

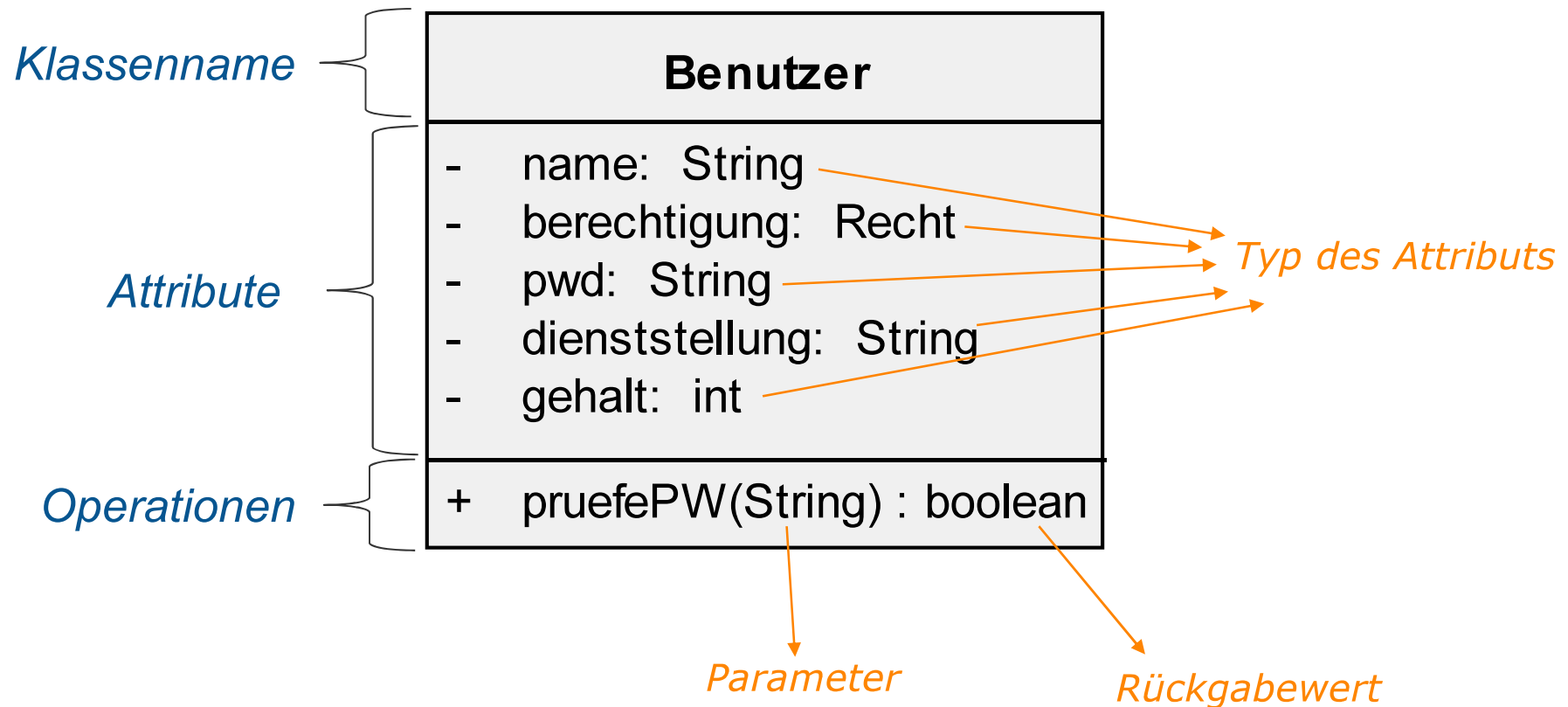
Inhalt

- **Klassendiagramm**
 - Klassen
 - Attribute und Operationen
 - Assoziationen
 - Schwache Aggregation
 - Starke Aggregation
 - Generalisierung
 - Zusammenfassendes Beispiel
 - Übersetzung nach Java
 - Datentypen in UML
- Objektdiagramm
- Paketdiagramm
- Abhängigkeiten

Klassendiagramm

- **Klasse in UML:** Schablone, Typ
- **Objekt:** Ausprägung einer Klasse
- **Klassendiagramm**
 - Beschreibt den **strukturellen Aspekt** eines Systems auf **Typebene** in Form von Klassen, Interfaces und Beziehungen

Notation für Klassen



Attribute und Operationen

■ Sichtbarkeiten von Attributen und Operationen:

- + ... public
- - ... private
- # ... protected
- ~ ... package (vgl. Java)

■ Eigenschaften von Attributen:

- „/“ attributname: abgeleitetes Attribut
 - Bsp.: /alter:int
- {optional}: Nullwerte sind erlaubt
- [n..m]: Multiplizität

Benutzer	
-	name: String
-	gebDatum: Date
-	/alter: int = {now() - gebDatum}
-	berechtigung: Recht
-	pwd: String
~	beschreibung: String {optional}
+	<u>anzahlBenutzer: int</u>
-	telefon: int [1..*]
+	pruefePW(String) : boolean
+	<u>ermittleAnzahlBenutzer() : int</u>

Klassenattribute/-operationen

Exkurs: Identifikation von Klassen

- Linguistische Analyse der Problembeschreibung nach R.J. Abbott, Program Design by Informal English Descriptions, CACM, Vol. 26, No. 11, 1983
- **Hauptwörter** herausfiltern
- **Faustregeln**
 - Eliminierung von irrelevanten Begriffen
 - Entfernen von Namen von Ausprägungen
 - Beseitigung vager Begriffe
 - Identifikation von Attributen
 - Identifikation von Operationen
 - Eliminierung von Begriffen, die zu Beziehungen aufgelöst werden können

Exkurs: Identifikation von Attributen

- **Adjektive** und **Mittelwörter** herausfiltern
- **Faustregeln**
 - Attribute beschreiben Objekte und sollten weder klassenwertig noch mehrwertig sein
 - abgeleitete Attribute sollten als solche gekennzeichnet werden
 - kontextabhängige Attribute sollten eher Assoziationen zugeordnet werden als Klassen
- Attribute sind i.A. nur unvollständig in der Anforderungsbeschreibung definiert

Exkurs: Identifikation von Operationen

- **Verben** herausfiltern
- **Faustregeln**
 - Welche Operationen kann man mit einem Objekt ausführen?
 - Nicht nur momentane Anforderungen berücksichtigen, sondern Wiederverwendbarkeit im Auge behalten
 - Welche Ereignisse können eintreten?
 - Welche Objekte können auf diese Ereignisse reagieren?
 - Welche anderen Ereignisse werden dadurch ausgelöst?

Bsp. - Bibliotheksverwaltung

Franz Müller soll neben anderen Leuten die Bibliothek der Universität benutzen können. Im Verwaltungssystem werden die **Benutzer** erfasst, von denen eine **eindeutige ID**, **Name** und **Adresse** bekannt sind, und die **Bücher**, von denen **Titel**, **Autor** und **ISBN-Nummer** gespeichert sind.

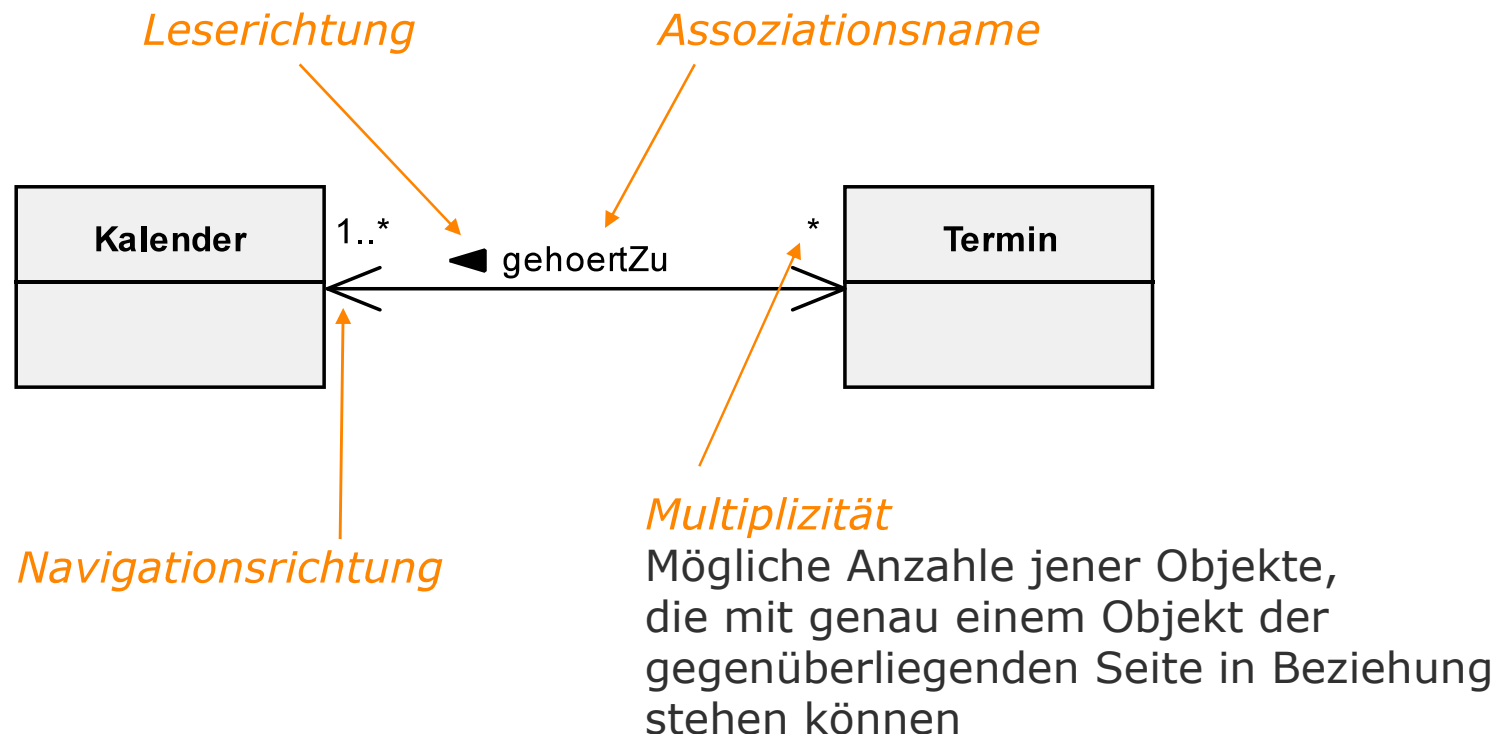
Buch
+ Titel: String
+ Autor: String
+ ISBN: int

Benutzer
+ ID: int
+ Name: String
+ Adresse: String

Frage: Was ist mit Franz Müller?

