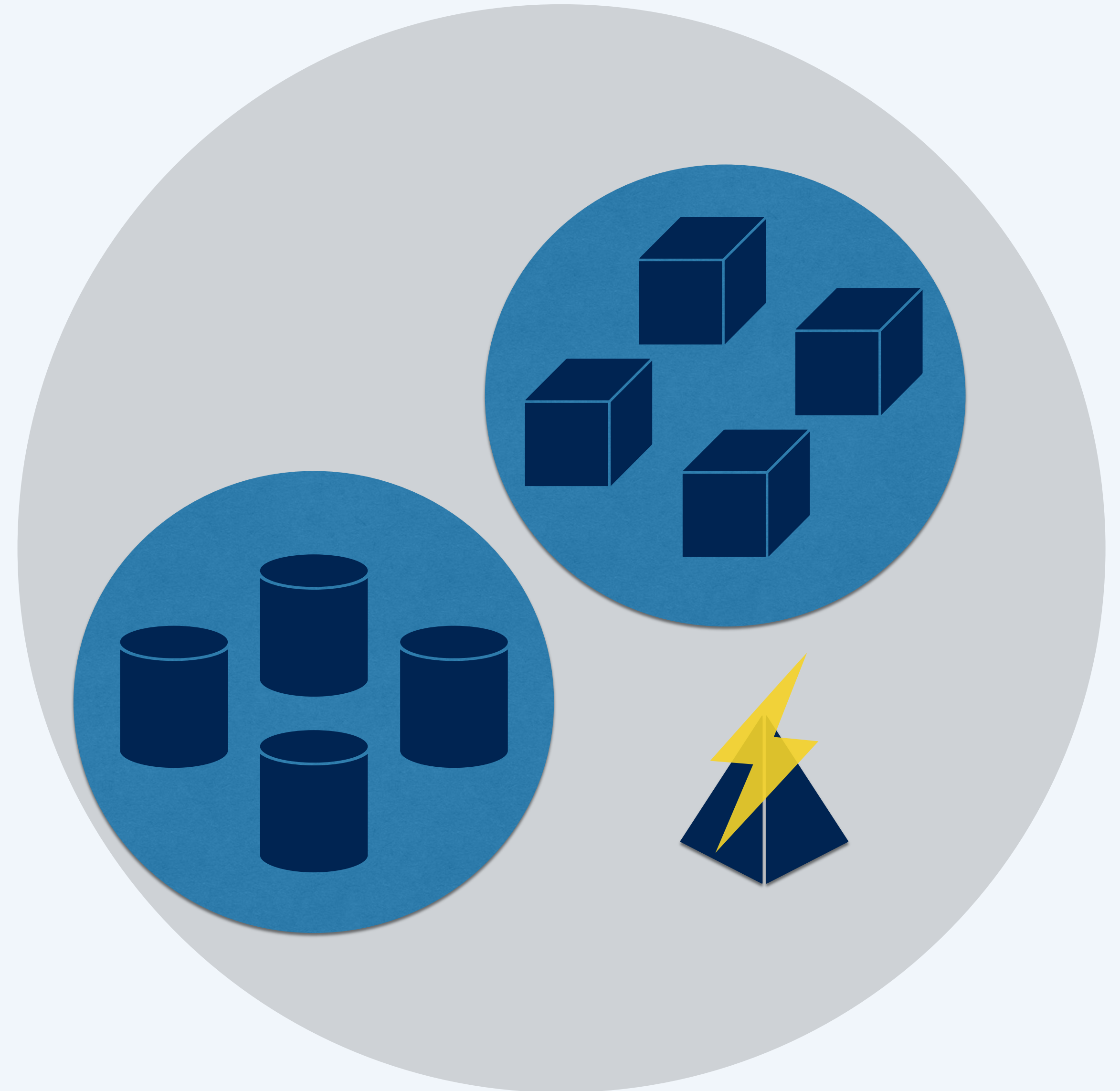
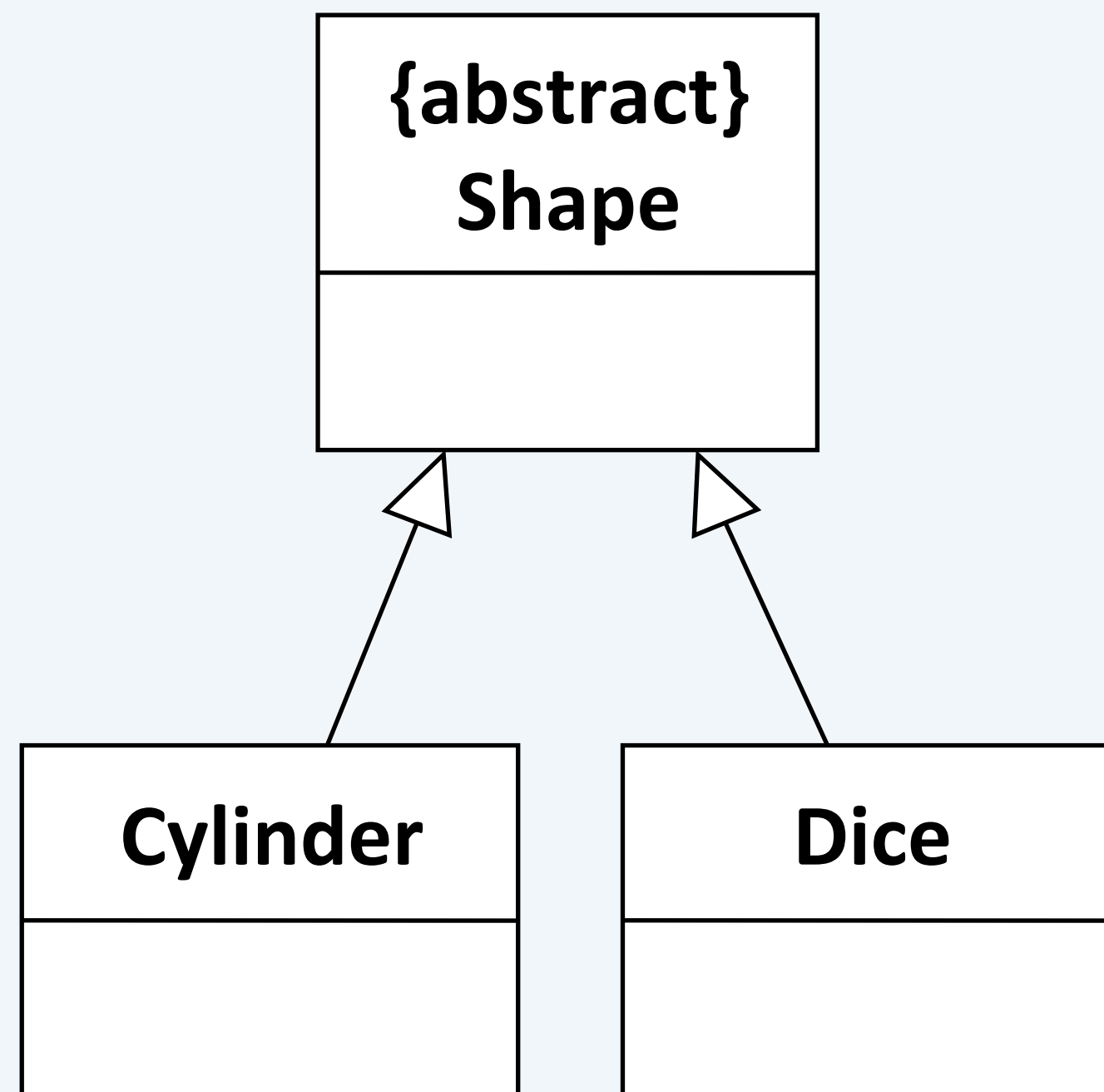
- Klassen können auch von mehreren Klassen erben
- Beispiel:



Tutor ist Employee und Student

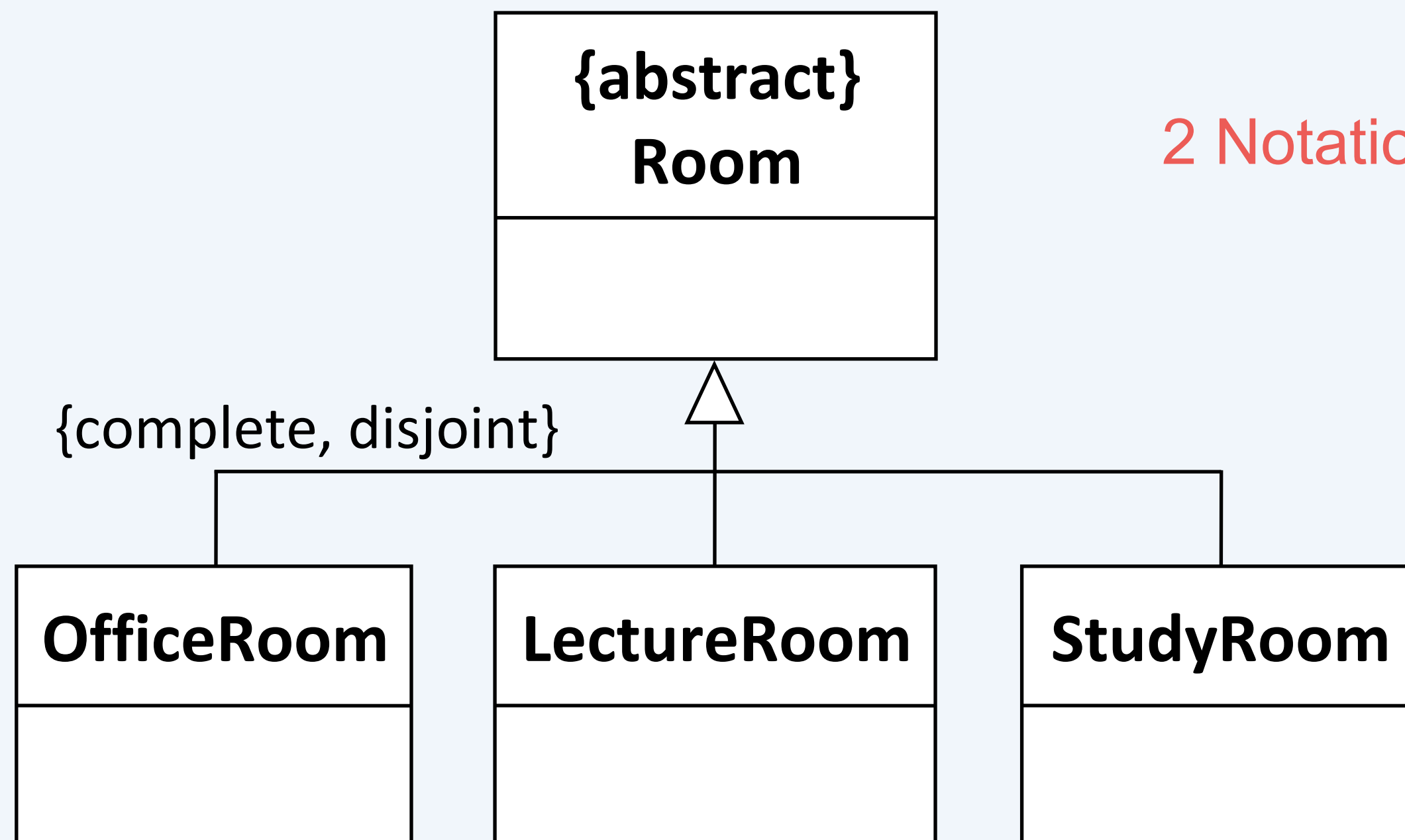
Generalisierung: Eigenschaften (1/2)

- Vollständige / unvollständige Aufteilung

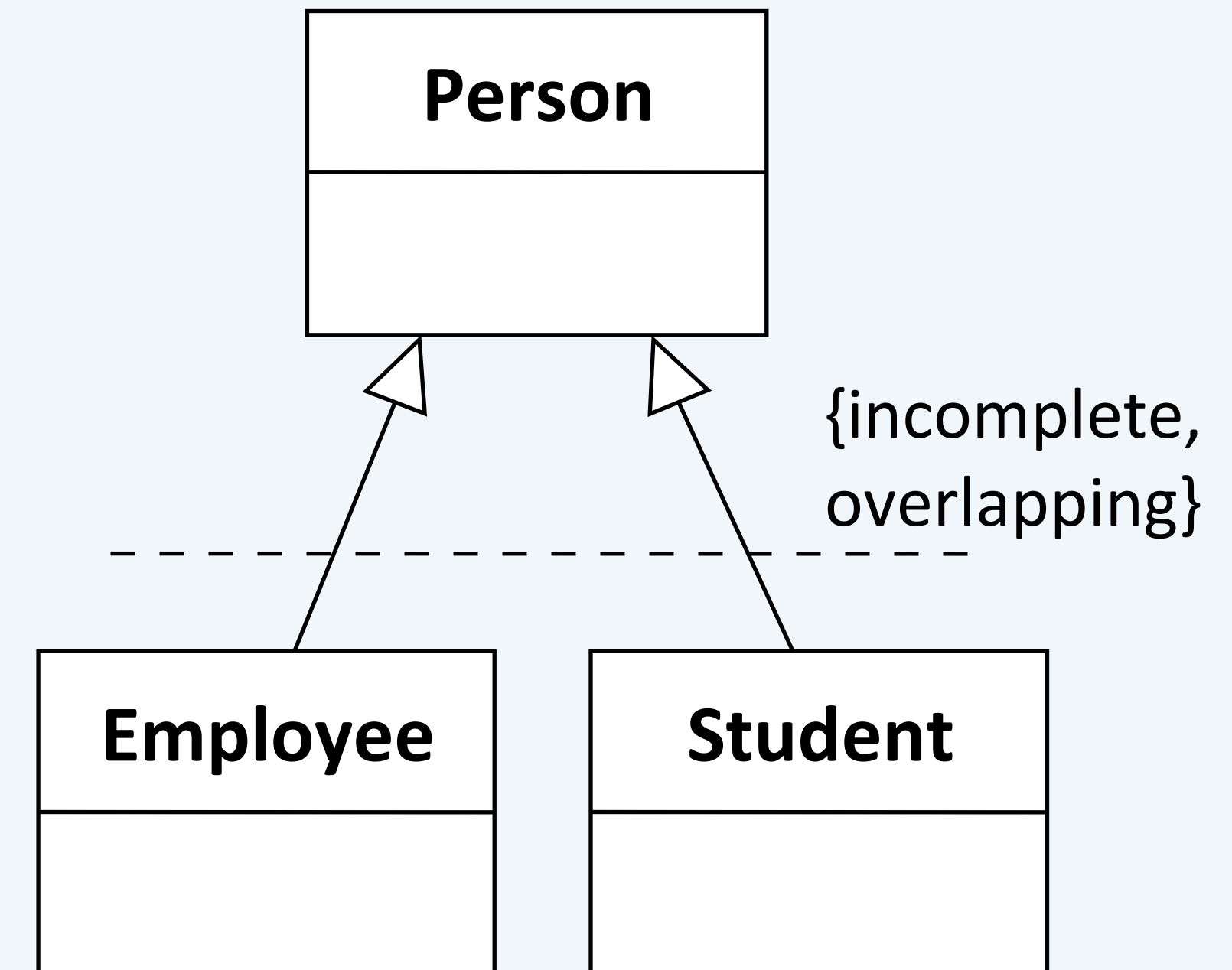


Generalisierung: Eigenschaften (2/2)

- Unterscheidung kann vorgenommen werden in
 - Unvollständig / vollständig
 - Überlappend / disjunkt
- Default: unvollständig, disjunkt



2 Notationen bei >1 Pfeil



Strukturmodellierung

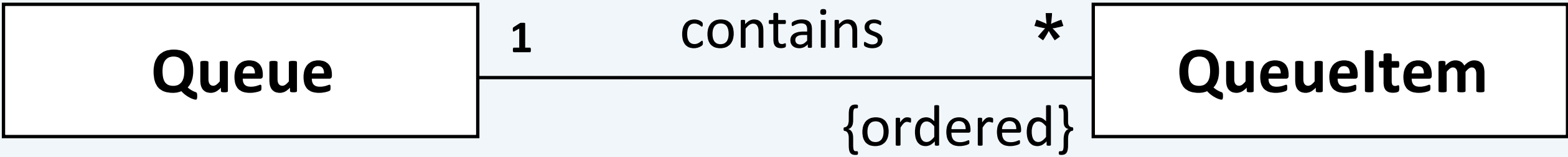
Die Ordnung und Eindeutigkeit von Assoziationen



Christian Huemer und Marion Scholz

Ordnung und Eindeutigkeit von Assoziationen

- Ordnung `{ordered}` ist unabhängig von Attributen

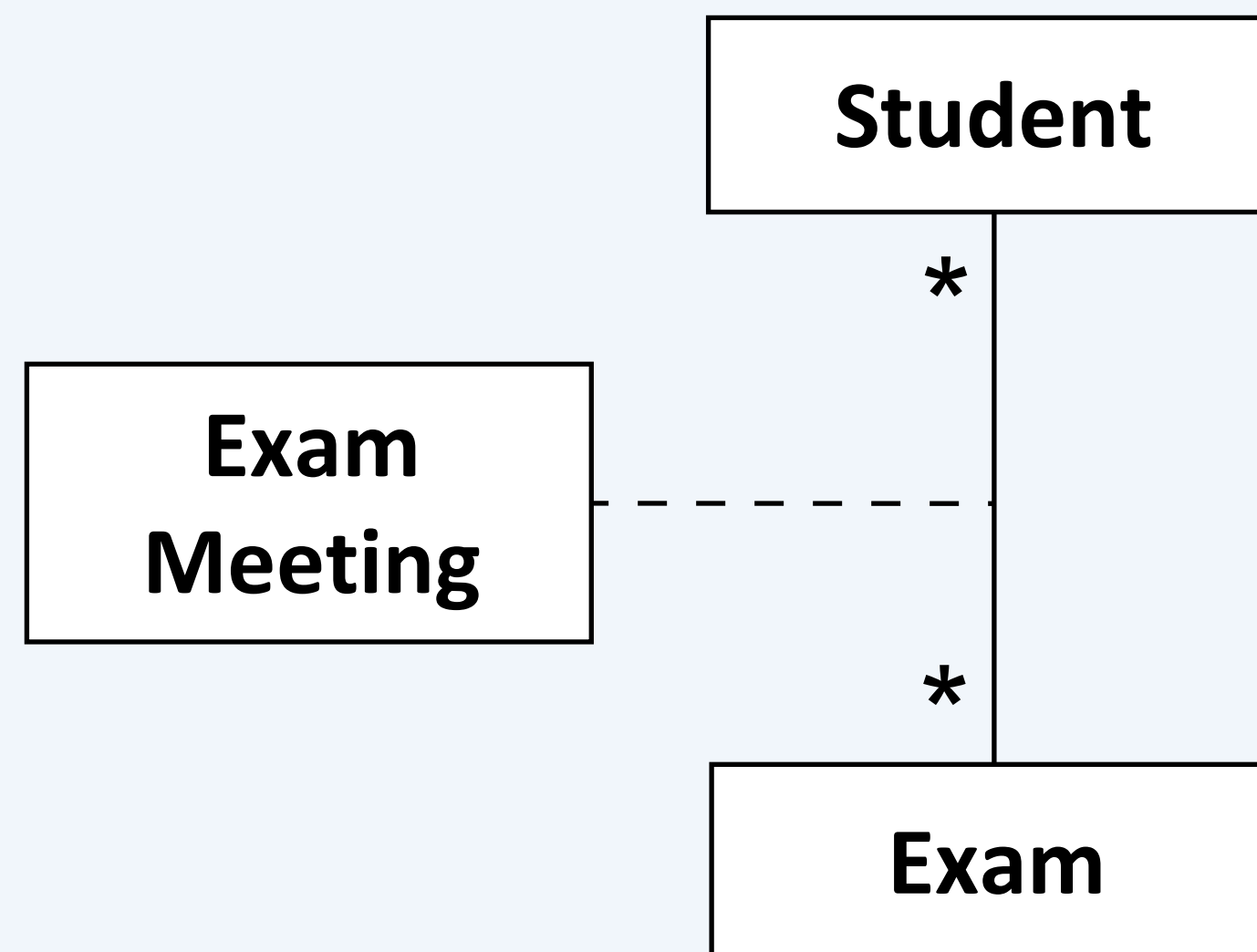


- Eindeutigkeit
 - Wie bei Attributen durch `{unique}` und `{nonunique}`
 - Kombination mit Ordnung `{set}`, `{bag}` und `{sequence}` bzw. `{seq}`

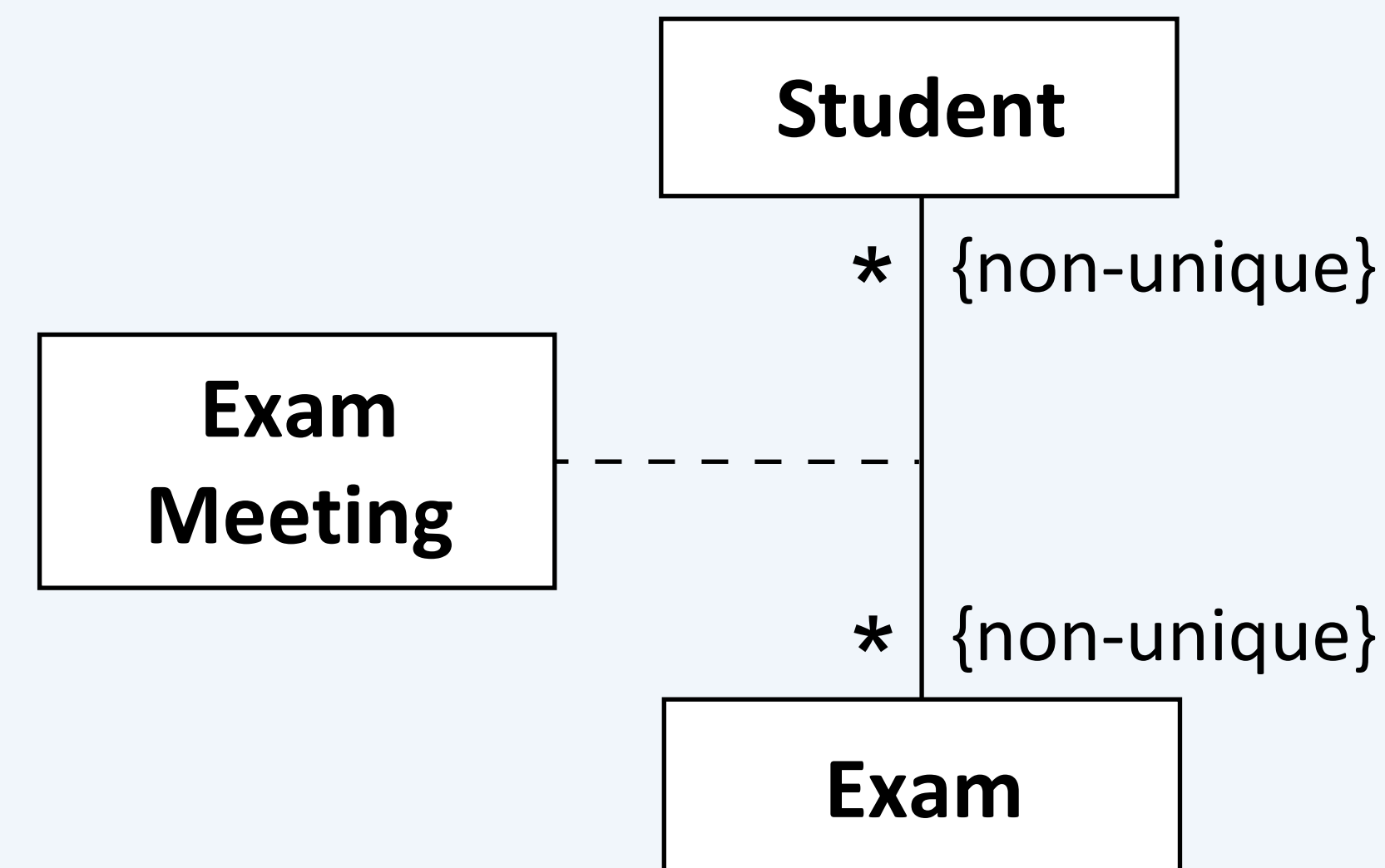
<u>Eindeutigkeit</u>	<u>Ordnung</u>	<u>Kombination</u>	<u>Beschreibung</u>
unique	unordered	set	Menge (Standardwert)
unique	ordered	orderedSet	Geordnete Menge
nonunique	unordered	bag	Multimenge (= Menge mit Duplikaten)
nonunique	ordered	sequence	Geordnete Menge mit Duplikaten (Liste)

Unique / Non-Unique (1/3)

- Default: keine Duplikate
- `{non-unique}`: Duplikate erlaubt



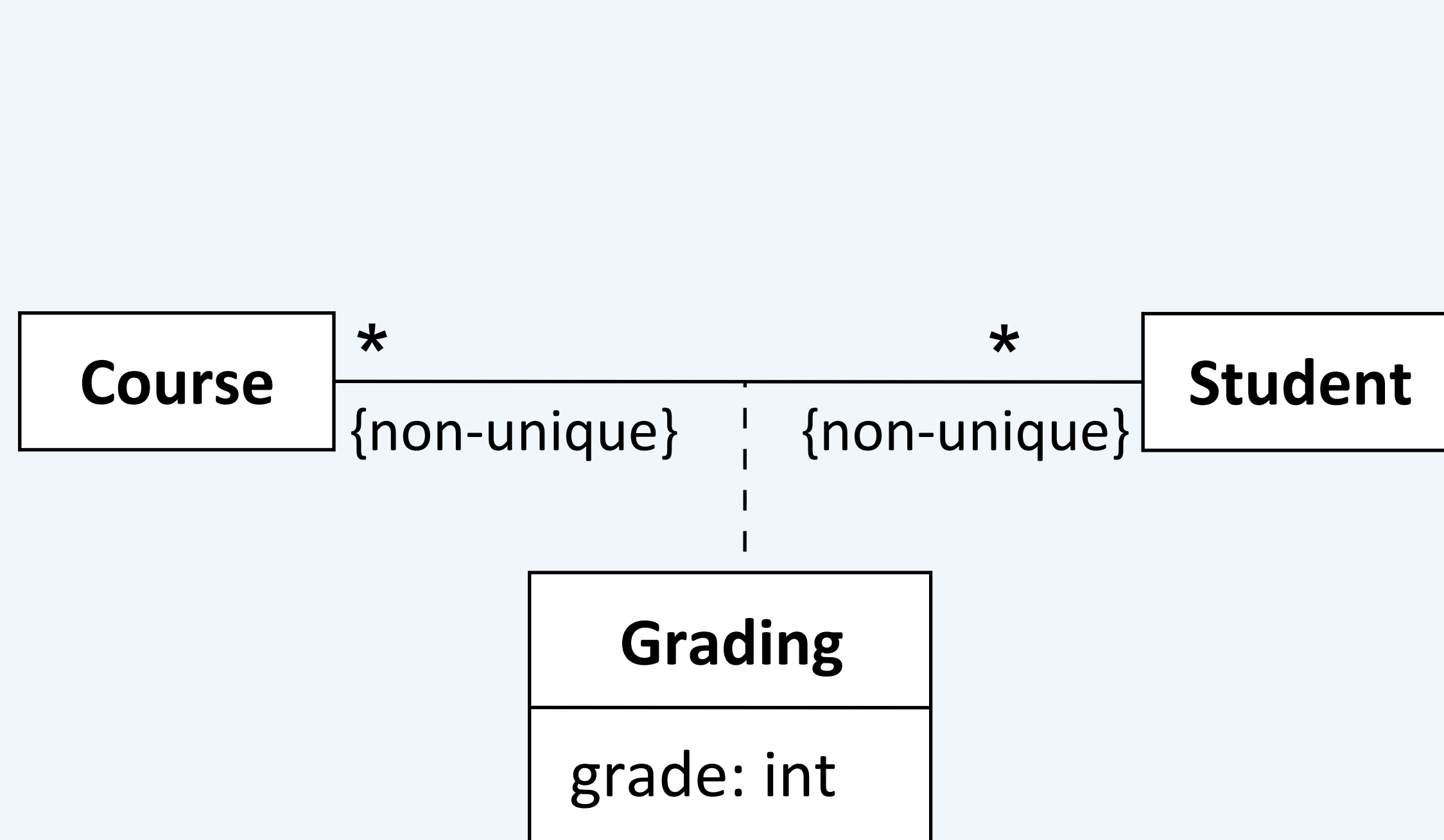
Ein **Student** kann für ein **Exam** nur **genau ein ExamMeeting** haben.



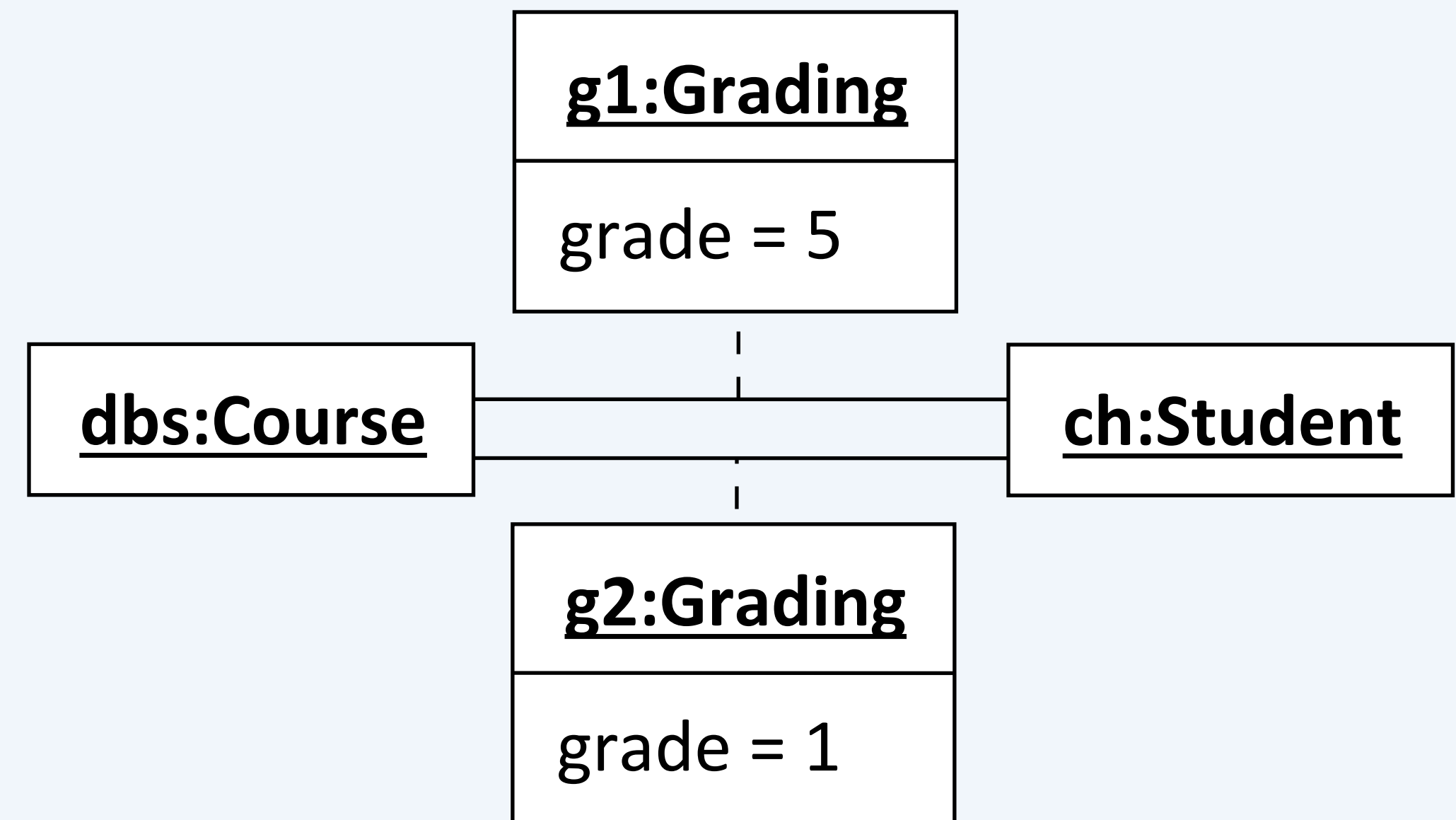
Ein **Student** kann für ein **Exam** **mehr als ein ExamMeeting** haben.