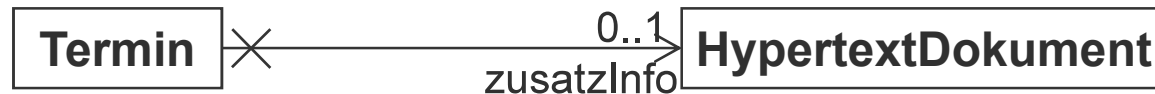
Assoziation

- Assoziationen zwischen Klassen modellieren mögliche **Objektbeziehungen** (*Links*) zwischen den **Instanzen der Klassen**



Assoziation: Navigationsrichtung

- Eine gerichtete Kante gibt an, **in welche Richtung die Navigation** von einem Objekt zu seinem Partnerobjekt **erfolgen kann**
- Ein **nicht-navigierbares Assoziationsende** wird durch ein "X" am Assoziationsende angezeigt



- Navigation von einem bestimmten Termin zum entsprechenden Dokument
- Umgekehrte Richtung - welche Termine beziehen sich auf ein bestimmtes Dokument? - wird nicht unterstützt
- **Ungerichtete Kanten** bedeuten "**keine Angabe** über Navigationsmöglichkeiten"
 - In Praxis wird oft bidirektionale Navigierbarkeit angenommen
- Die Angabe von Navigationsrichtungen stellt einen Hinweis für die spätere Entwicklung dar

Assoziation als Attribut

- Ein **navigierbares Assoziationsende** hat die gleiche Semantik wie ein Attribut der Klasse am gegenüberliegenden Assoziationsende
- Ein navigierbares 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

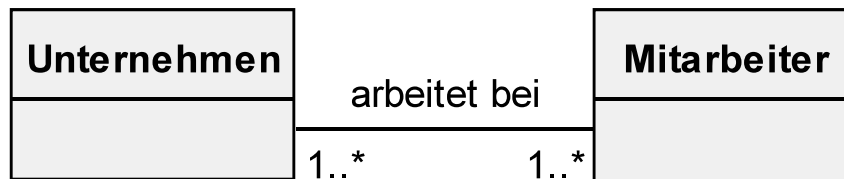


HypertextDokument

Assoziation: Beispiele



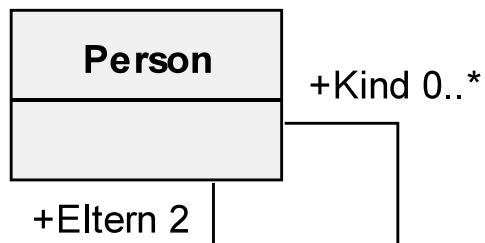
Ein Auto hat genau einen Besitzer, eine Person kann aber mehrere Autos besitzen (oder keines).



In einem Unternehmen arbeitet mind. ein Mitarbeiter, ein Mitarbeiter arbeitet mind. in einem Unternehmen



Eine Bestellung besteht aus 1-n Produkten, Produkte können beliebig oft bestellt werden. Von einer Bestellung kann festgestellt werden, welche Produkte sie beinhaltet.

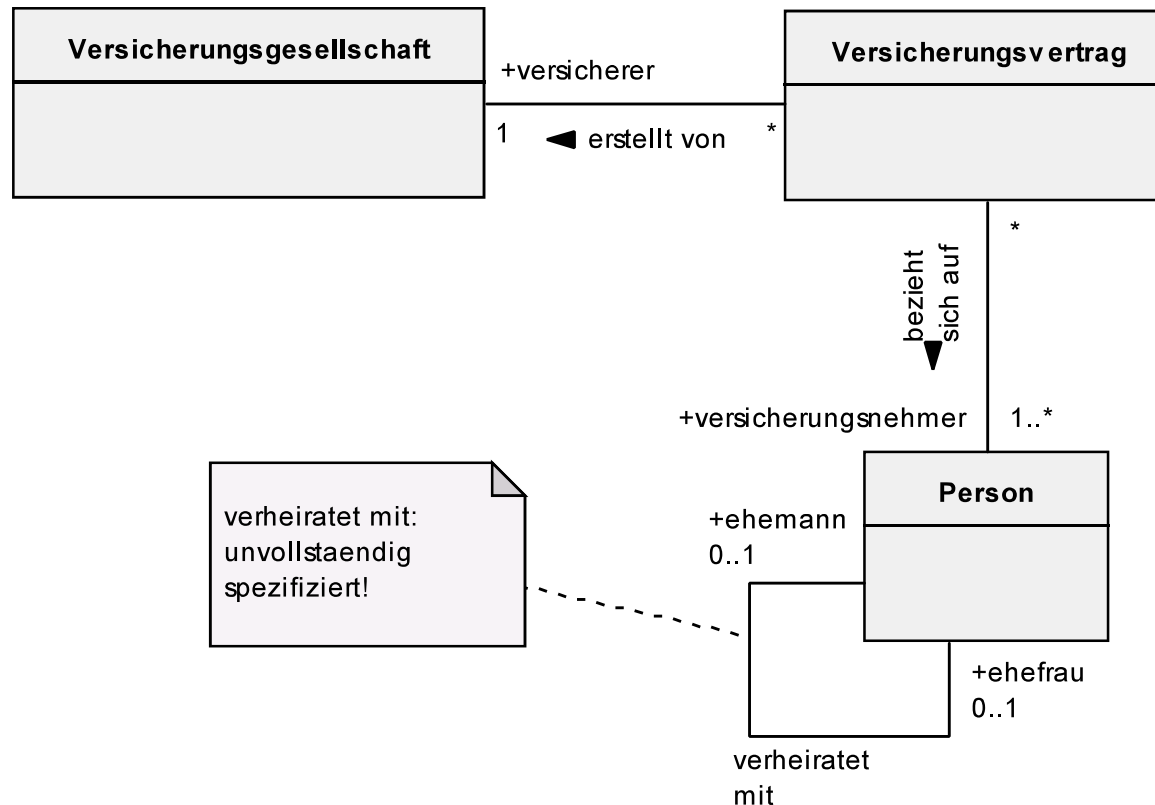


Eine Person hat 2 Eltern, die Personen sind, und 0 bis beliebig viele Kinder.
Ist durch dieses Modell ausgeschlossen, dass eine Person Kind von sich selbst ist?



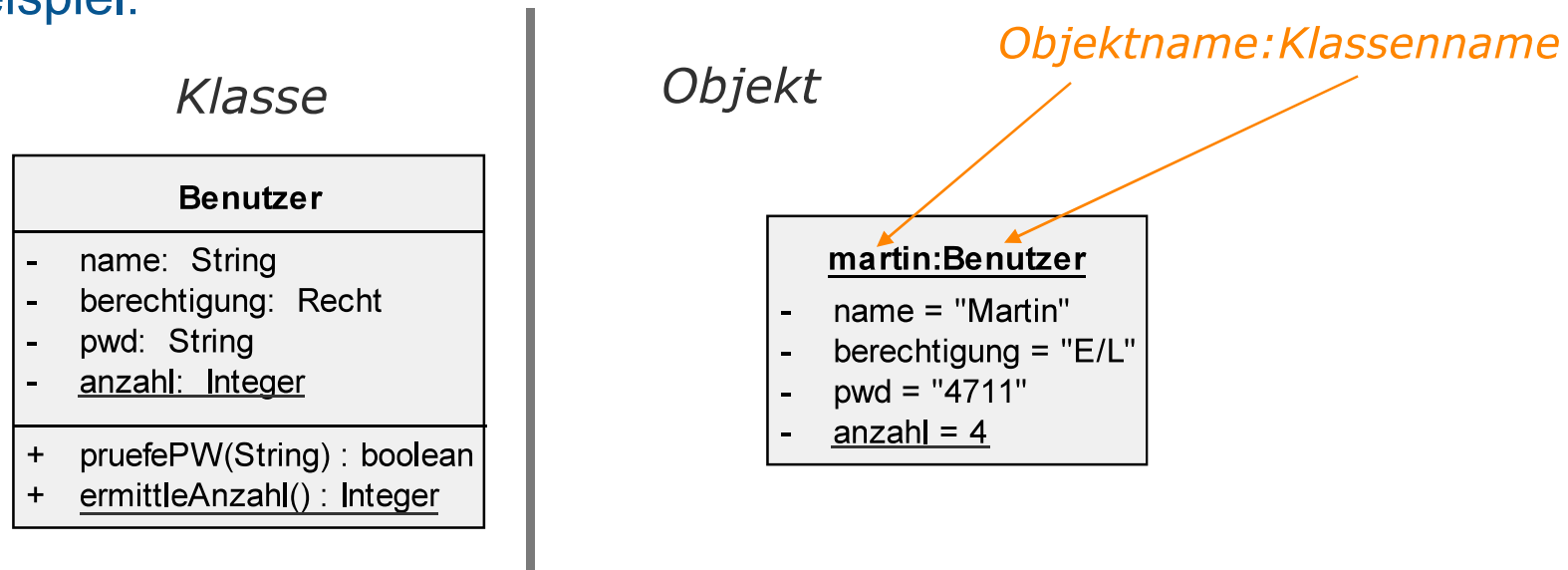
Assoziation: Rollen

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



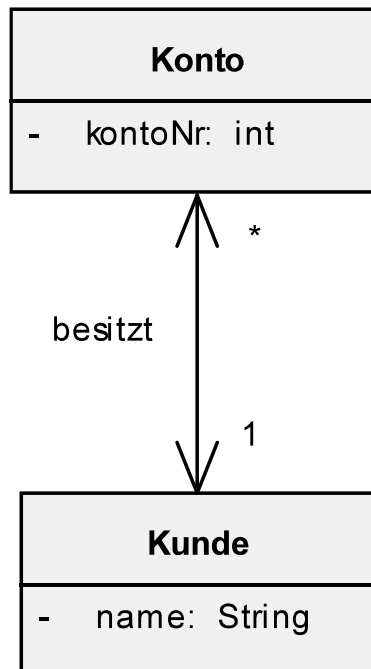
Objektdiagramm

- Beschreibt den strukturellen Aspekt eines Systems auf Instanzebene in Form von Objekten und Links
- Momentaufnahme (snapshot) des Systems – konkretes Szenario
- Ausprägung zu einem Klassendiagramm
- Eigentlich eine »Instanzspezifikation«
- Prinzipiell kann jede Diagrammart auf Instanzebene modelliert werden
- Beispiel:

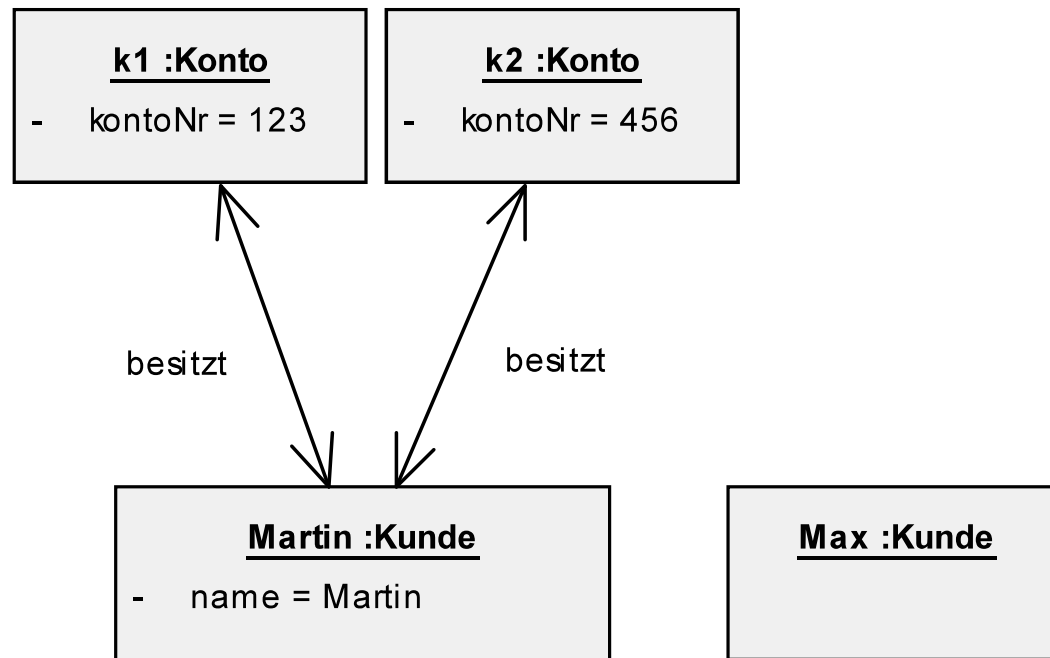


Objektdiagramm: Beispiel

Klassendiagramm

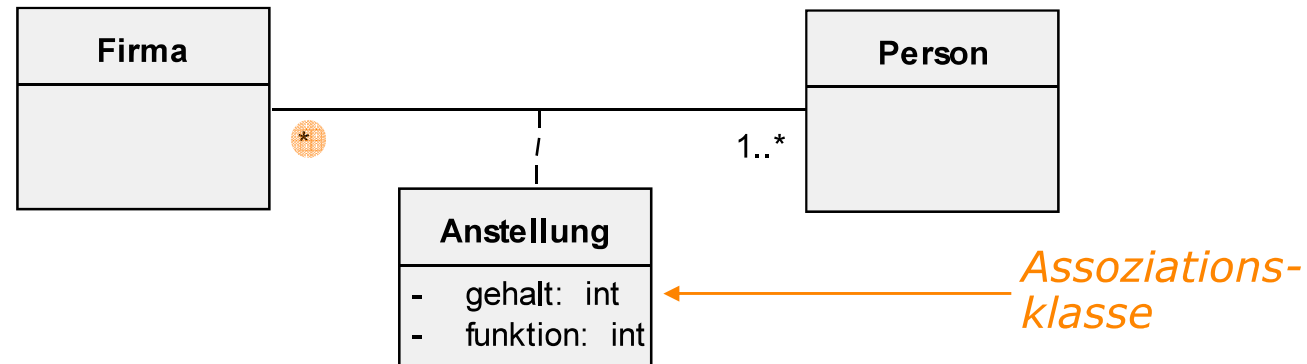


Objektdiagramm

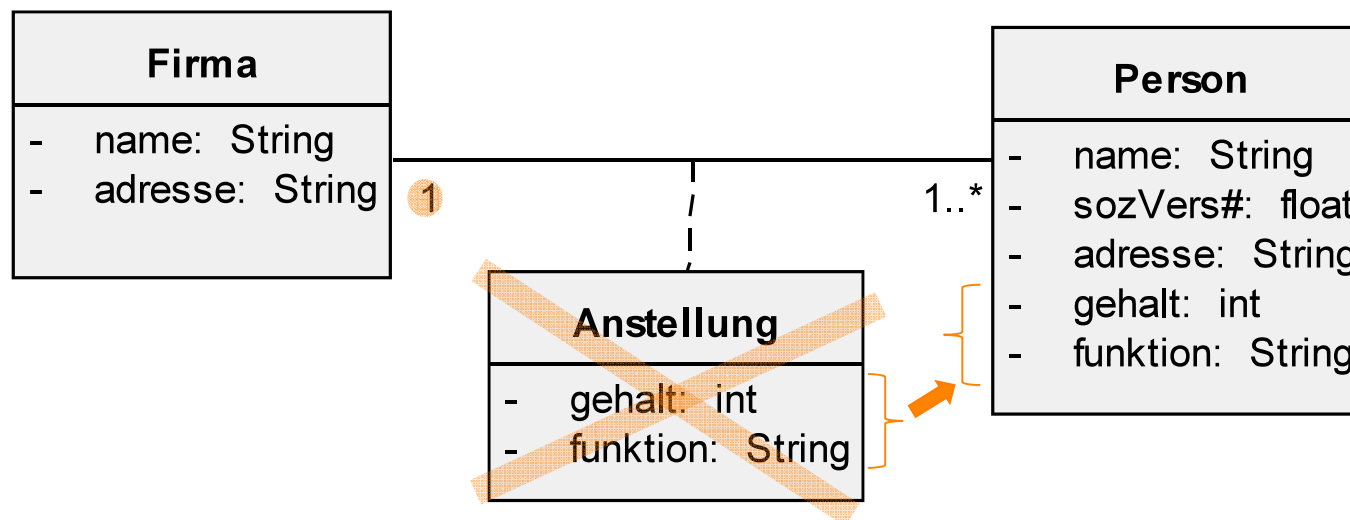


Assoziationsklasse (1/2)

- Kann **Attribute der Assoziation** enthalten
 - Bei m:n-Assoziationen mit Attributen notwendig

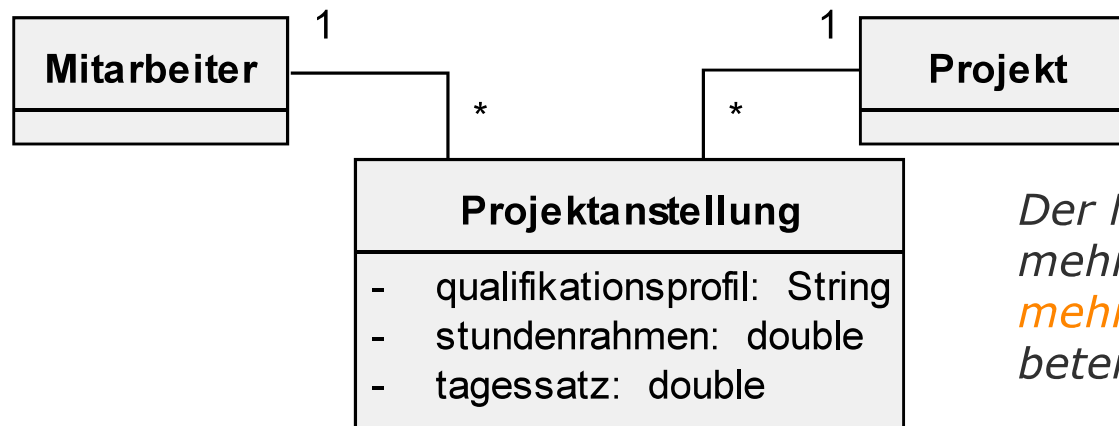


- Bei 1:1 und 1:n-Assoziationen sinnvoll aus Flexibilitätsgründen (Änderung der Multiplizität möglich)

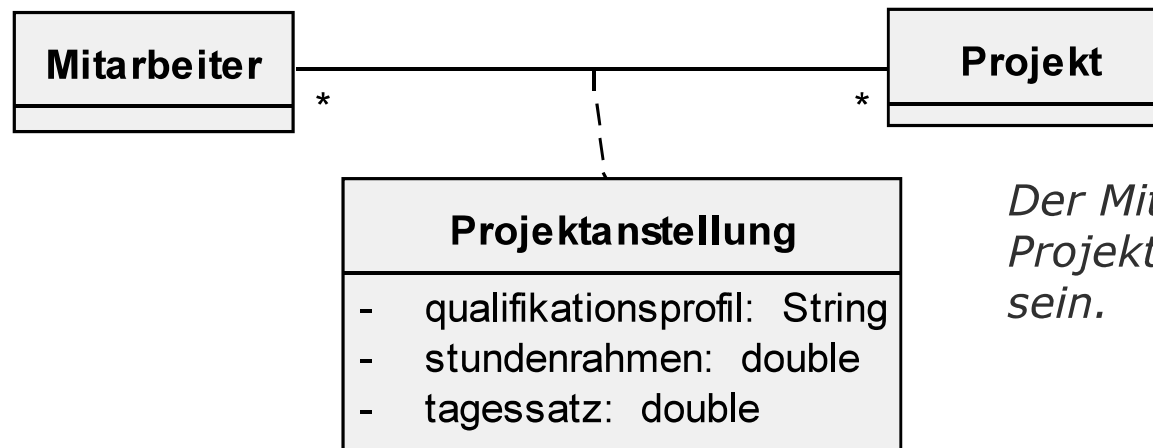


Assoziationsklasse (2/2)

- Normale Klasse ungleich Assoziationsklasse



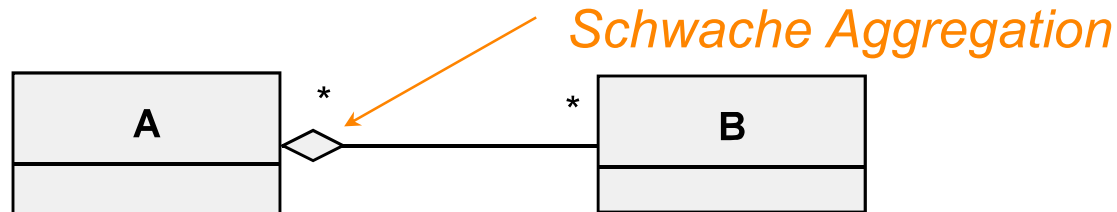
*Der Mitarbeiter M kann über mehrere Projektanstellungen **mehrfach** an dem Projekt P beteiligt sein.*