Unique / Non-Unique (2/3)

Klassendiagramm

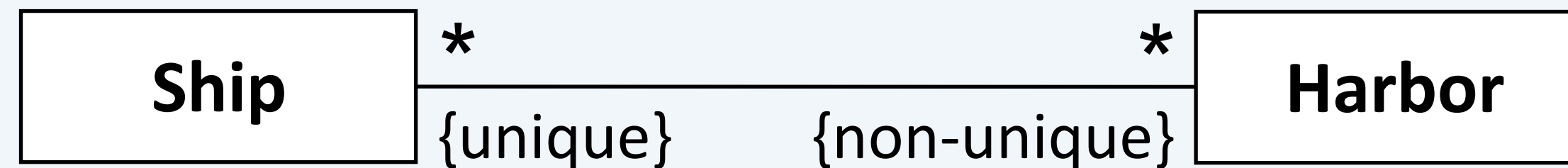


Objektdiagramm



Unique / Non-Unique (3/3)

- Es soll gespeichert werden, wie oft ein bestimmtes **Ship** in welchem **Harbor** war
- Es ist nicht relevant, dass der **Harbor** weiß wie oft ein bestimmtes **Ship** da war



- Abbildung mittels Links im Objektdiagramm NICHT möglich



Strukturmodellierung

Gesamtbeispiel Klassendiagramm



Christian Huemer und Marion Scholz

Beispiel – Universitätsinformationssystem

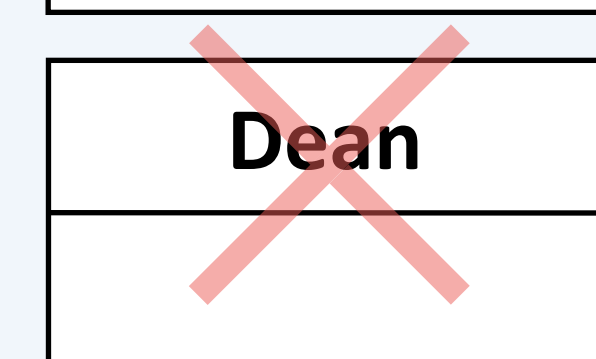
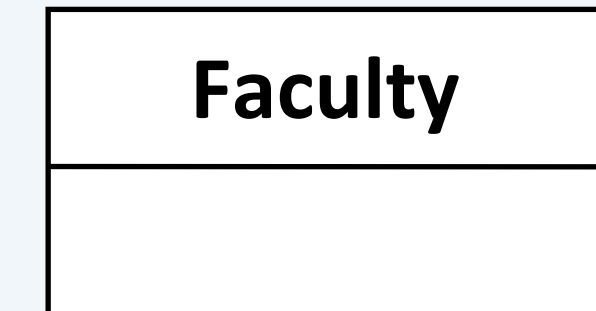
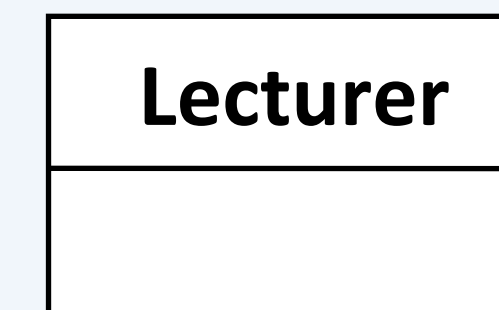
- Eine Universität besteht aus mehreren Fakultäten, die sich aus verschiedenen Instituten zusammensetzen. Jede Fakultät und jedes Institut hat einen Namen. Für jedes Institut ist eine Adresse bekannt.
- Jede Fakultät wird von einem Dekan bzw. einer Dekanin geleitet, der bzw. die MitarbeiterIn der Universität ist.
- Die Gesamtzahl der MitarbeiterInnen ist bekannt. MitarbeiterInnen haben eine Sozialversicherungsnummer, einen Namen und eine E-Mail-Adresse. Es wird zwischen Forschungs- und Verwaltungspersonal unterschieden.
- Wissenschaftliche MitarbeiterInnen (WM) sind mindestens einem Institut zugeordnet. Das Forschungsfeld jedes/r WM ist bekannt. Darüber hinaus können WM für eine bestimmte Anzahl von Stunden in Projekte eingebunden werden, wobei Name, Start- und Enddatum der Projekte bekannt sind. Manche WM halten Kurse. Dann werden sie Lehrende genannt.
- Kurse haben eine eindeutige Nummer (ID), einen Namen und eine wöchentliche Dauer in Stunden.

Schritt 1: Klassen identifizieren

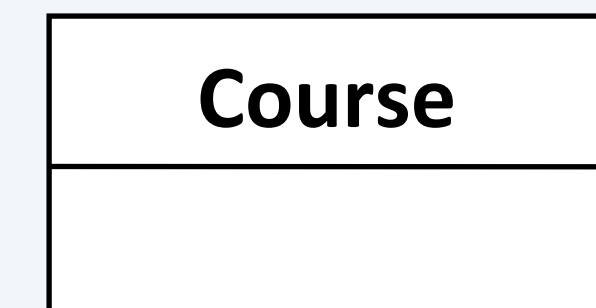
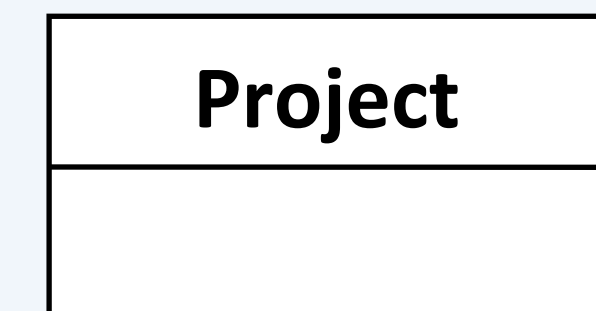
- Eine Universität besteht aus mehreren Fakultäten, die sich aus verschiedenen Instituten zusammensetzen. Jede Fakultät und jedes Institut hat einen Namen. Für jedes Institut ist eine Adresse bekannt.
- Jede Fakultät wird von einem Dekan bzw. einer Dekanin geleitet, der bzw. die MitarbeiterIn der Universität ist.
- Die Gesamtzahl der MitarbeiterInnen ist bekannt. MitarbeiterInnen haben eine Sozialversicherungsnummer, einen Namen und eine E-Mail-Adresse. Es wird zwischen Forschungs- und Verwaltungspersonal unterschieden.
- Wissenschaftliche MitarbeiterInnen (WM) sind mindestens einem Institut zugeordnet. Das Forschungsfeld jedes/r WM ist bekannt. Darüber hinaus können WM für eine bestimmte Anzahl von Stunden in Projekte eingebunden werden, wobei Name, Start- und Enddatum der Projekte bekannt sind. Manche WM halten Kurse. Dann werden sie Lehrende genannt.
- Kurse haben eine eindeutige Nummer (ID), einen Namen und eine wöchentliche Dauer in Stunden.



Wir modellieren
das System „University“



Dean hat nicht mehr
Attribute als jeder
andere **Employee**



Schritt 2: Attribute identifizieren

- Eine Universität besteht aus mehreren Fakultäten, die sich aus verschiedenen Instituten zusammensetzen. Jede Fakultät und jedes Institut hat einen Namen. Für jedes Institut ist eine Adresse bekannt.
- Jede Fakultät wird von einem Dekan bzw. einer Dekanin geleitet, der bzw. die MitarbeiterIn der Universität ist.
- Die Gesamtzahl der MitarbeiterInnen ist bekannt. MitarbeiterInnen haben eine Sozialversicherungsnummer, einen Namen und eine E-Mail-Adresse. Es wird zwischen Forschungs- und Verwaltungspersonal unterschieden.
- Wissenschaftliche MitarbeiterInnen (WM) sind mindestens einem Institut zugeordnet. Das Forschungsfeld jedes/r WM ist bekannt. Darüber hinaus können WM für eine bestimmte Anzahl von Stunden in Projekte eingebunden werden, wobei Name, Start- und Enddatum der Projekte bekannt sind. Manche WM halten Kurse. Dann werden sie Lehrende genannt.
- Kurse haben eine eindeutige Nummer (ID), einen Namen und eine wöchentliche Dauer in Stunden.

Institute
+ name: String
+ address: String

Faculty
+ name: String

Employee
+ ssNo: int
+ name: String
+ email: String
+ <u>counter</u> : int

Research Associate
+ fieldOfStudy: String

Administrative Employee

Project
+ name: String
+ start: Date
+ end: Date

Lecturer

Course
+ name: String
+ id: int
+ hours: float

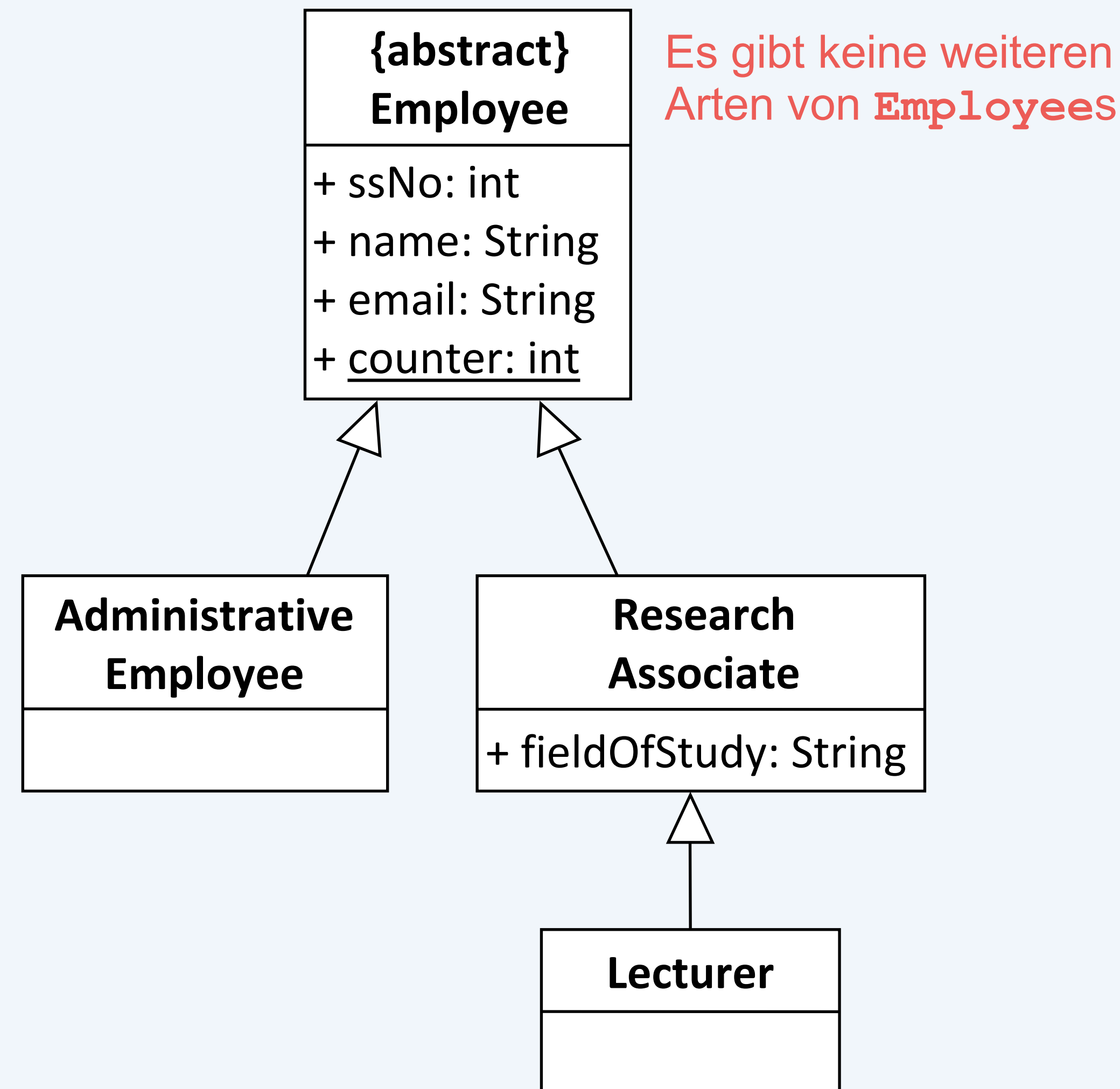
Schritt 3: Beziehungen identifizieren (1/6)

■ Drei Arten von Beziehungen:

- Generalisierung
- Assoziation
- Aggregation

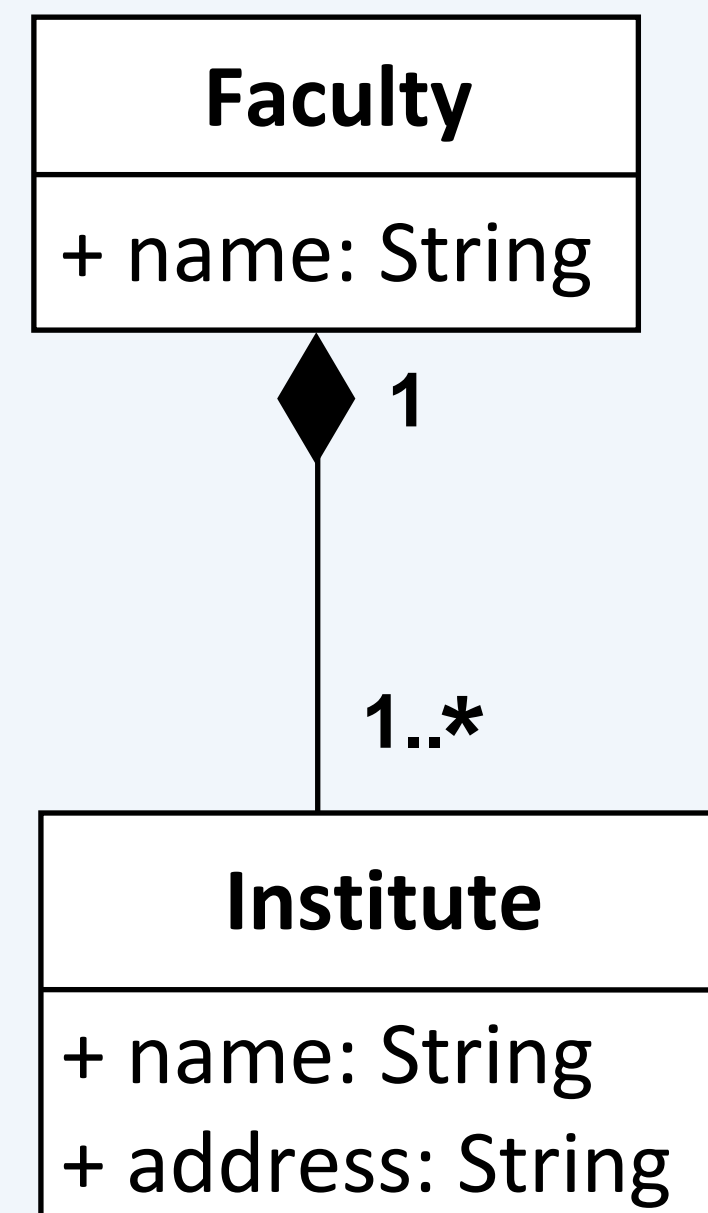
■ Anzeichen für eine Generalisierung

- "Es wird zwischen Forschungs- und Verwaltungspersonal unterschieden."
- "Manche Wissenschaftliche MitarbeiterInnen halten Kurse. Dann werden sie Lehrende genannt."