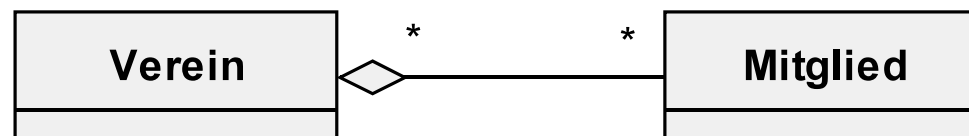
*Der Mitarbeiter M kann an dem Projekt P nur **einmal** beteiligt sein.*



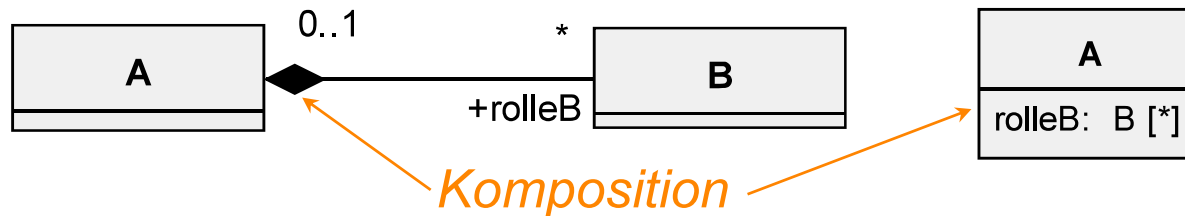
Schwache Aggregation



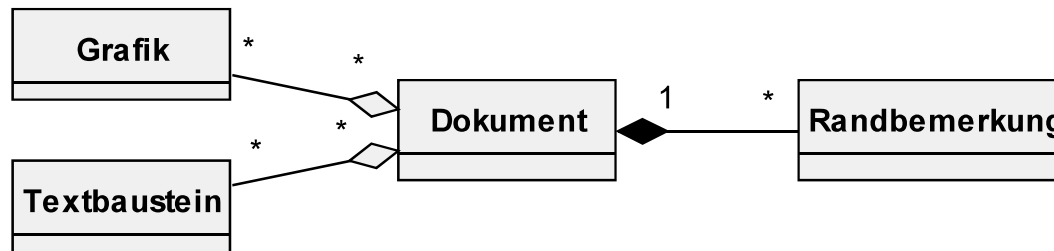
- Schwache Zugehörigkeit der Teile, d.h. Teile sind **unabhängig** von ihrem Ganzen
- Die Multiplizität des aggregierenden Endes der Beziehung (Raute) kann > 1 sein
- Es gilt nur eingeschränkte **Propagierungssemantik**
- Die zusammengesetzten Objekte bilden einen **gerichteten, azyklischen Graphen**



Starke Aggregation (= Komposition)

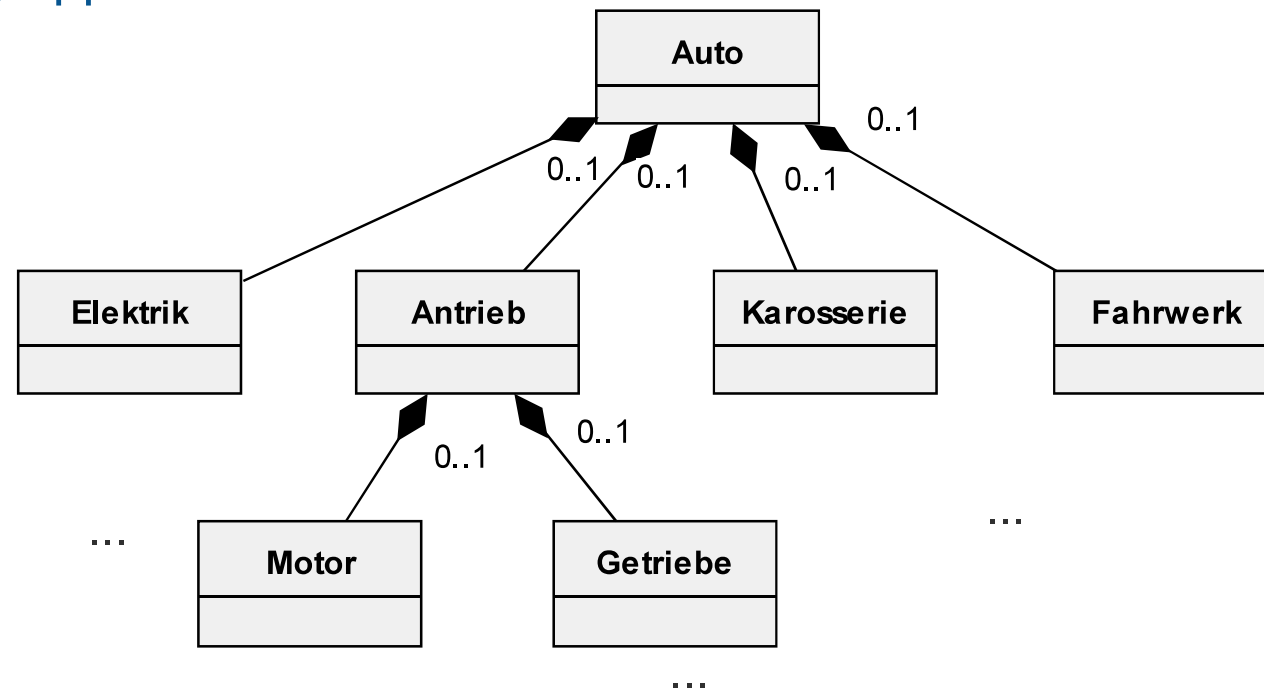


- Ein bestimmter Teil darf zu einem bestimmten Zeitpunkt in **maximal einem zusammengesetzten Objekt enthalten** sein
- Die **Multiplizität** des aggregierenden Endes der Beziehung kann (maximal) 1 sein
- **Abhängigkeit** der Teile vom zusammengesetzten Objekt
- **Propagierungssemantik**
- Die zusammengesetzten Objekte bilden einen **Baum**



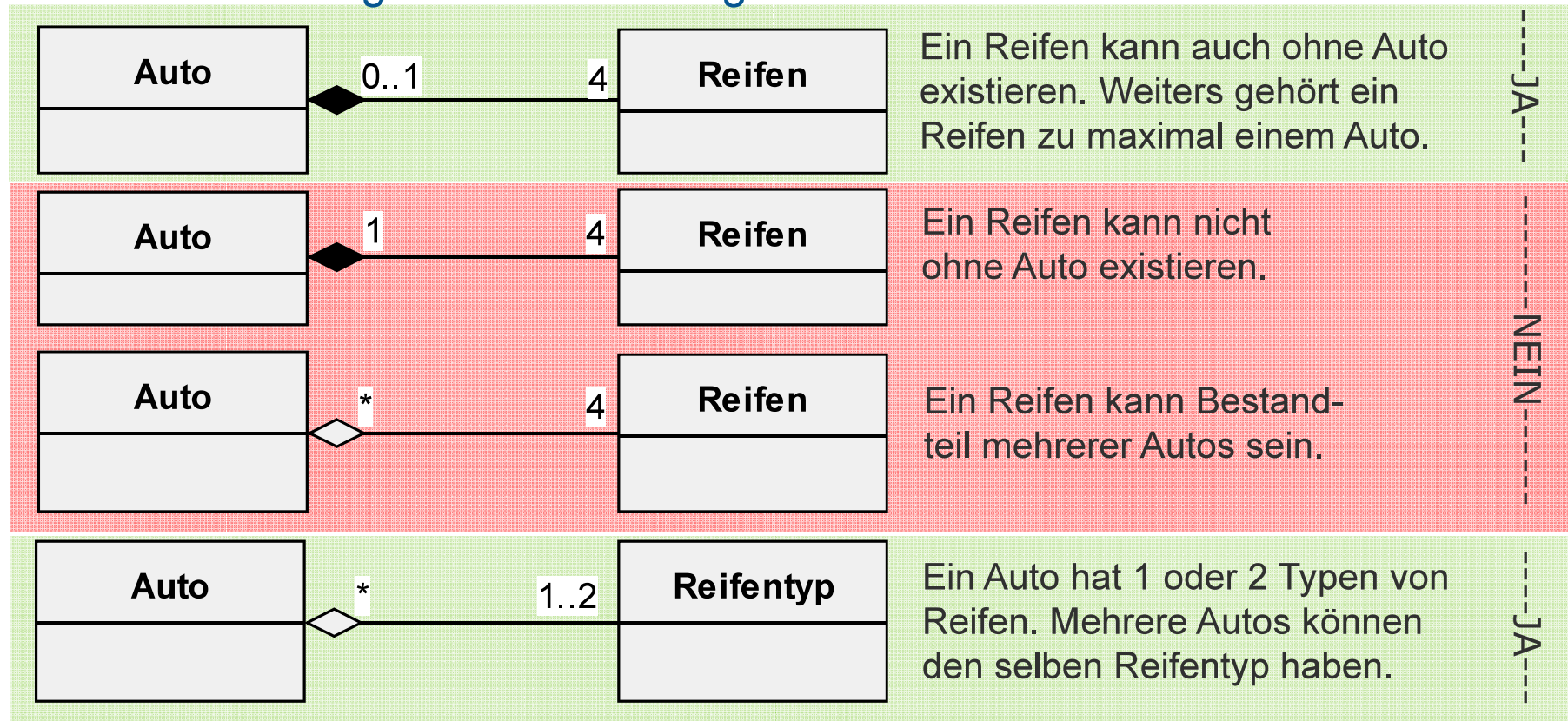
Starke Aggregation

- Mittels starker Aggregation kann eine Hierarchie von „Teil-von“-Beziehungen dargestellt werden (Transitivität!)
- Beispiel: Baugruppen von Auto



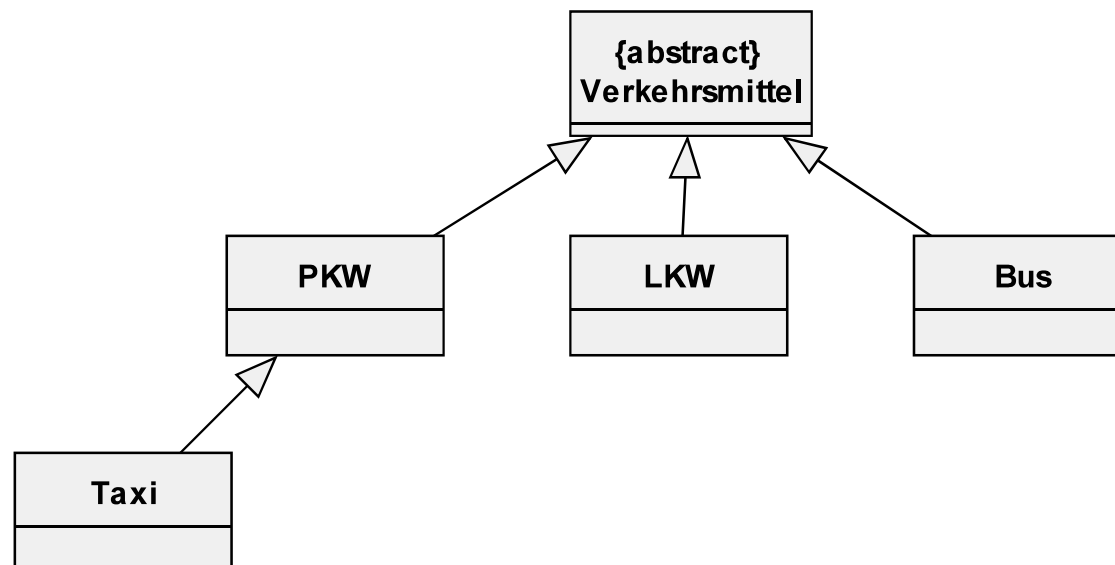
Komposition und Aggregation

- Welche der folgenden Beziehungen trifft zu?



Generalisierung

- **Taxonomische Beziehung** zwischen einer spezialisierten Klasse und einer allgemeineren Klasse
 - Die spezialisierte Klasse **erbt** die Eigenschaften der allgemeineren Klasse
 - 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!)



Abstrakte Klasse (1/2)

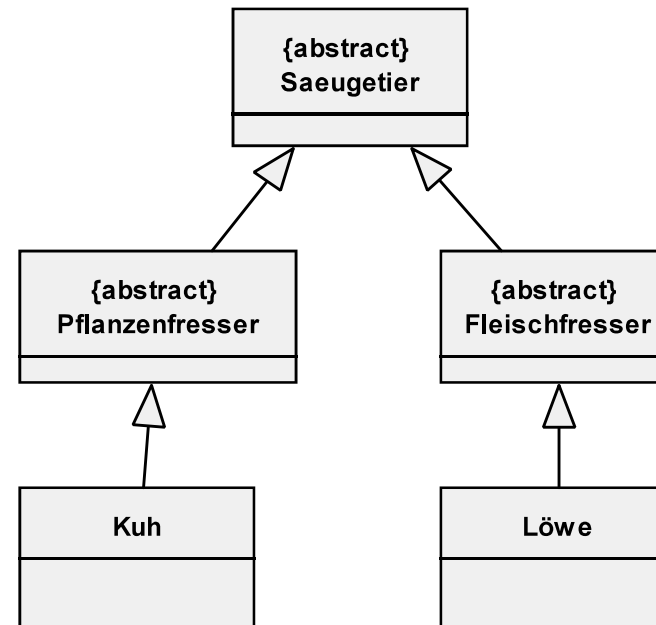
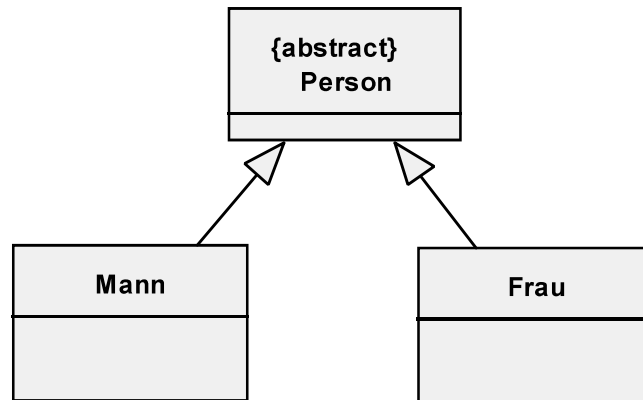
- Klasse, die **nicht instanziiert** werden kann
- **Nur in Generalisierungshierarchien** sinnvoll
- Dient zum "**Herausheben**" **gemeinsamer Merkmale** einer Reihe von Unterklassen
- Notation: Schlüsselwort {abstract} oder Klassenname in kursiver Schrift



- Mit analoger Notation wird zwischen konkreten (= implementierten) und abstrakten (= nur spezifizierten) **Operationen** einer Klasse unterschieden

Abstrakte Klasse (2/2)

- Beispiele:

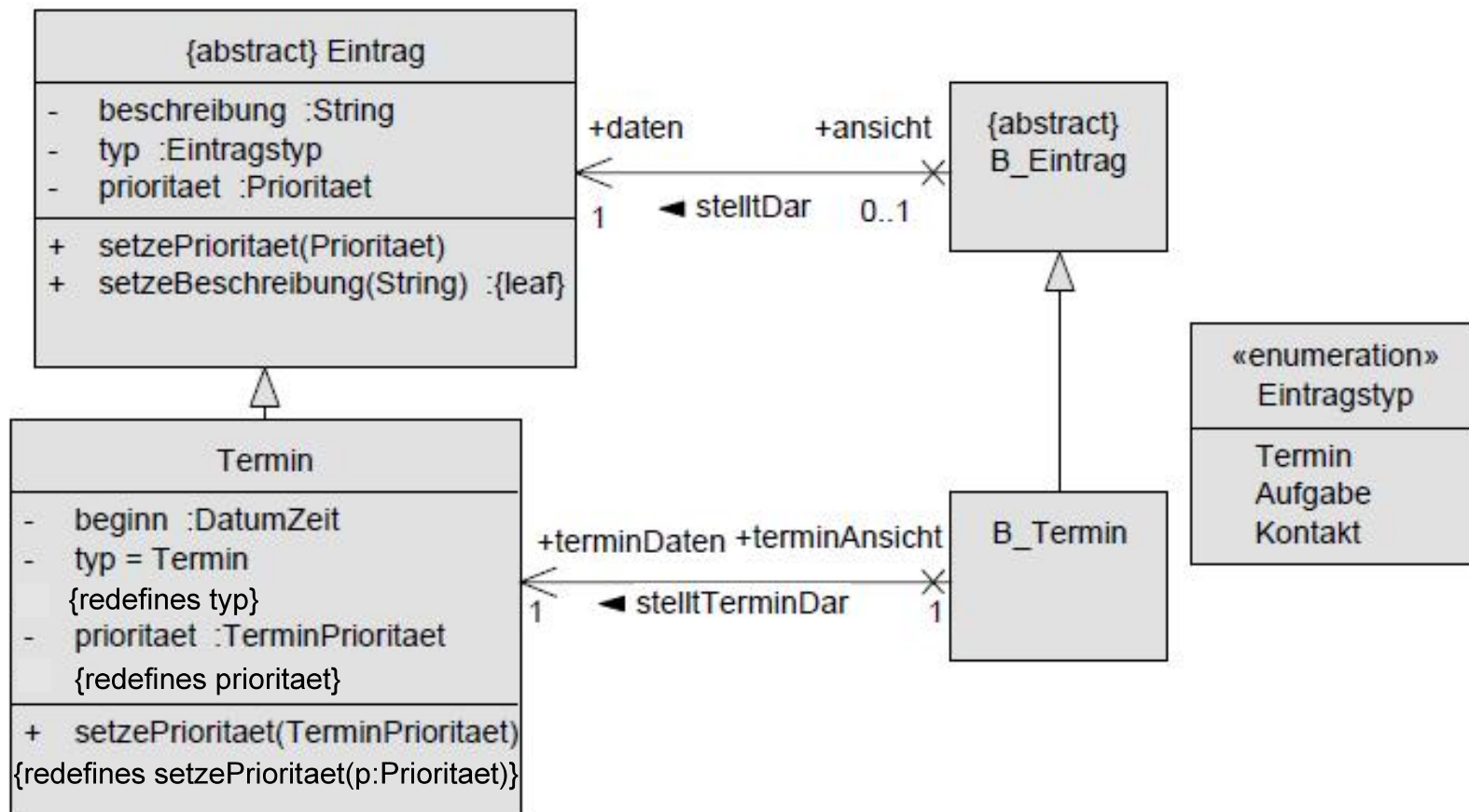


Generalisierung: Redefinition von geerbten Merkmalen (1/2)

- Geerbte Merkmale können **in Subklasse redefiniert** werden
 - {redefines <feature>}
- **Redefinierbare Merkmale**
 - Attribute
 - Navigierbare Assoziationsenden
 - Operationen
- Redefinition von Operationen in (in)direkten Subklassen kann auch verhindert werden, indem die Operation mit der Eigenschaft {leaf} gekennzeichnet wird
- Das redefinierte Merkmal muss konsistent zum ursprünglichen Merkmal sein – verschiedenste **Konsistenzregeln** – z.B.
 - Ein redefiniertes Attribut ist konsistent zum ursprünglichen Attribut, wenn sein **Typ gleich oder ein Subtyp** des ursprünglichen Typs ist
 - Das **Intervall der Multiplizität** muss in jenem des ursprünglichen Attributs **enthalten** sein
 - Die Signatur einer Operation muss die **gleiche Anzahl an Parametern** aufweisen, etc.



Generalisierung: Redefinition von geerbten Merkmalen (2/2)



Aufgabenstellung

- Gesucht ist ein vereinfacht dargestelltes Modell der TU Wien entsprechend der folgenden Spezifikation.

Die TU besteht aus mehreren Fakultäten, die sich wiederum aus verschiedenen Instituten zusammensetzen. Jede Fakultät und jedes Institut besitzt eine Bezeichnung. Für jedes Institut ist eine Adresse bekannt. Jede Fakultät wird von ihrem Dekan, einem Mitarbeiter, geleitet.

Die Gesamtanzahl der Mitarbeiter ist bekannt. Mitarbeiter haben eine Sozialversicherungsnummer, einen Namen und eine E-Mail-Adresse. Es wird zwischen wissenschaftlichem und nicht-wissenschaftlichem Personal unterschieden.

Wissenschaftliche Mitarbeiter sind zumindest einem Institut zugeordnet.

Für jeden wissenschaftlichen Mitarbeiter ist seine Fachrichtung bekannt.

Weiters können wissenschaftliche Mitarbeiter für eine gewisse Anzahl an Stunden an Projekten beteiligt sein, von welchen ein Name und Anfangs- und Enddatum bekannt sind.

Manche wissenschaftliche Mitarbeiter führen Lehrveranstaltungen durch – diese werden als Vortragende bezeichnet. LVAs haben eine ID, einen Namen und eine Stundenanzahl.