Schritt 3: Beziehungen identifizieren (2/6)

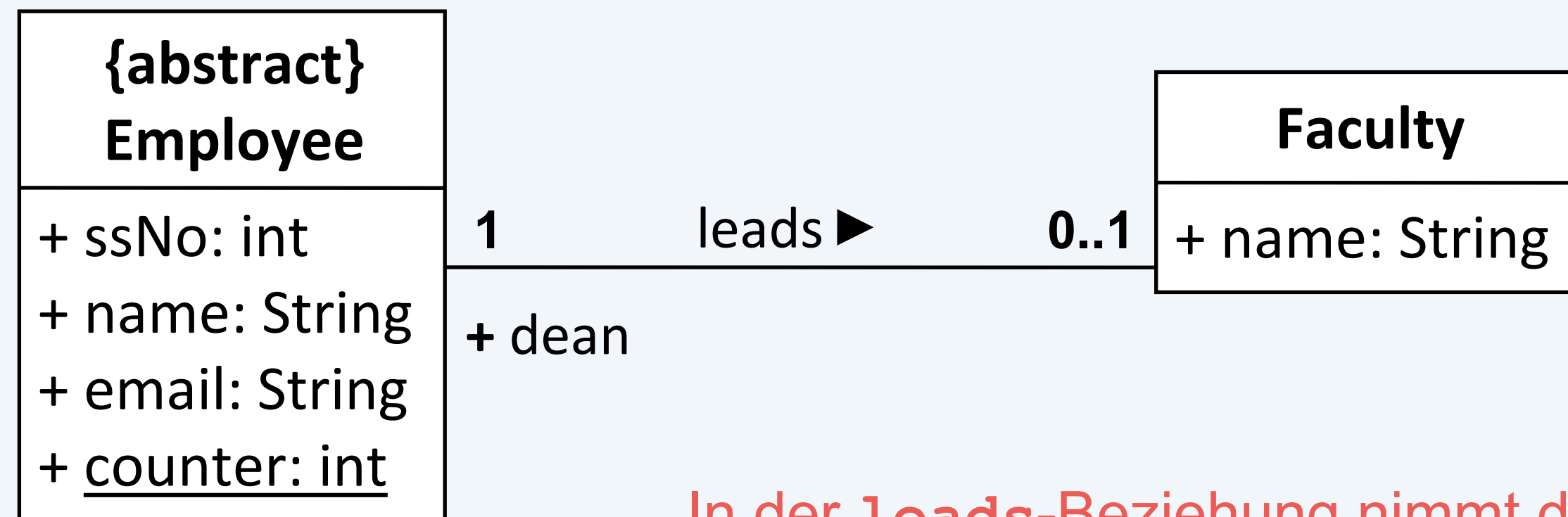
- "Eine Universität besteht aus mehreren Fakultäten, die sich aus verschiedenen Instituten zusammensetzen."



Komposition, um die
Existenzabhängigkeit abzubilden

Schritt 3: Beziehungen identifizieren (3/6)

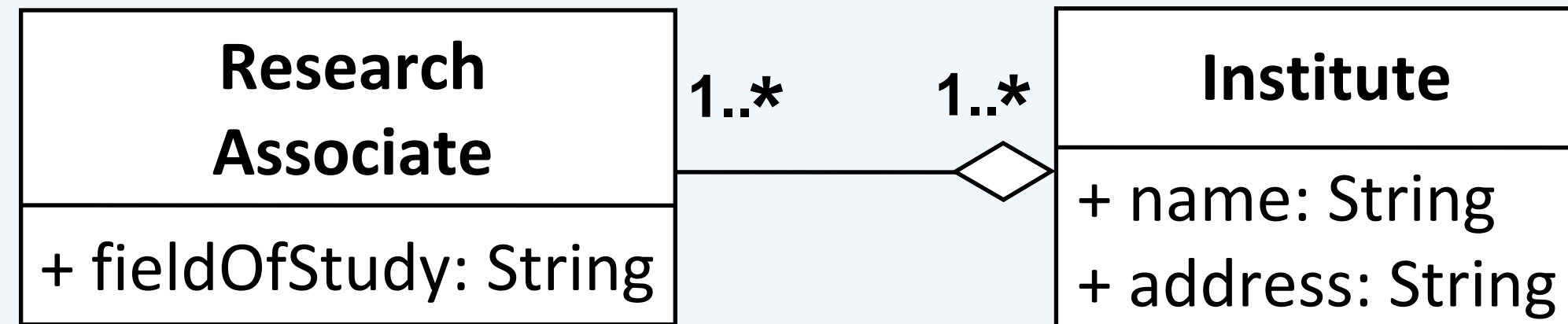
- "Jede Fakultät wird von einem Dekan bzw. einer Dekanin geleitet, der bzw. die MitarbeiterIn der Universität ist."



In der **leads**-Beziehung nimmt der **Employee** die Rolle des **dean** ein.

Schritt 3: Beziehungen identifizieren (4/6)

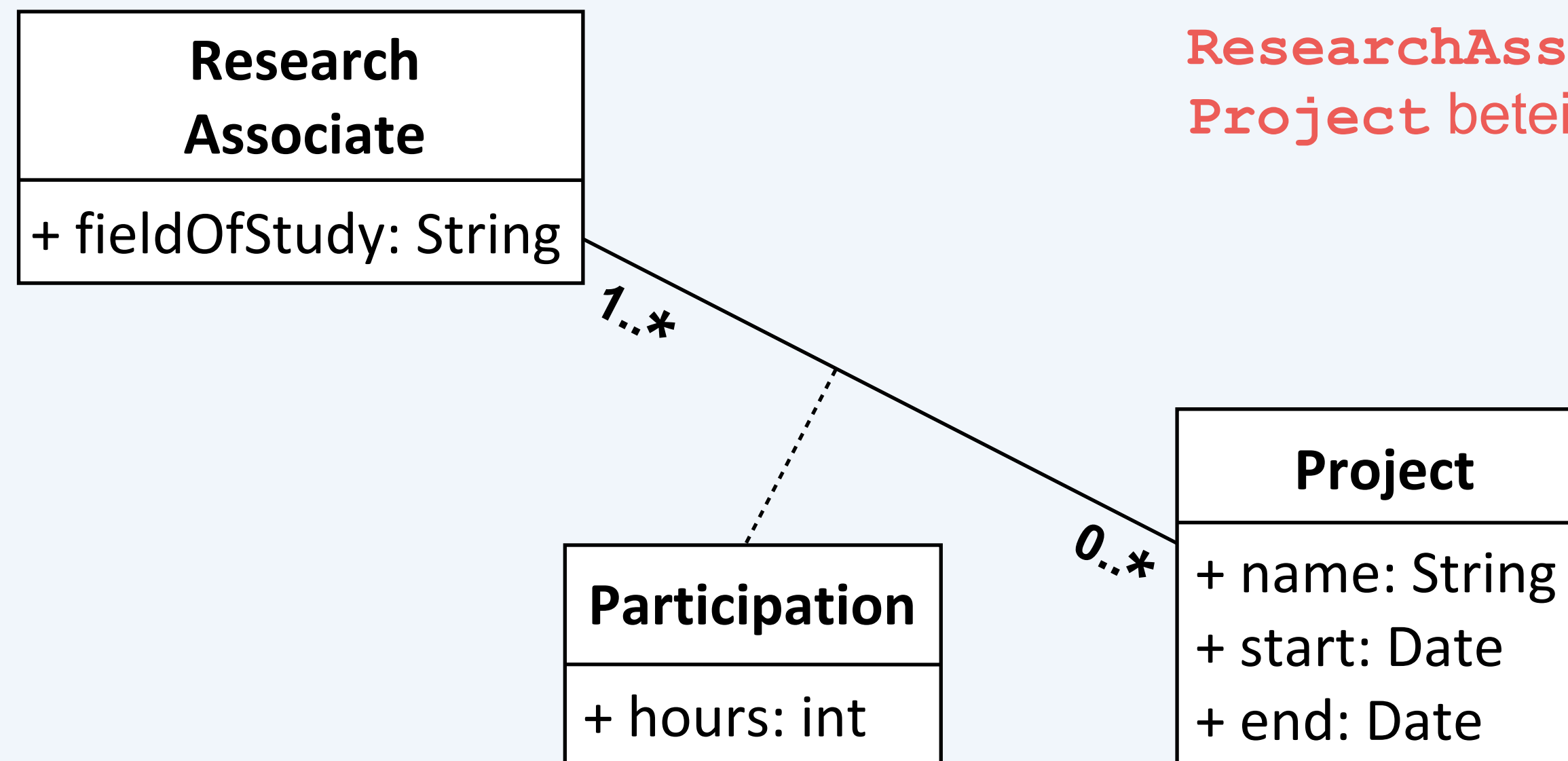
- "Wissenschaftliche MitarbeiterInnen sind mindestens einem Institut zugeordnet."



Schwache Aggregation um zu zeigen, dass **ResearchAssociates** Teil eines **Institute** sind, aber keine Existenzabhängigkeit besteht.

Schritt 3: Beziehungen identifizieren (5/6)

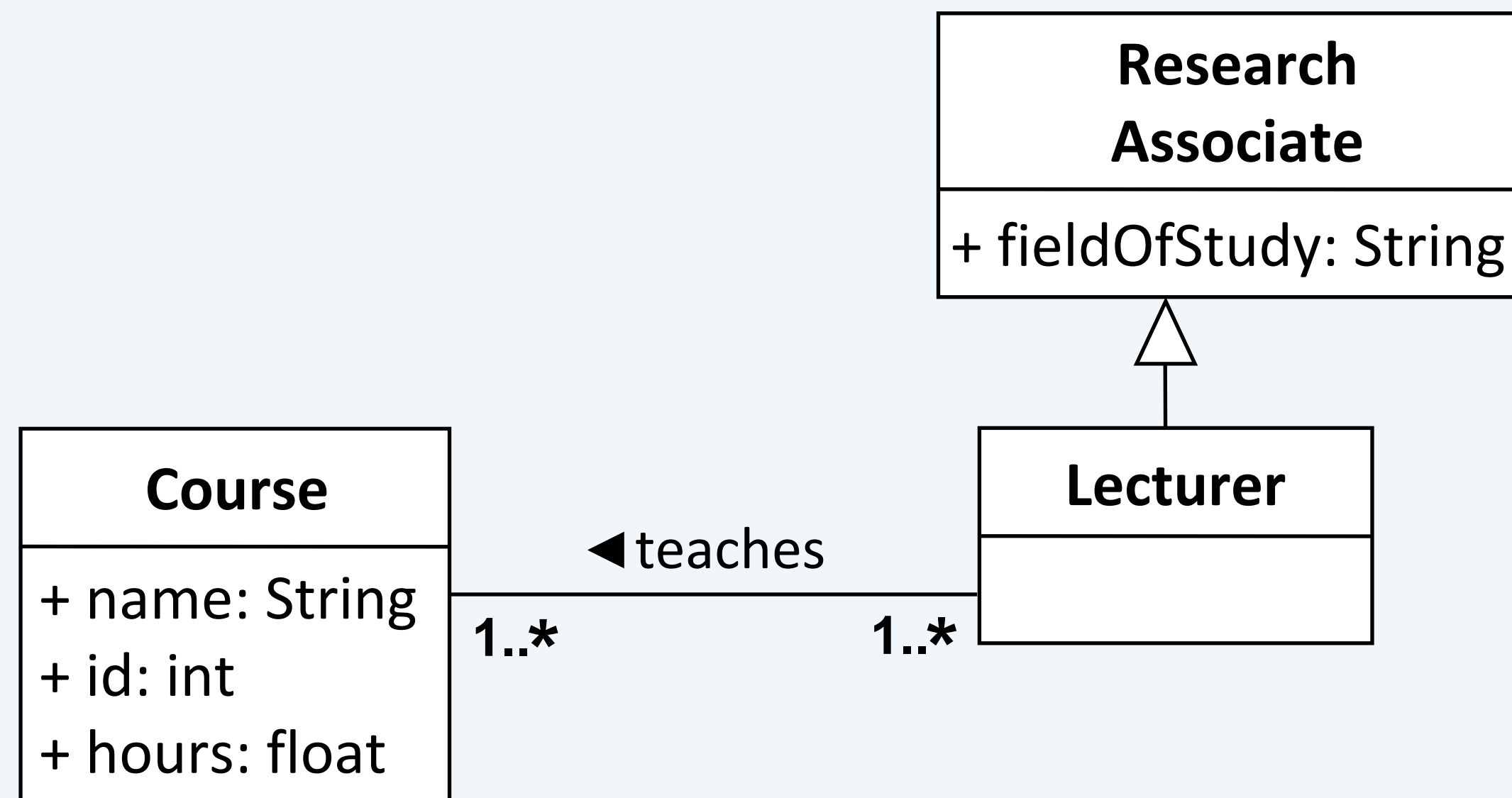
- "Darüber hinaus können WM für eine bestimmte Anzahl von Stunden in Projekte eingebunden werden."



Die Assoziationsklasse ermöglicht es, die Stundenzahl zu speichern, die jeder einzelne **ResearchAssociate** an einem einzelnen **Project** beteiligt ist.

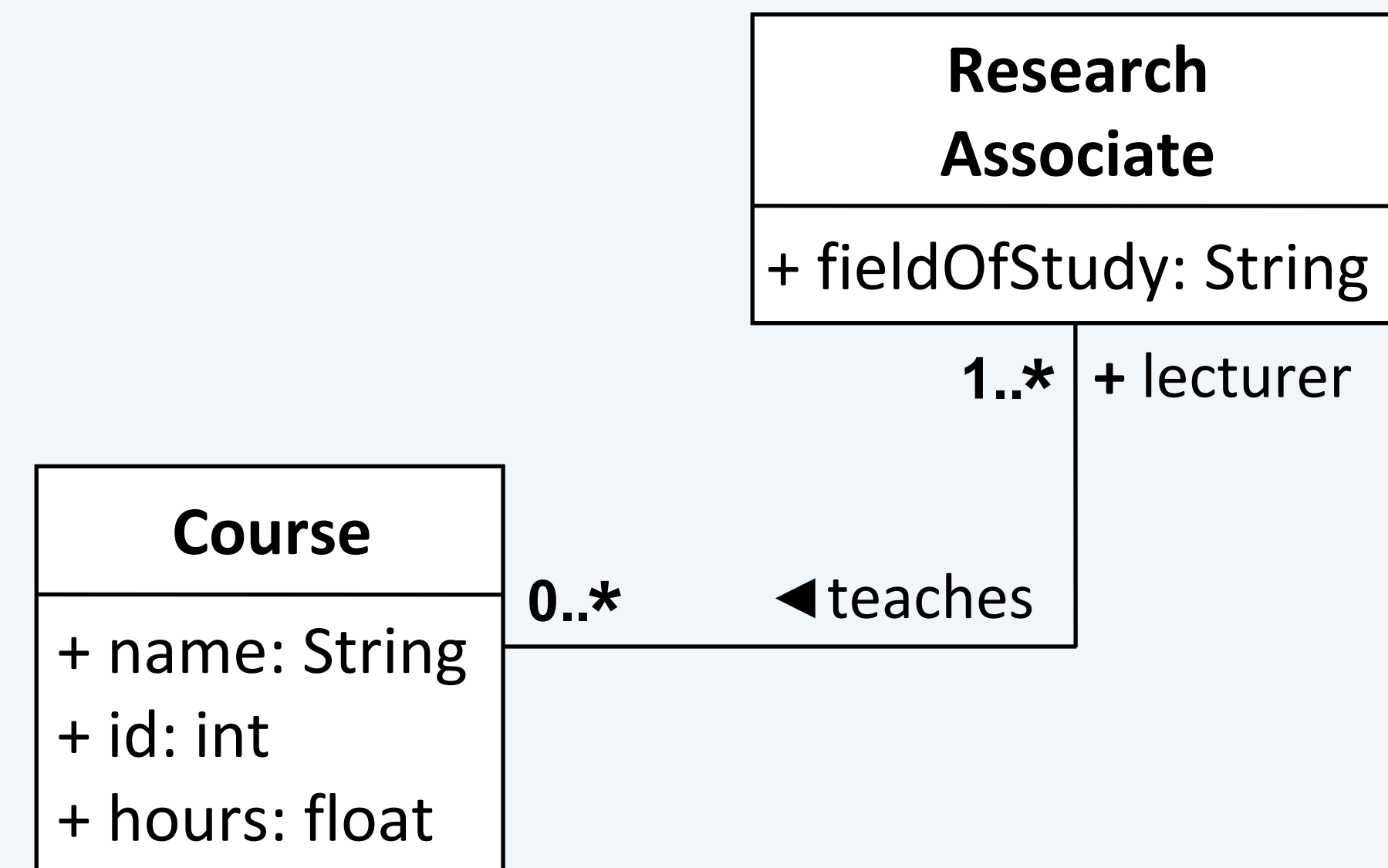
Schritt 3: Beziehungen identifizieren (6/6)

- "Manche Wissenschaftliche MitarbeiterInnen halten Kurse. Dann werden sie Lehrende genannt."