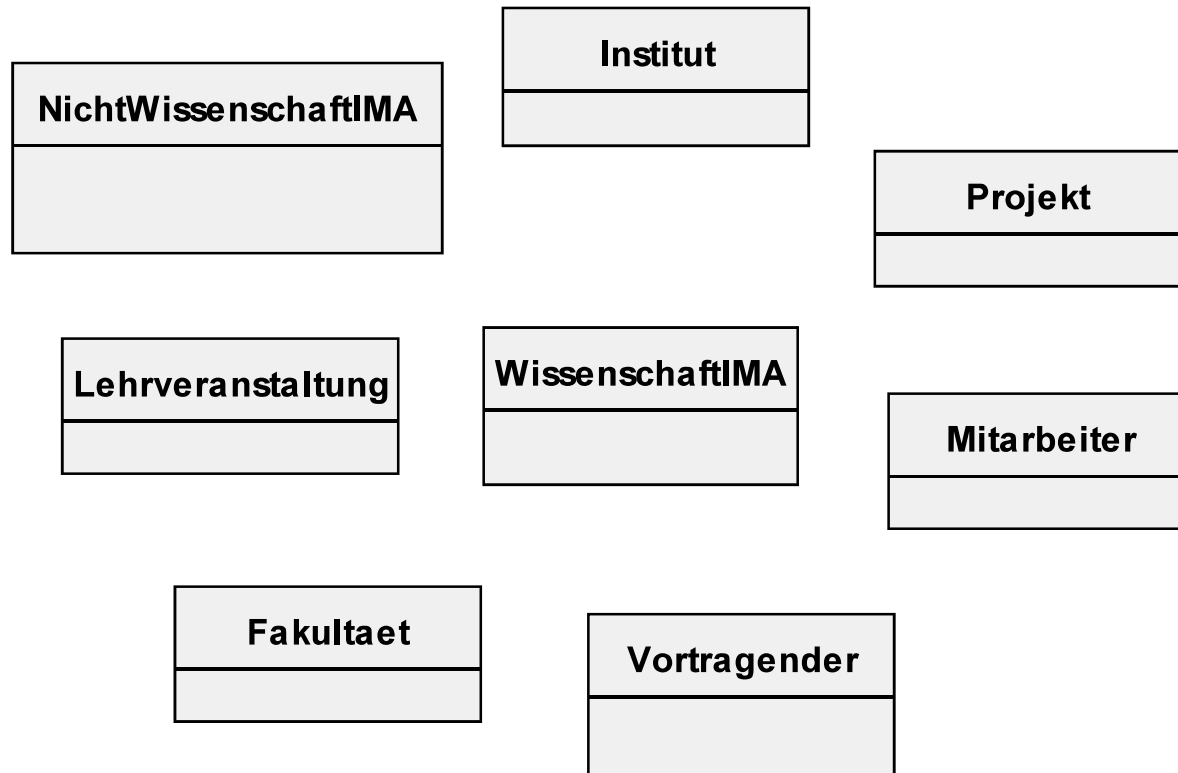
Identifikation von Klassen (1/2)

Die TU besteht aus mehreren **Fakultäten**, die sich wiederum aus verschiedenen **Instituten** zusammensetzen. Jede Fakultät und jedes Institut besitzt eine Bezeichnung. Für jedes Institut ist eine Adresse bekannt. Jede Fakultät wird von ihrem Dekan, einem Mitarbeiter, geleitet. Die Gesamtanzahl der **Mitarbeiter** ist bekannt. Mitarbeiter haben eine Sozialversicherungsnummer, einen Namen und eine E-Mail-Adresse. Es wird zwischen **wissenschaftlichem** und **nicht-wissenschaftlichem Personal** unterschieden.

Wissenschaftliche Mitarbeiter sind zumindest einem Institut zugeordnet. Für jeden wissenschaftlichen Mitarbeiter ist seine Fachrichtung bekannt. Weiters können wissenschaftliche Mitarbeiter für eine gewisse Anzahl an Stunden an **Projekten** beteiligt sein, von welchen ein Name und Anfangs- und Enddatum bekannt sind.

Manche wissenschaftliche Mitarbeiter führen **Lehrveranstaltungen** durch – diese werden als **Vortragende** bezeichnet. LVAs haben eine ID, einen Namen und eine Stundenanzahl.

Identifikation von Klassen (2/2)



Identifikation von Attributen (1/2)

Die TU besteht aus mehreren Fakultäten, die sich wiederum aus verschiedenen Instituten zusammensetzen. Jede Fakultät und jedes Institut besitzt eine **Bezeichnung**. Für jedes Institut ist eine **Adresse** bekannt. Jede Fakultät wird von ihrem Dekan, einem Mitarbeiter, geleitet. Die **Gesamtanzahl** der Mitarbeiter ist bekannt. Mitarbeiter haben eine **Sozialversicherungsnummer**, einen **Namen** und eine **E-Mail-Adresse**. Es wird zwischen wissenschaftlichem und nicht-wissenschaftlichem Personal unterschieden.

Wissenschaftliche Mitarbeiter sind zumindest einem Institut zugeordnet. Für jeden wissenschaftlichen Mitarbeiter ist seine **Fachrichtung** bekannt. Weiters können wissenschaftliche Mitarbeiter für eine **gewisse Anzahl an Stunden** an Projekten beteiligt sein, von welchen ein **Name** und **Anfangs- und Enddatum** bekannt sind.

Manche wissenschaftliche Mitarbeiter führen Lehrveranstaltungen durch - diese werden als Vortragende bezeichnet. LVAs haben eine **ID**, einen **Namen** und eine **Stundenanzahl**.

Identifikation von Attributen (2/2)

Mitarbeiter
+ svnr: int
+ name: String
+ email: String
+ <u>anzahl: int</u>

Fakultaet
+ bezeichnung: String

Institut
+ bezeichnung: String
+ adresse: String

Lehrveranstaltung
+ name: String
+ id: int
+ stunden: float

WissenschaftlMA
+ fachrichtung: String

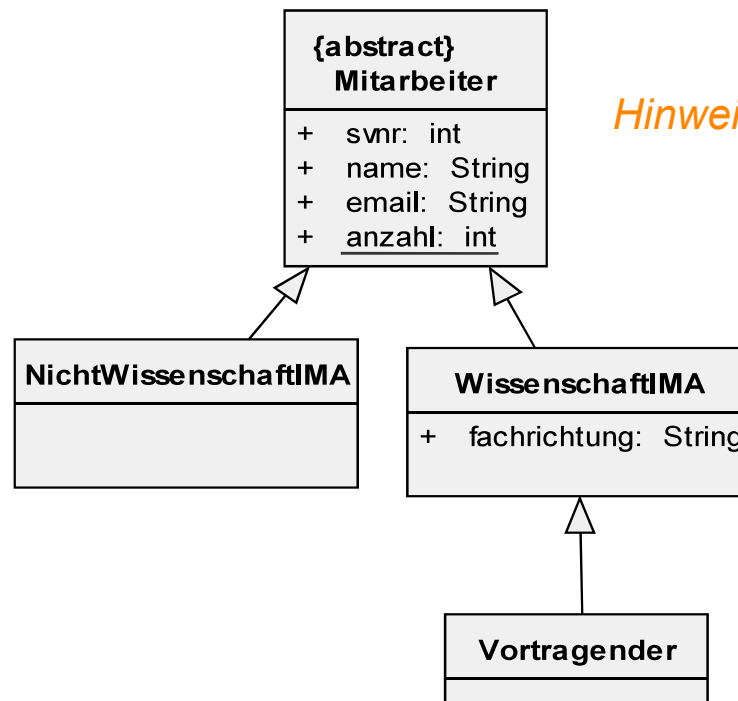
Projekt
+ name: String
+ beginn: Date
+ ende: Date

Vortragender

NichtWissenschaftlMA

Generalisierung

- Es wird zwischen wissenschaftlichem und nicht-wissenschaftlichem Personal unterschieden.
- Manche wissenschaftliche Mitarbeiter führen Lehrveranstaltungen durch – diese werden als Vortragende bezeichnet.

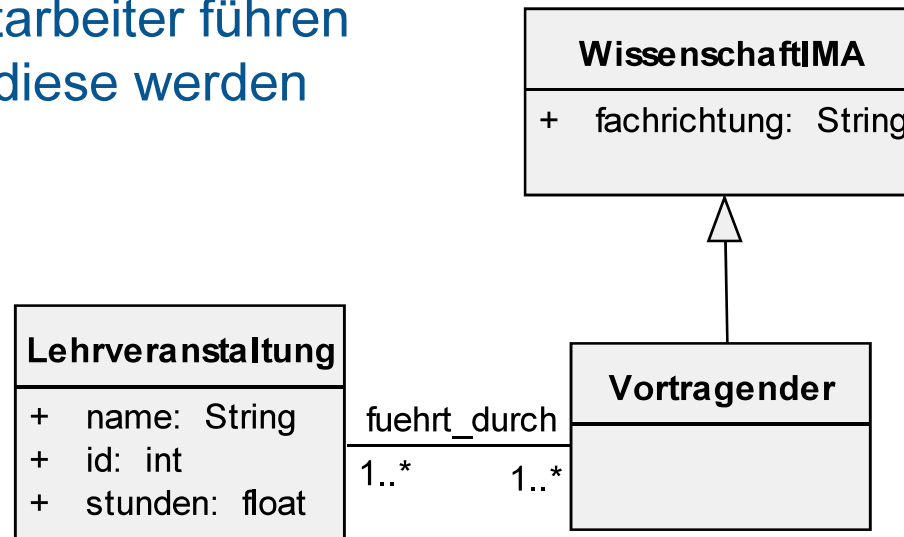


Hinweis: Mitarbeiter ist nun eine abstrakte Klasse.

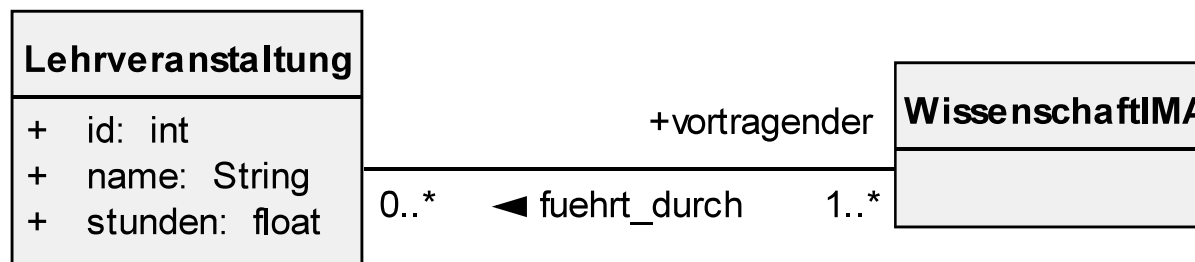


Assoziation (1/2)

- Manche wissenschaftliche Mitarbeiter führen Lehrveranstaltungen durch – diese werden als Vortragende bezeichnet.