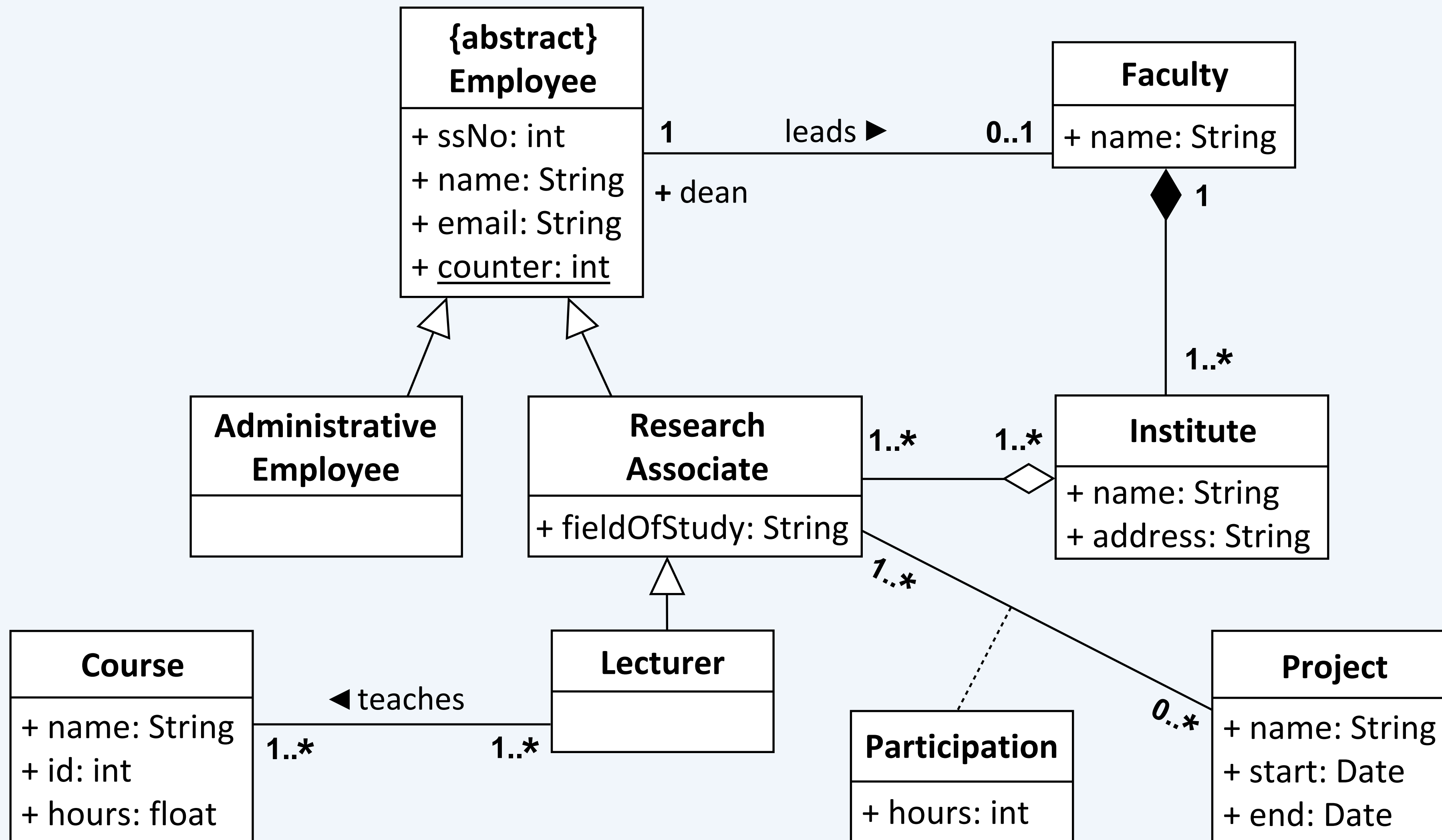
Lecturer erbt alle Eigenschaften und Beziehungen von **ResearchAssociate**.
Ein **Lecturer** hat zusätzlich eine Beziehung `teaches` zu **Course**.

oder



ResearchAssociate hat eine Beziehung `teaches` zu **Course**, in der Rolle `lecturer`.

...und jetzt alles zusammen



Strukturmodellierung

Die Datentypen

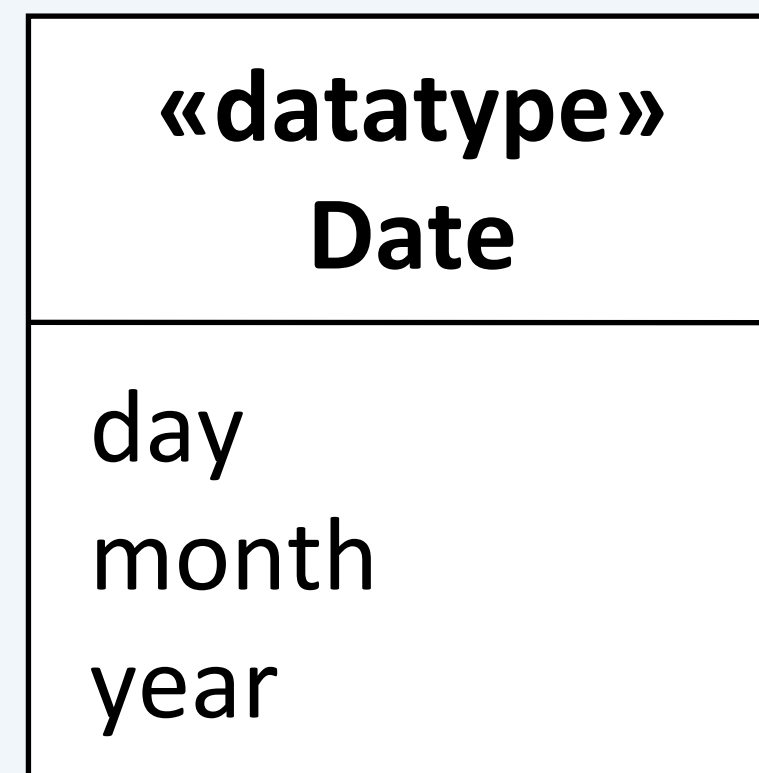


Christian Huemer und Marion Scholz

Datentypen in UML



- Instanzen eines Datentyps haben keine Identität
 - Objekte: Instanzen einer Klasse
 - Werte: Instanzen eines Datentyps (z.B. Zahl 2)
- Notation: Rechteck mit Schlüsselwort «**datatype**» im ersten Abschnitt
- Arten von Datentypen:
 - Primitive Datentypen
 - Datentype mit Attributen (und Operationen)
 - Aufzählungstypen

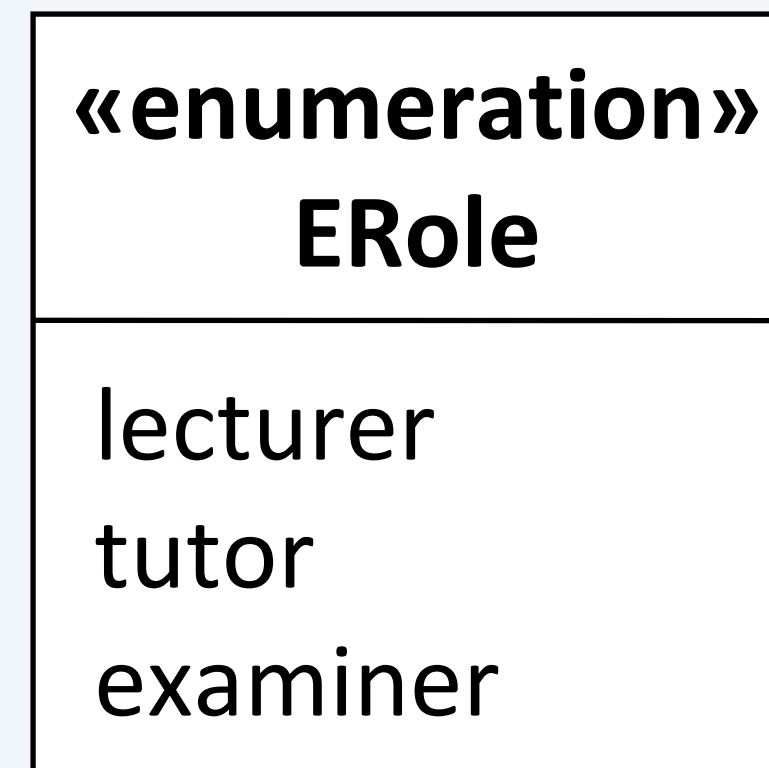
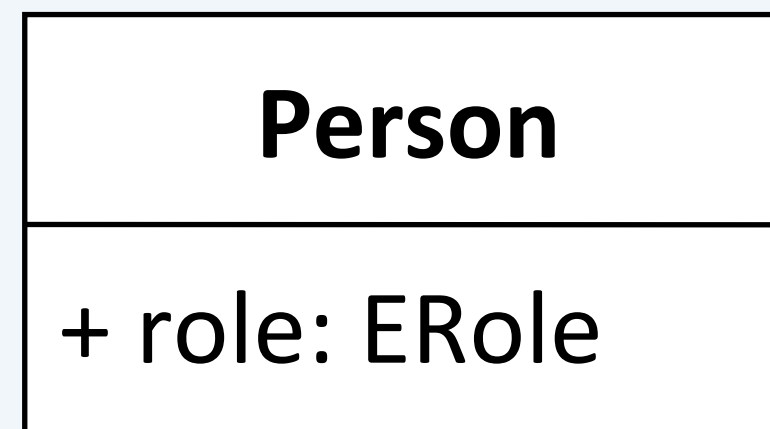


Arten von Datentypen : Primitive Datentypen

- Primitive Datentypen: Datentypen ohne innere Struktur
- Von UML vordefinierte, primitive Datentypen:
 - Boolean
 - Integer
 - UnlimitedNatural
 - String
- Primitive Datentypen können auch selbst definiert werden:
 - Schlüsselwort «**primitive**»

Arten von Datentypen : Aufzählungstypen

- Festlegung des Wertebereichs durch Aufzählung der möglichen Werte
- Notation: Klassensymbol mit Schlüsselwort «**enumeration**»
- Mögliche Ausprägungen werden durch benutzerdefinierte Bezeichner (Literele) angegeben



Strukturmodellierung

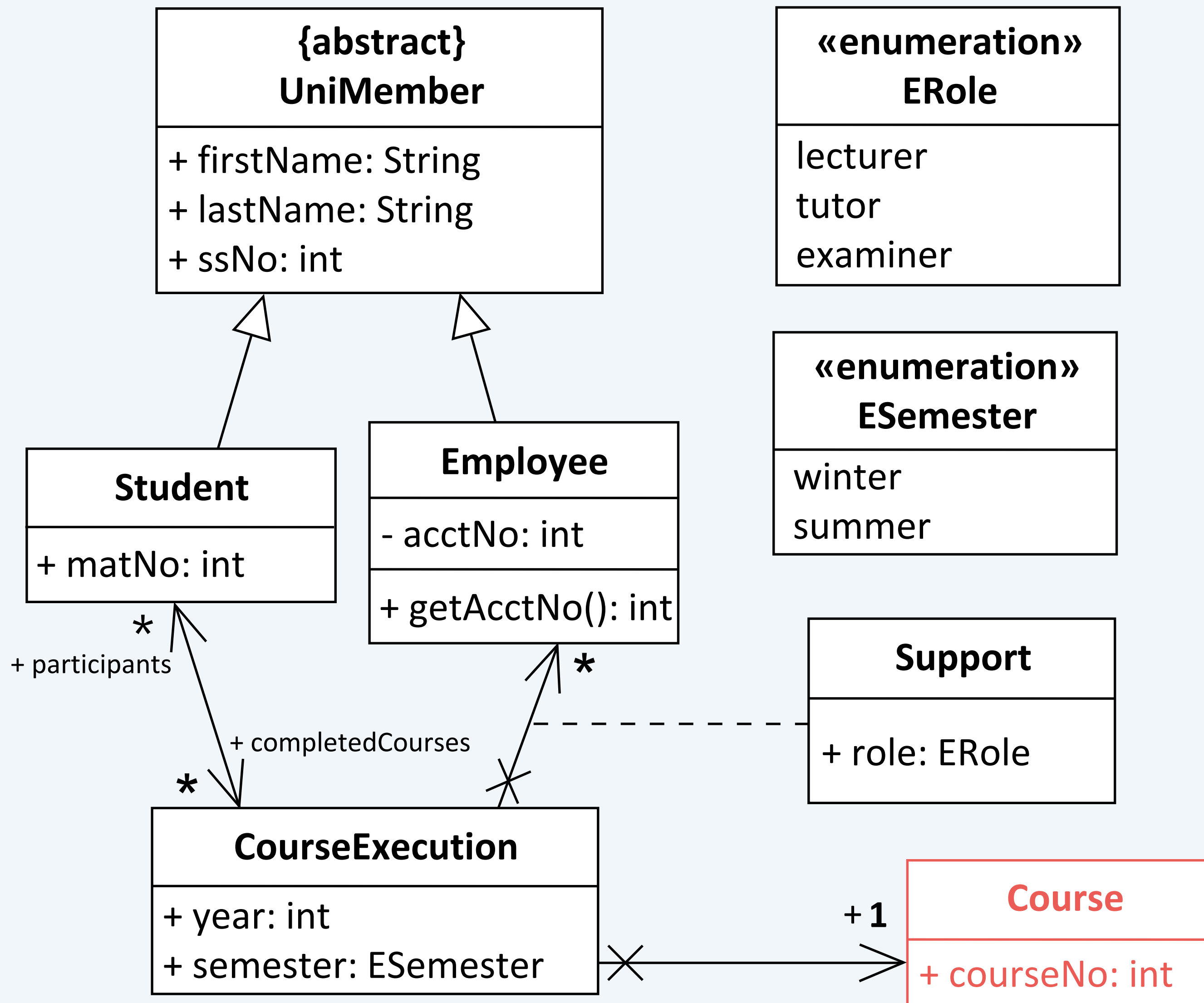
Die Übersetzung nach Java



Christian Huemer und Marion Scholz

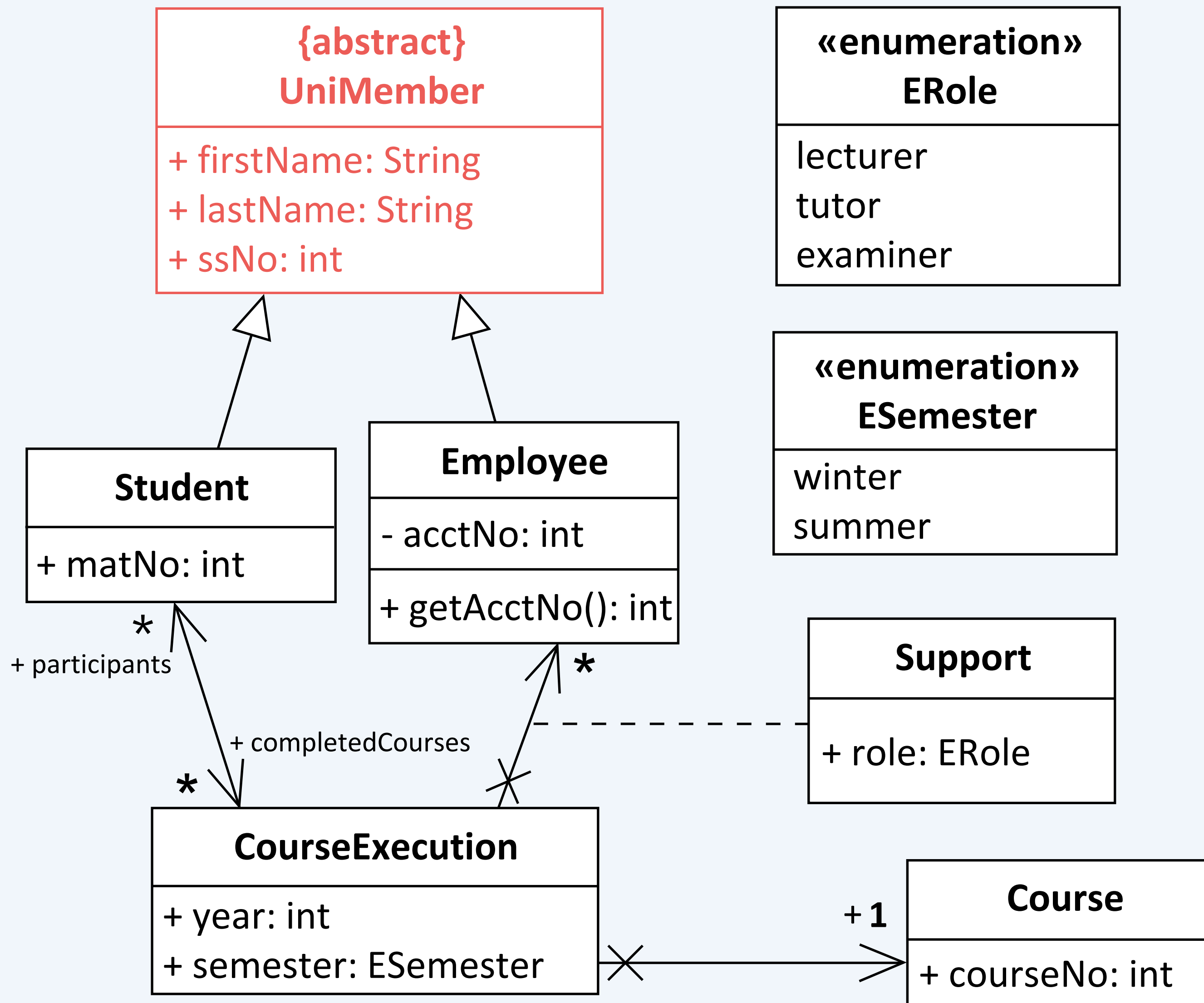
- Klassendiagramme werden oft mit der Intention erstellt, die modellierten Elemente in einer objektorientierten Programmiersprache umzusetzen
- Die Übersetzung kann in vielen Fällen automatisch erfolgen und bedarf nur geringer manueller Intervention

Übersetzung nach Java – Beispiel (1/6)



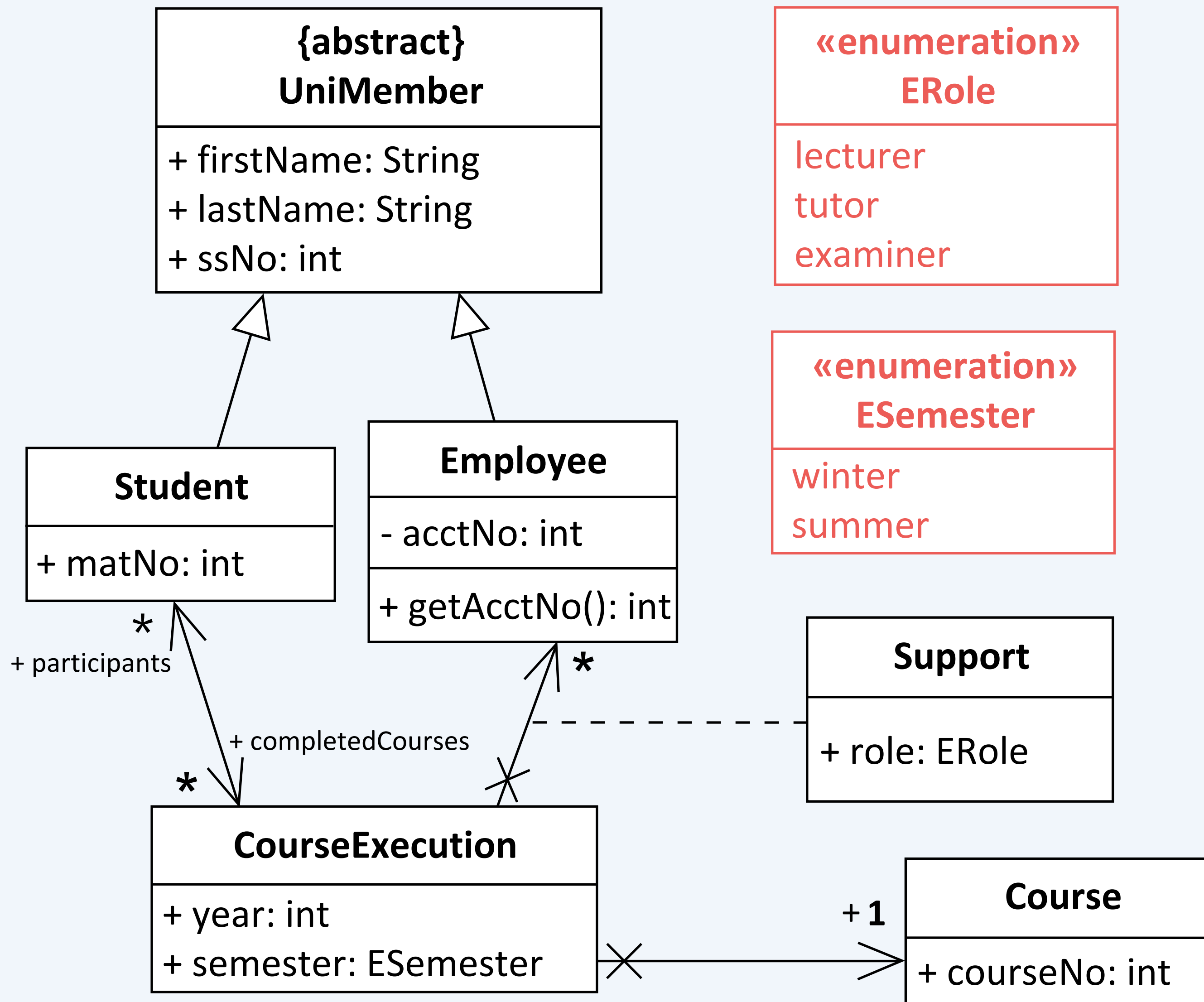
```
class Course {
    public int courseNo;
}
```

Übersetzung nach Java – Beispiel (2/6)



```
abstract class UniMember {
    public String firstName;
    public String lastName;
    public int ssNo;
}
```

Übersetzung nach Java – Beispiel (3/6)



«enumeration»
ERole

lecturer
tutor
examiner

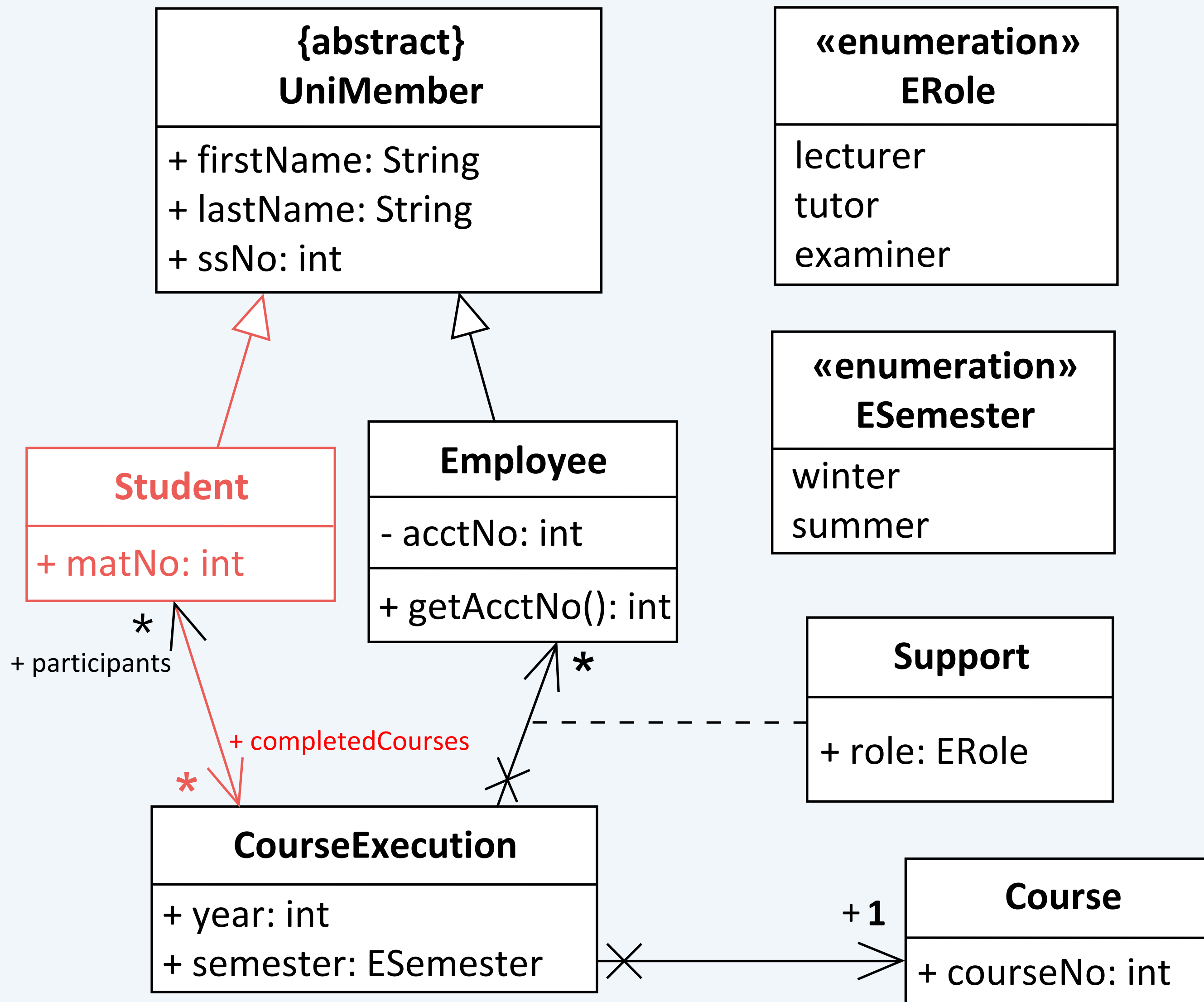
«enumeration»
ESemester

winter
summer

```
Enumeration ERole {  
    lecturer,  
    tutor,  
    examiner  
}
```

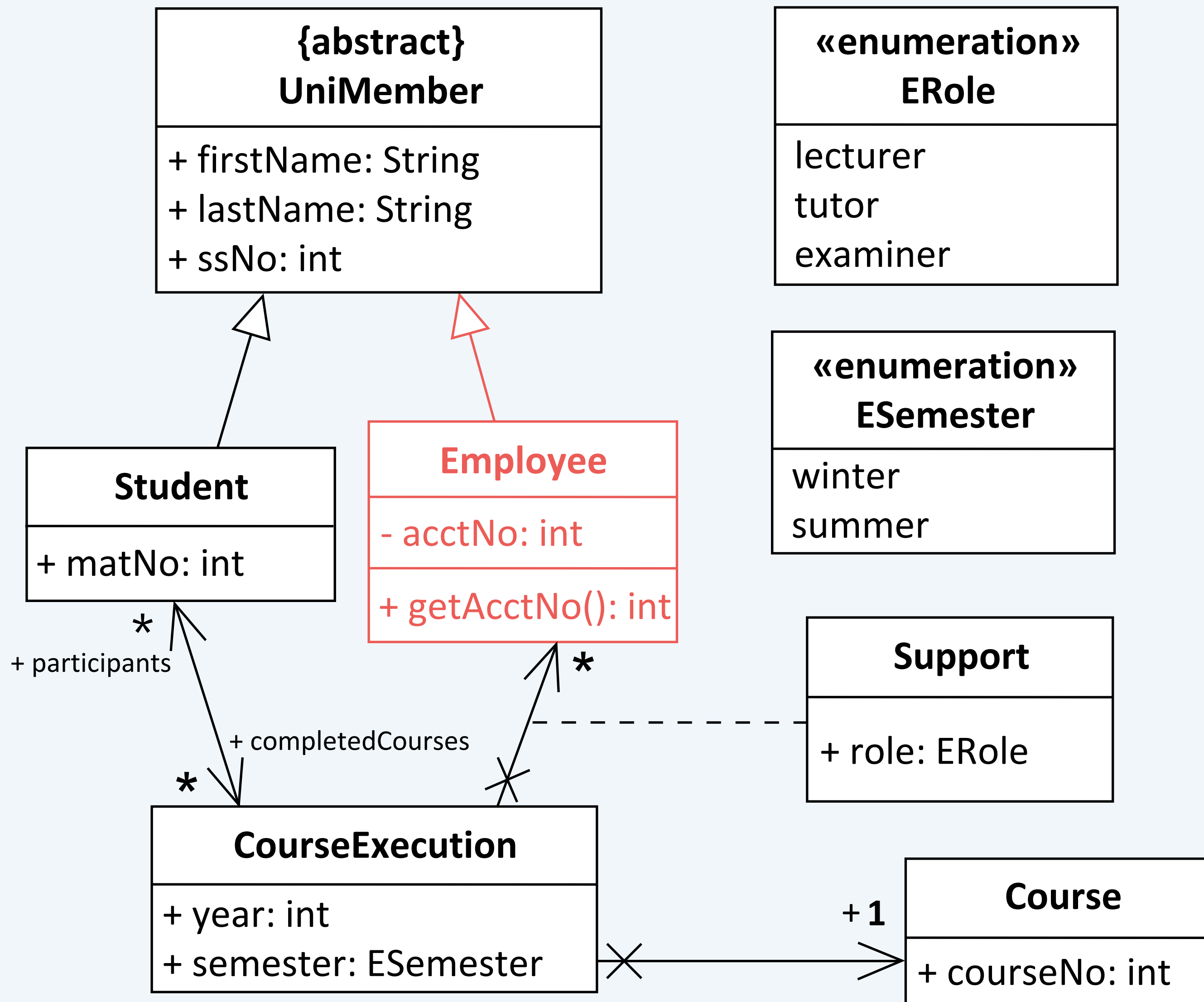
```
Enumeration ESemester {  
    winter,  
    summer  
}
```


Übersetzung nach Java – Beispiel (4/6)



```
class Student extends UniMember {
    public int matNo;
    public CourseExecution []
        completedCourses;
}
```

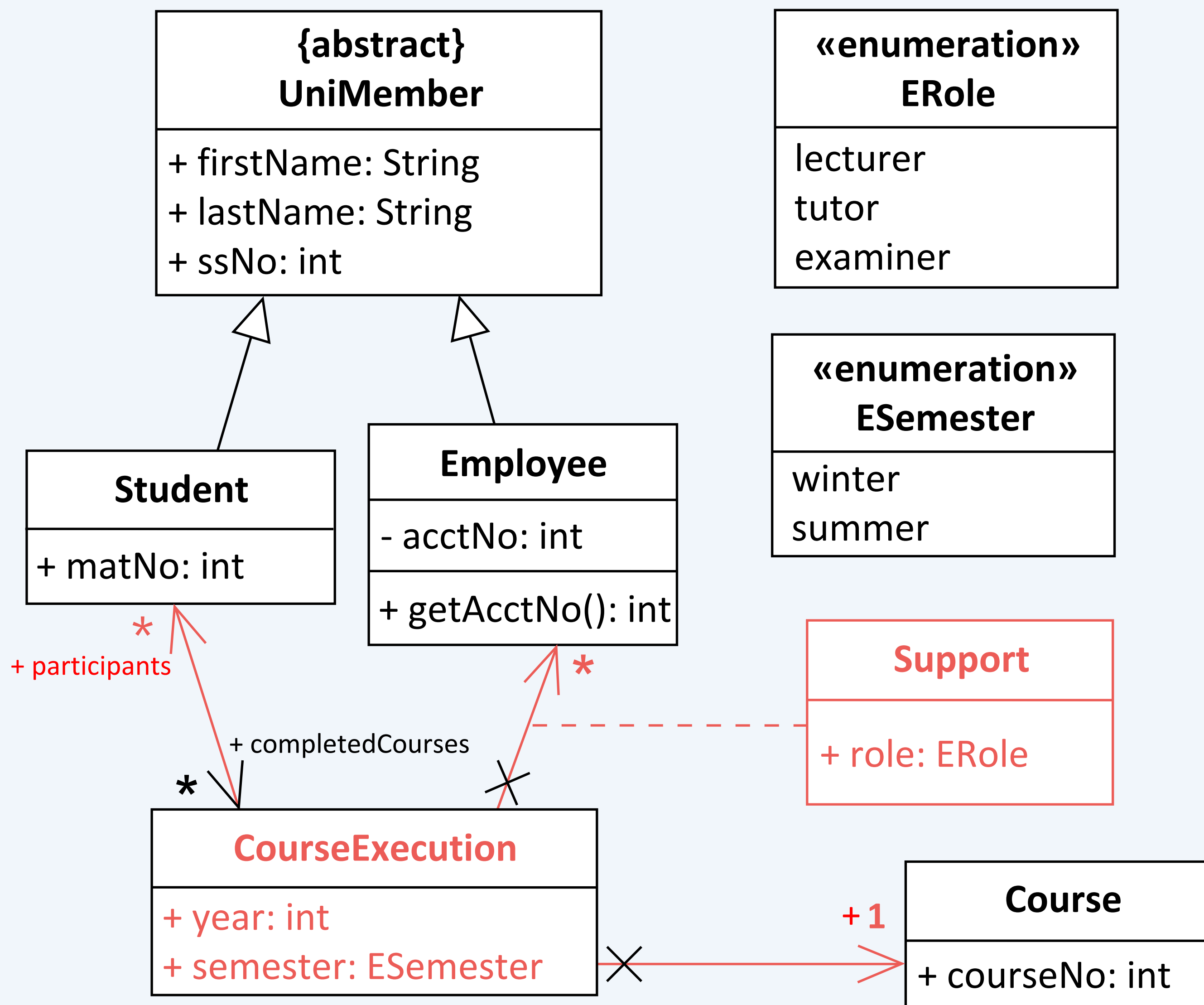
Übersetzung nach Java – Beispiel (5/6)



```
class Employee extends UniMember {
    private int acctNo;

    public int getAcctNo () {
        return acctNo;
    }
}
```