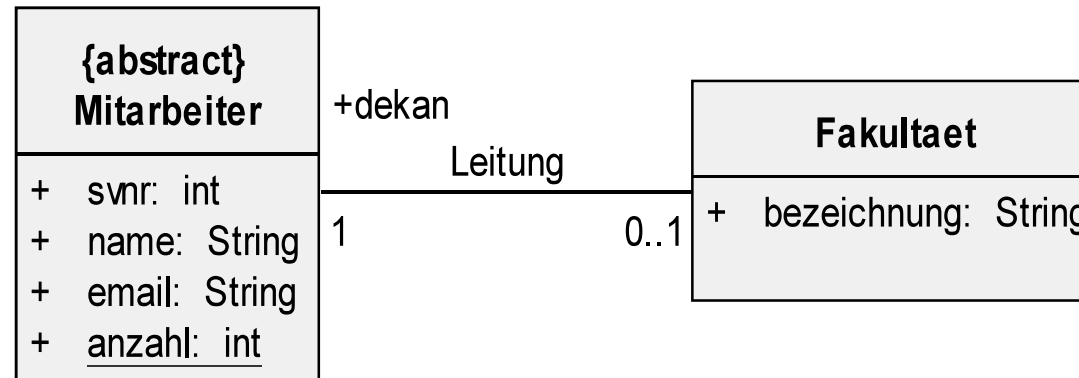


- Alternative Modellierung:



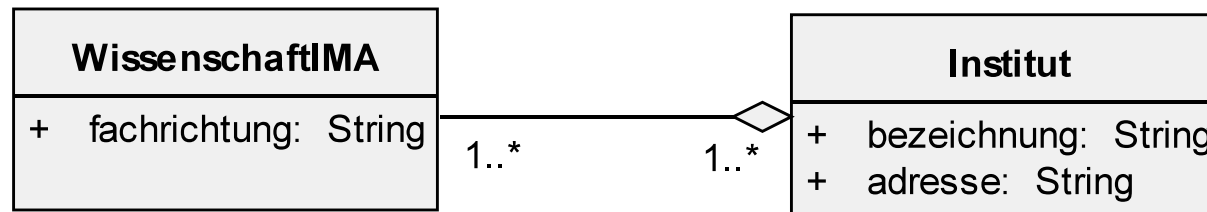
Assoziation (2/2)

- Jede Fakultät wird von ihrem Dekan, einem Mitarbeiter, geleitet.



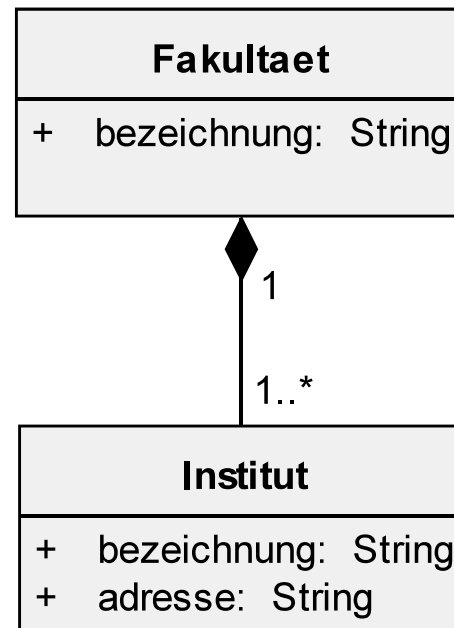
Schwache Aggregation

- Wissenschaftliche Mitarbeiter sind zumindest einem Institut zugeordnet.



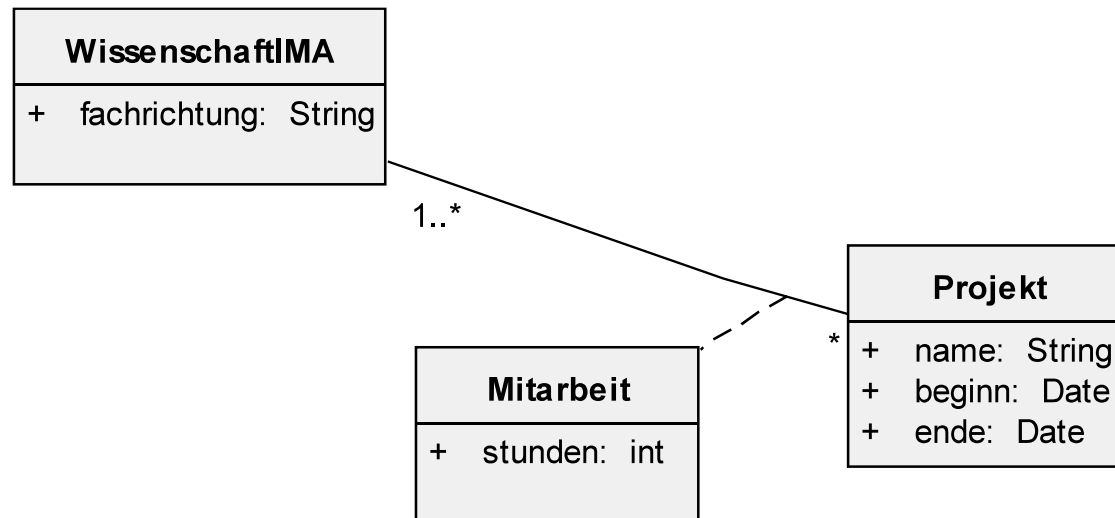
Starke Aggregation

- Die TU besteht aus mehreren Fakultäten, die sich wiederum aus verschiedenen Instituten zusammensetzen.

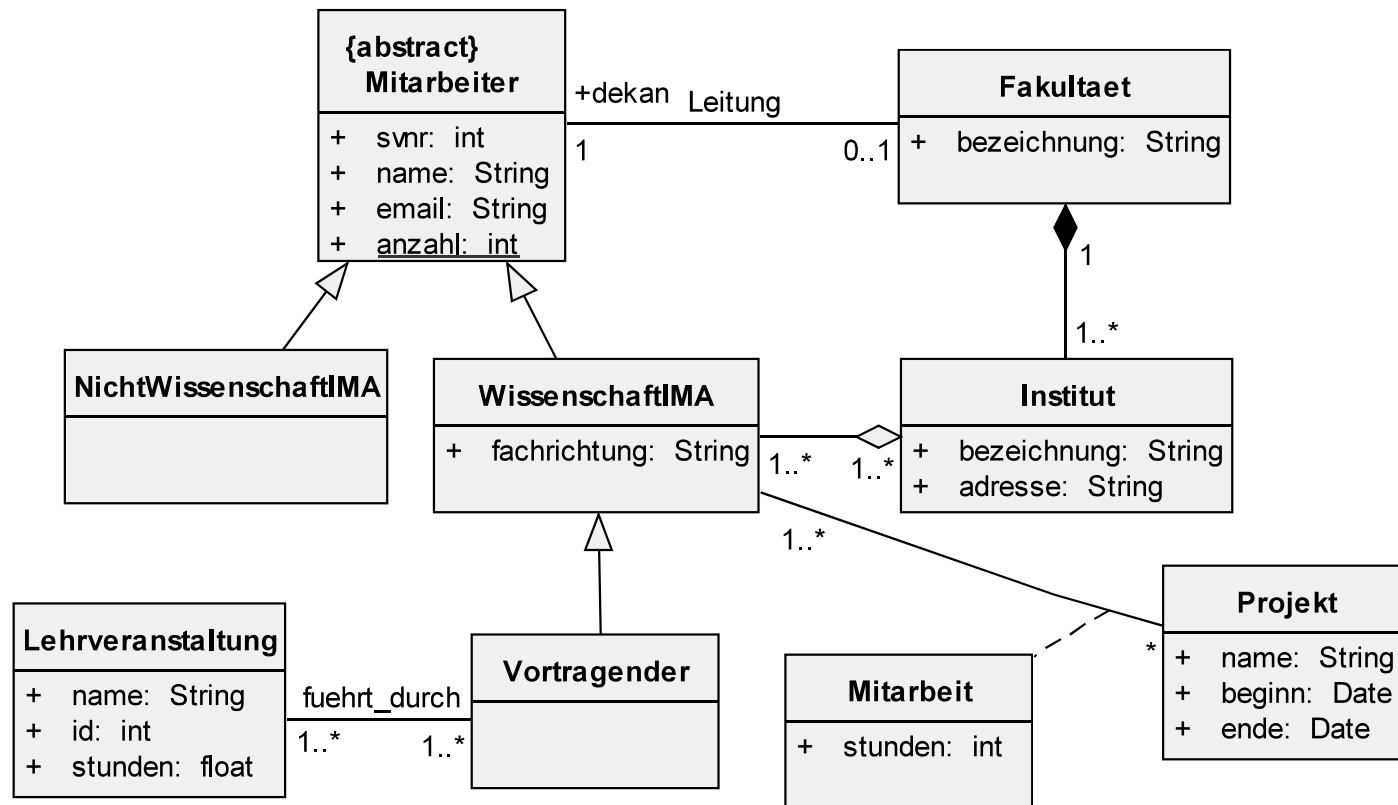


Assoziationsklasse

- Weiters können wissenschaftliche Mitarbeiter an Projekten beteiligt sein.



... das gesamte Diagramm



Übersetzung nach Java:

Klassen (1/2)

Lehrveranstaltung
+ name: String
+ id: int
+ stunden: float



```
class Lehrveranstaltung {  
  
    public String name;  
    public int id;  
    public float stunden;  
  
    public Lehrveranstaltung(String name, int id,  
                             float stunden) {  
  
        this.name = name;  
        this.id = id;  
        this.stunden = stunden;  
    }  
}
```

- Erstellung einer konkreten Instanz oom:

```
Lehrveranstaltung oom = new Lehrveranstaltung(  
    "Objektorientierte Modellierung", 394, 4);
```

- Zugriff auf Attribut name: oom.name;

Übersetzung nach Java:

Klassen (2/2)

Lehrveranstaltung
- name: String - id: int - stunden: float
+ getName() : String + getID() : int + getStunden() : float



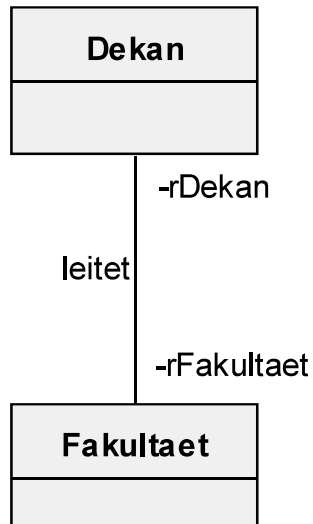
```
class Lehrveranstaltung {  
  
    private String name;  
    private int id;  
    private float stunden;  
  
    public Lehrveranstaltung(String name, int id,  
                             float stunden) {  
        this.name = name;  
        this.id = id;  
        this.stunden = stunden;  
    }  
    public String getName() { return name; }  
    public int getId() { return id; }  
    public float getStunden() { return stunden; }  
}
```