Übersetzung nach Java – Beispiel (6/6)



```
class CourseExecution {
    public int year;
    public ESemester semester;
    public Student [] participants;
    public Course the_course;
    public Hashtable support;
    // Key: Employee
    // Value: ERole
}
```

Übersetzung nach Java: Zusammenfassung



- Klassen → Java-Klassen
- Attribute → Instanzvariablen
- Operationen → Methoden
- Klassenvariablen und -operationen: **static**
- Assoziationen
 - Aufnahme von Attribut nur bei Navigierbarkeit
 - Multiplizität = 1: Variable
 - Multiplizität ≥ 2 : Arrays, ArrayLists,...
- Einfachvererbung: **extends**
- Assoziationsklassen: Hashtables

Strukturmodellierung Reverse Engineering



Christian Huemer und Marion Scholz

Reverse Engineering



```
class Person {  
    public String name;  
    public Dog bf;  
}  
  
class Dog {  
    public String name;  
    public Person o;  
}
```



Reverse Engineering



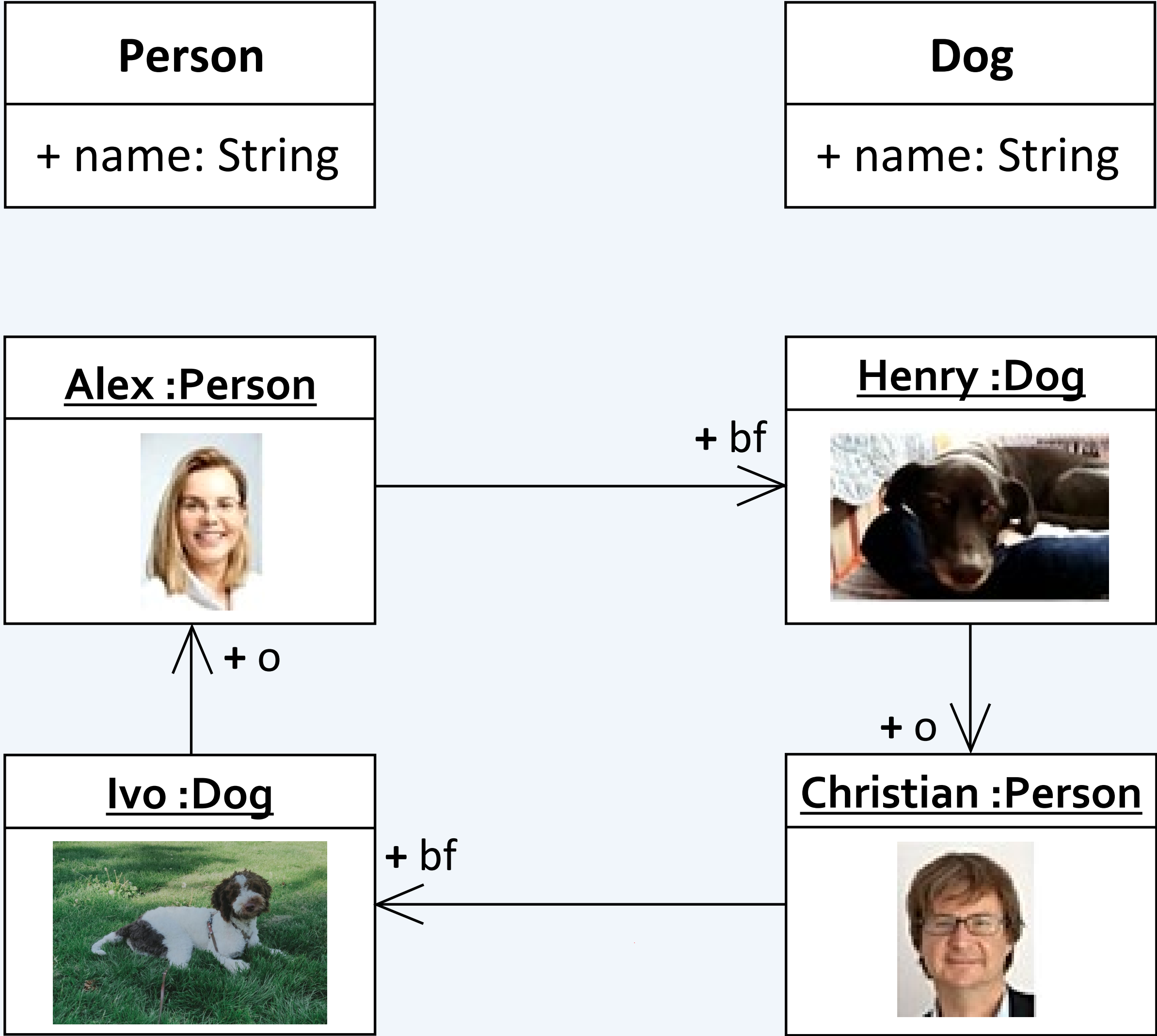
```
class Person {
  public String name;
  public Dog bf;
}

class Dog {
  public String name;
  public Person o;
}
```



Reverse Engineering

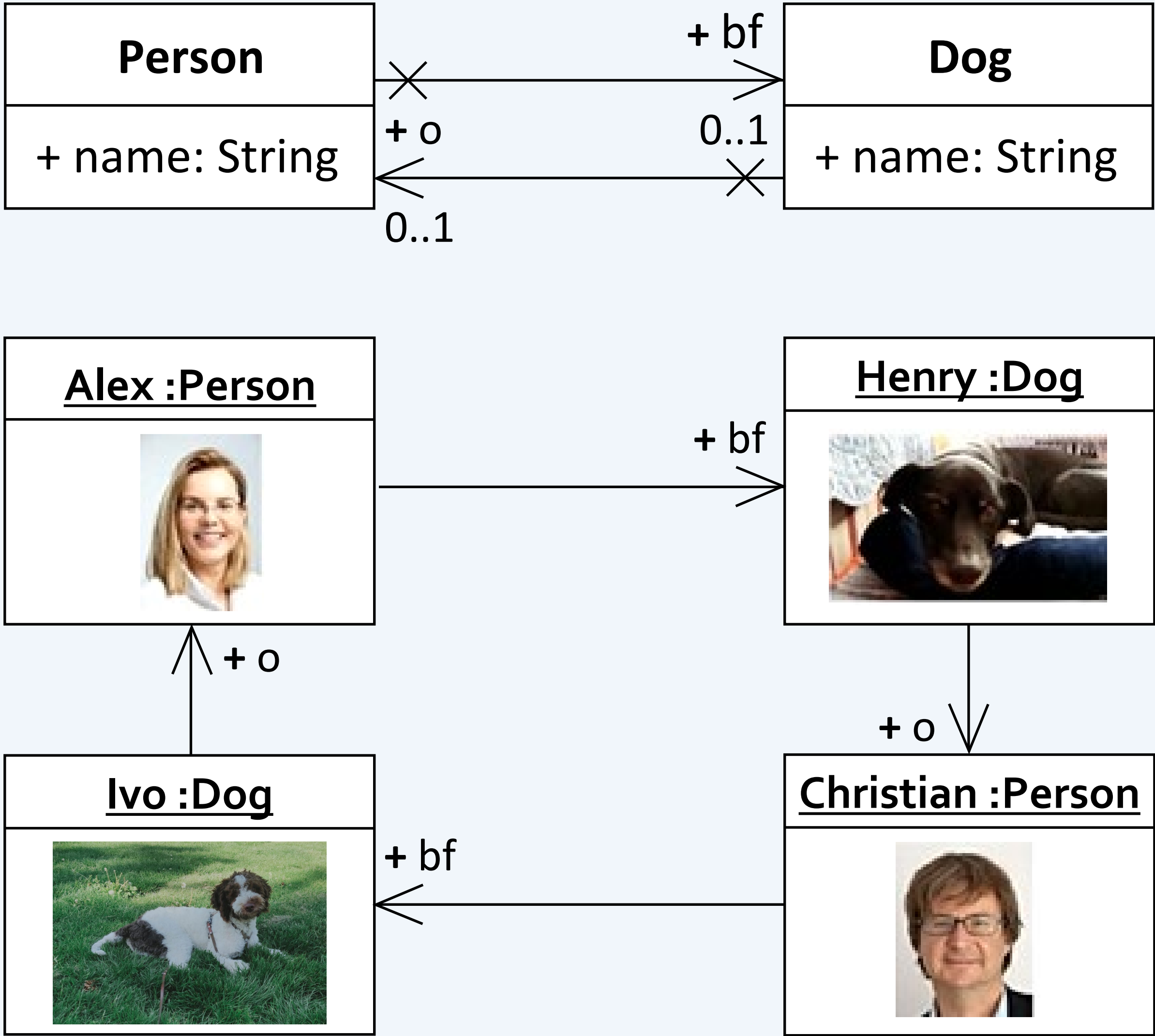
```
class Person {  
  public String name;  
  public Dog bf;  
}  
  
class Dog {  
  public String name;  
  public Person o;  
}
```



Reverse Engineering

```
class Person {
  public String name;
  public Dog bf;
}

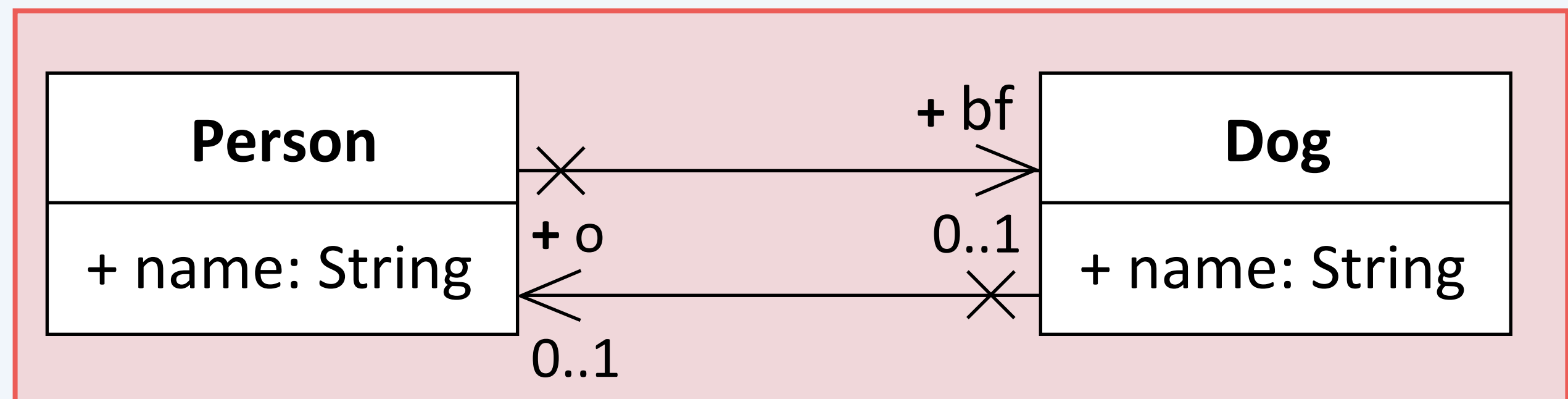
class Dog {
  public String name;
  public Person o;
}
```



Forward vs. Reverse Engineering

```
class Person {  
  public String name;  
  public Dog bf;  
}
```

```
class Dog {  
  public String name;  
  public Person ○;  
}
```



Strukturmodellierung Das Paketdiagramm



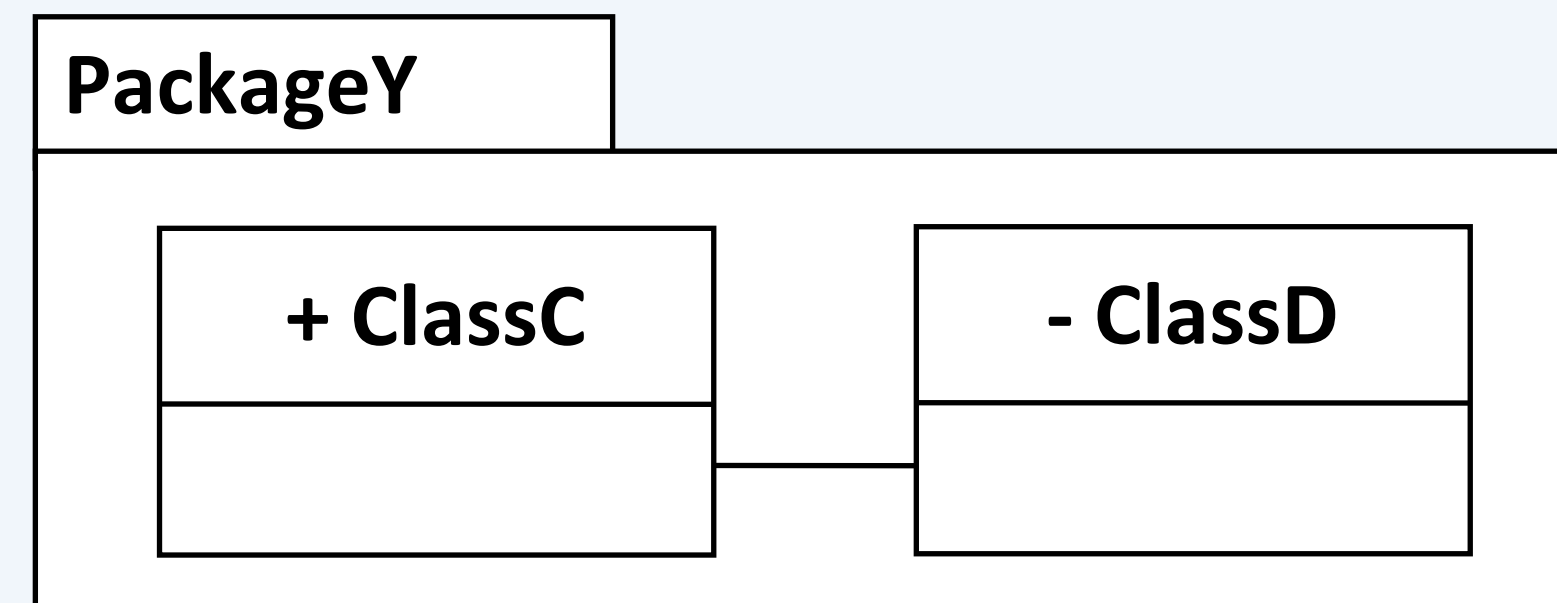
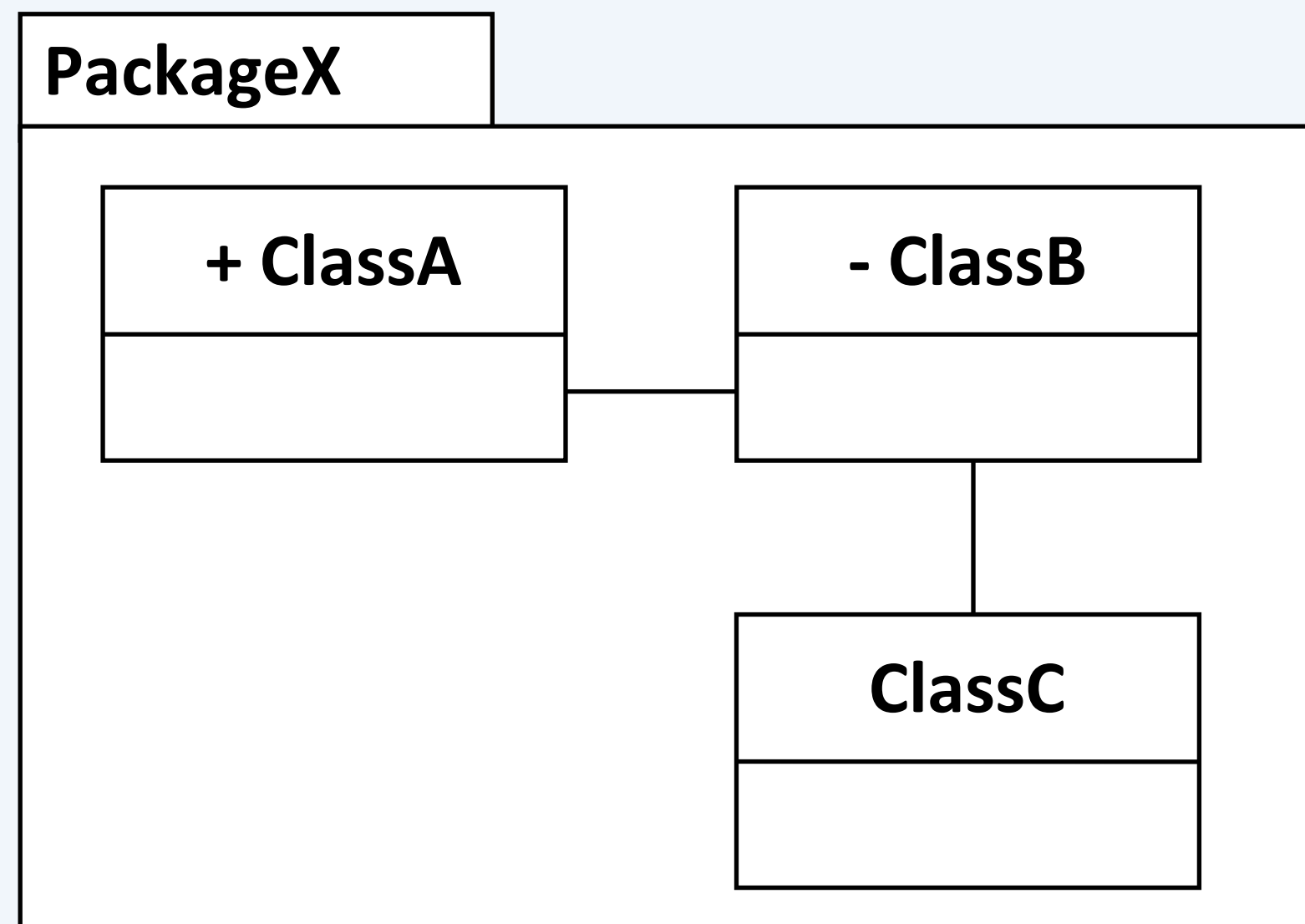
Christian Huemer und Marion Scholz

- UML-Abstraktionsmechanismus: Paket
- Modellelemente können höchstens **einem** Paket zugeordnet sein

- Partitionierungskriterien:
 - Funktionale Kohäsion
 - Informationskohäsion
 - Zugriffskontrolle
 - Verteilungsstruktur
 -
- Pakete bilden einen eigenen Namensraum
- Sichtbarkeit der Elemente kann definiert werden als "+" oder "-"

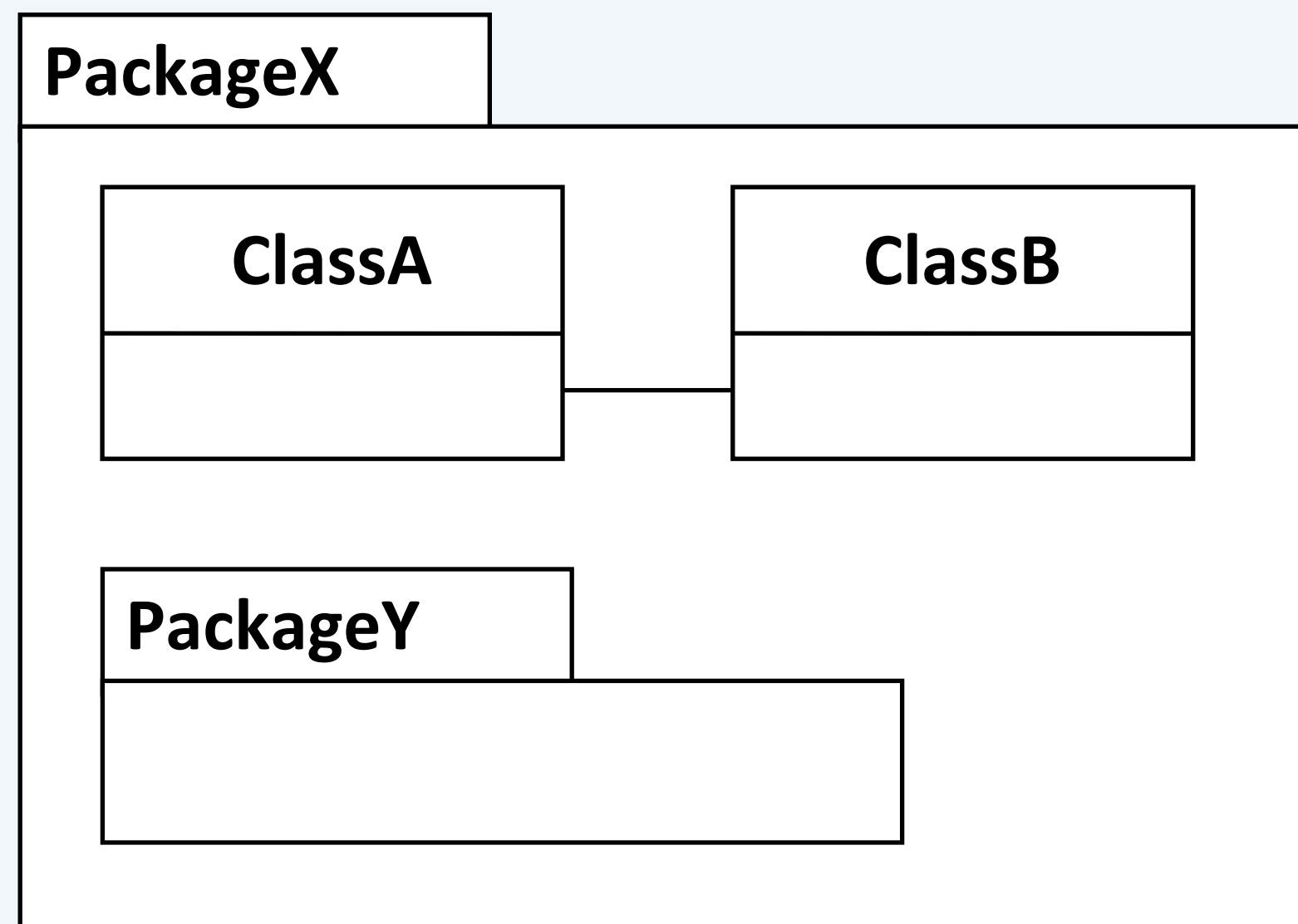
Verwendung von Elementen anderer Pakete

- Elemente eines Pakets benötigen Elemente eines anderen
- Qualifizierung dieser "externen" Elemente
 - Zugriff über qualifizierten Namen
 - Nur auf öffentliche Elemente eines Pakets

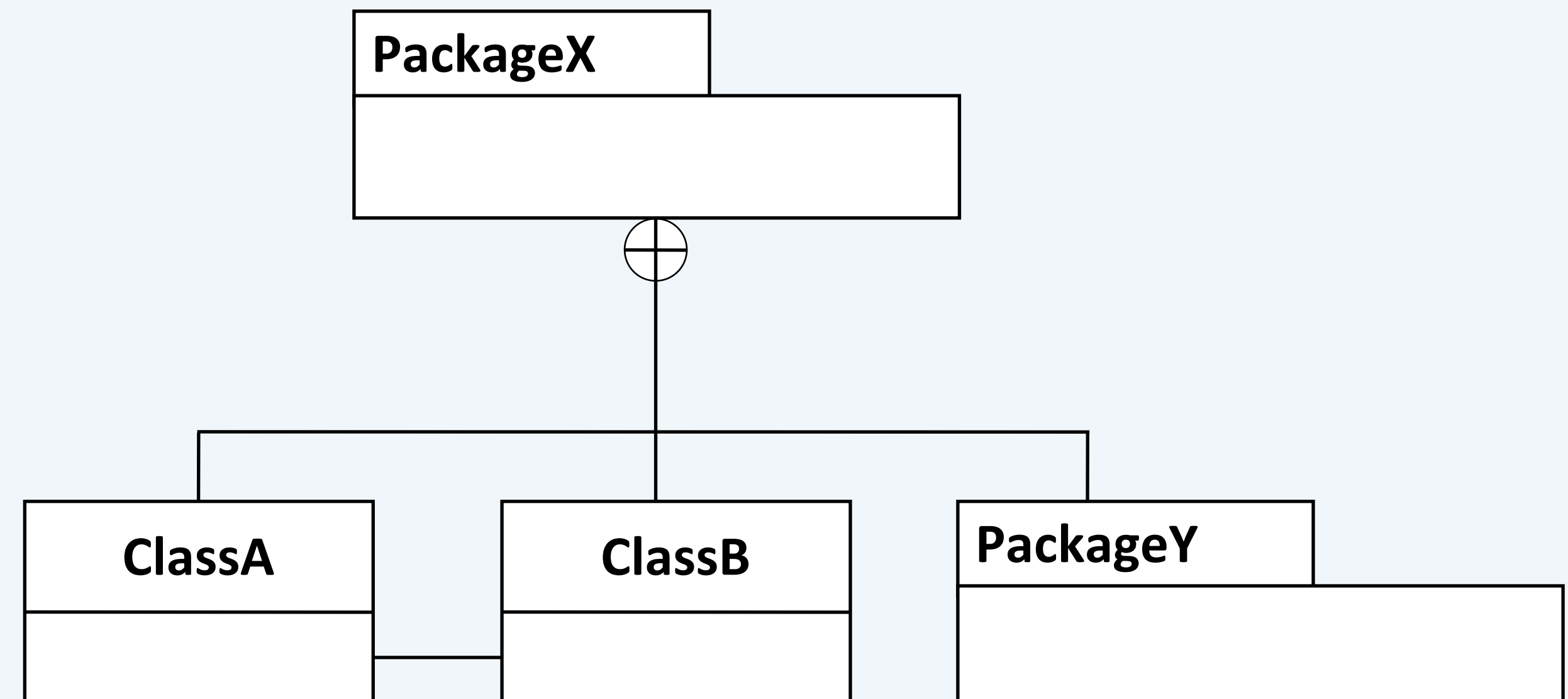


Hierarchien von Paketen

- Pakete können geschachtelt werden
 - Beliebige Tiefe
 - Paket-Hierarchie bildet einen Baum
- Zwei Darstellungsformen



oder



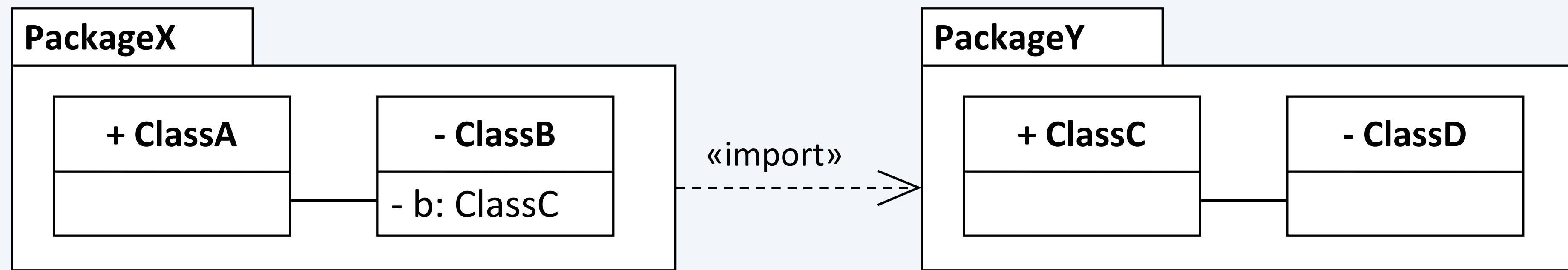
Import von Elementen und Paketen

- Import einzelner Elemente
 - Voraussetzung: Sichtbarkeit des Elements ist öffentlich
- Import ganzer Pakete
 - Äquivalent mit Element-Import aller öffentlich sichtbaren Elemente des importierten Pakets
- Sichtbarkeiten
 - Beim Import kann die Sichtbarkeit der importierten Elemente und Pakete neu bestimmt werden
 - Sichtbarkeit nur öffentlich oder privat ("+" oder "-")
 - «**import**»-Beziehungen für öffentliche Sichtbarkeit
 - «**access**»-Beziehungen für private Sichtbarkeit

Import von Elementen und Paketen – «import» (1/2)

■ Veränderung des Namensraums

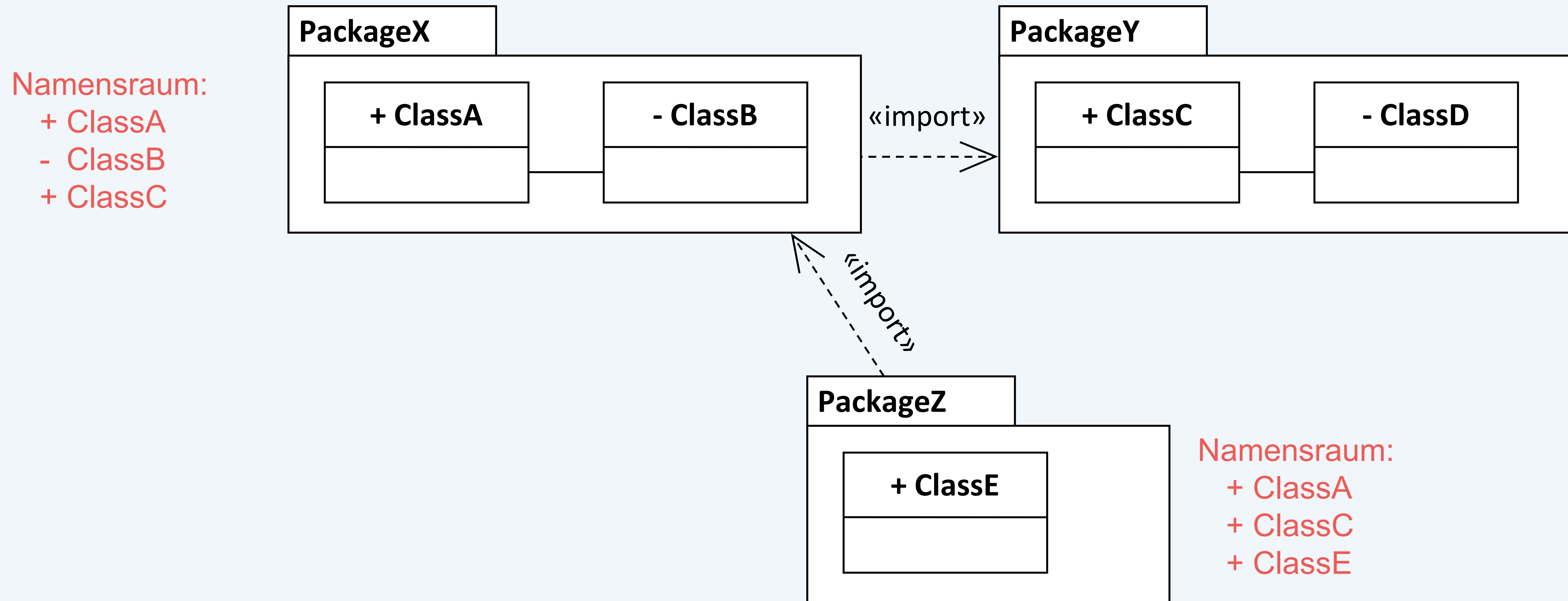
- Lädt die Namen des importierten Pakets in den Namensraum des Klienten
- Ändert damit den Namensraum des Klienten
- Qualifizierte Namen sind nicht mehr nötig



Import von Elementen und Paketen – «import» (2/2)

■ Transitivität

- Die importierten Namen sind öffentlich und finden somit bei erneutem Import Berücksichtigung



Import von Elementen und Paketen – «access»

- Nicht-transitiv
- Änderung der Sichtbarkeit der importierten Elemente auf privat