- *Erstellung einer Instanz oom: wie vorher*
- *Zugriff auf Attribut name:* ~~oom.name~~; oom.getName();



Übersetzung nach Java:

1:1-Assoziation

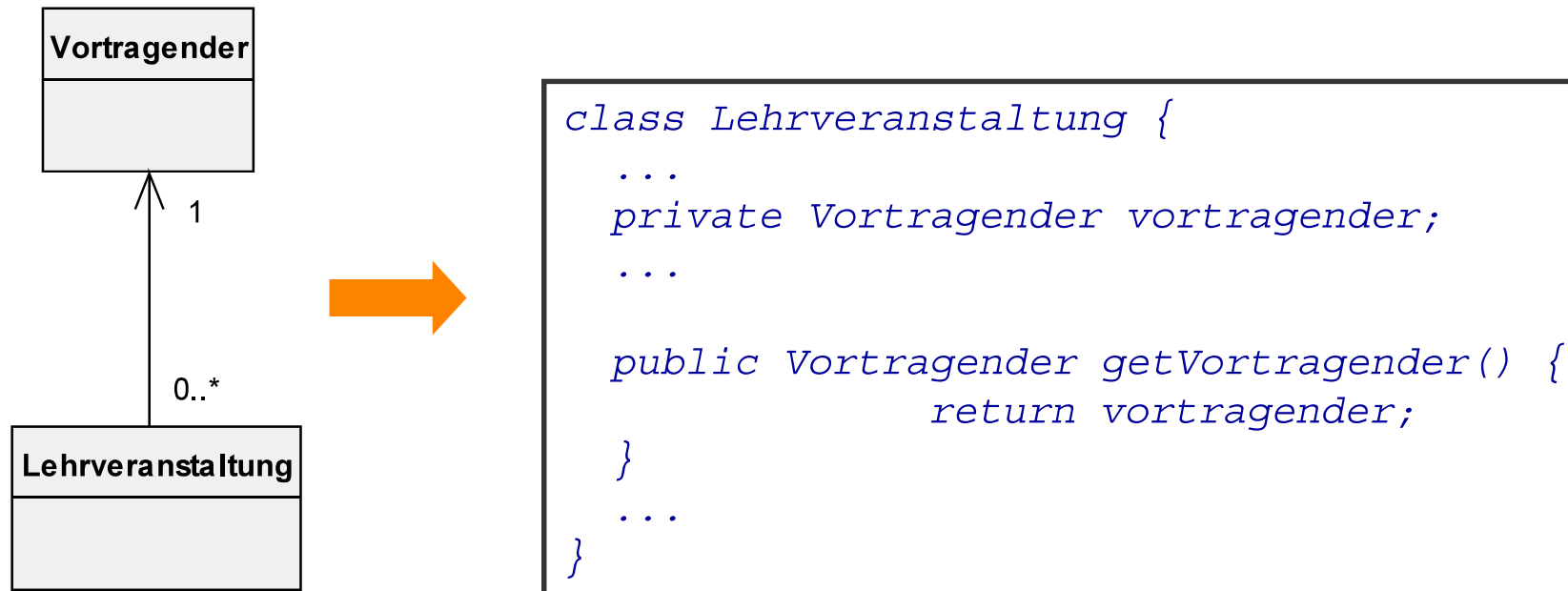


```
class Dekan {
    ...
    private Fakultaet rFakultaet;
    ...
    public Fakultaet getFakultaet() {
        return rFakultaet; }
}
class Fakultaet {
    ...
    private Dekan rDekan;
    ...
    public Dekan getDekan() { return rDekan; }
}
```

- Die entsprechenden Rollen werden als Attribute in die jeweils gegenüberliegende Klasse eingefügt
- Ist die Multiplizität 0..1, kann das entsprechende Attribut unter Umständen auch den Wert `null` haben

Übersetzung nach Java:

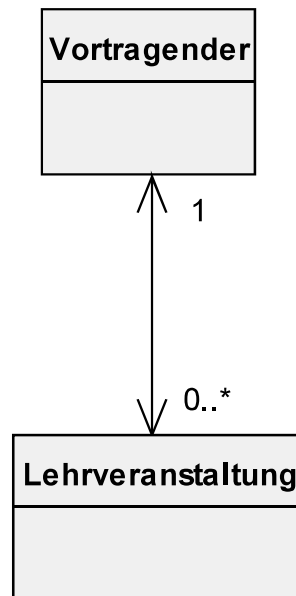
Unidirektionale ?:1-Assoziation



- Keine Änderung an Vortragender!
- Viel leichter zu implementieren als bidirektionale Assoziation

Übersetzung nach Java:

Bidirektionale 1:*-Assoziation



```
java.util.ArrayList;

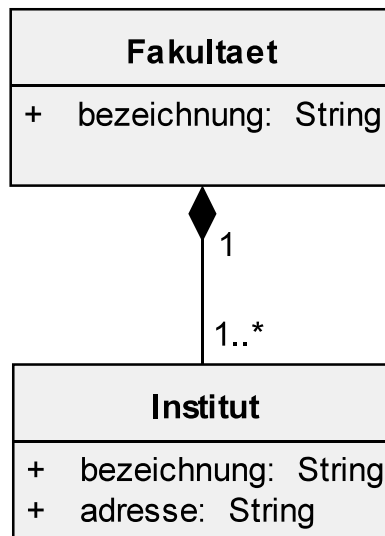
class Vortragender {
    ...
    private ArrayList lvas;
    ...

    public Vortragender {
        lvas = new ArrayList();
    }
    ...
}
```

- Lehrveranstaltung wird wie vorher implementiert
- Wenn `n` fix vorgegeben ist (nicht `*`), dann kann auch ein Array verwendet werden



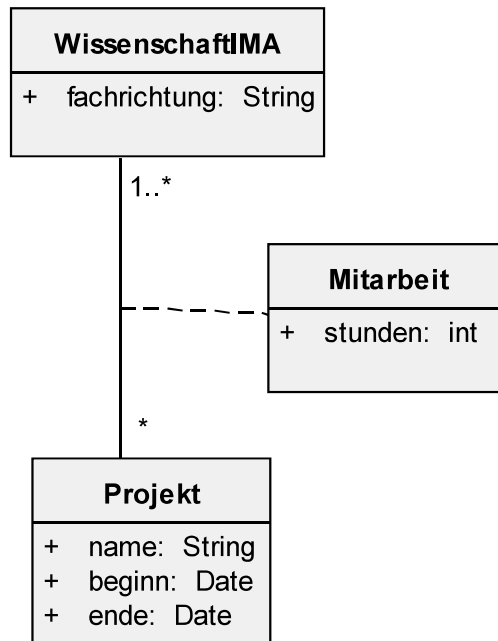
Übersetzung nach Java: Starke Aggregation



```
class Fakultät {
    ...
    private Institut i1, i2, ..., in;
    ...
    public Fakultät () {
        i1 = new Institut();
        i2 = new Institut();
        ...
        in = new Insitut();
    }
    ...
}
```

- Nun müssen die Operationen, die auf den Instituten durchgeführt werden können, durch die Fakultät zur Verfügung gestellt werden

Übersetzung nach Java: Assoziationsklasse



```
import java.util.Hashtable;

class WissenschaftlMA {
    ...
    private Hashtable rProjekt;
        //Schluessel: Projekt
        // Wert: Mitarbeit
    ...
}
```

- Die Assoziation wird mit Hilfe einer Hashtable abgebildet
- Ist die Assoziation nicht gerichtet, muss in der gegenüberliegenden Klasse ebenfalls eine Hashtable eingefügt werden

Übersetzung nach Java:

Zusammenfassung

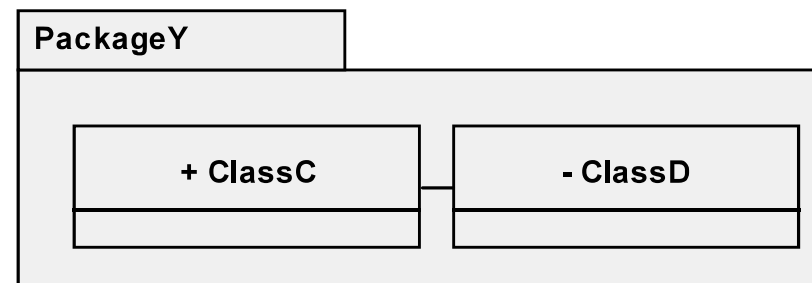
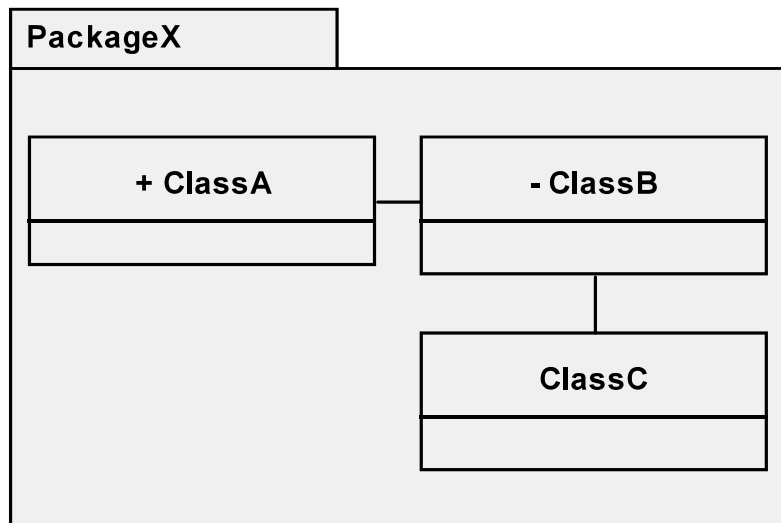
- Klassen werden nach Java-Klassen übersetzt
- Attribute und Operationen werden in Java als Instanzvariablen und Methoden dargestellt
- Klassenvariable und –operationen (unterstrichen im Klassendiagramm) werden mit dem Schlüsselwort `static` versehen
- Assoziationen werden mit Hilfe von Variablen ausgedrückt
 - für 1:1-Beziehungen reicht jeweils eine Variable vom Typ der verbundenen Klasse
 - für 1:n-Beziehungen braucht man Arrays, ArrayLists oder Ähnliches
 - Angabe von Navigationsrichtung (unidirektional) vereinfacht i.A. die Implementierung
 - Operationen zur Verwaltung der Assoziationen müssen eingefügt werden
- Einfachvererbung wird von Java direkt unterstützt (`extends`)
- Assoziationsklassen können mit Hashtables dargestellt werden

Paketdiagramm

- 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

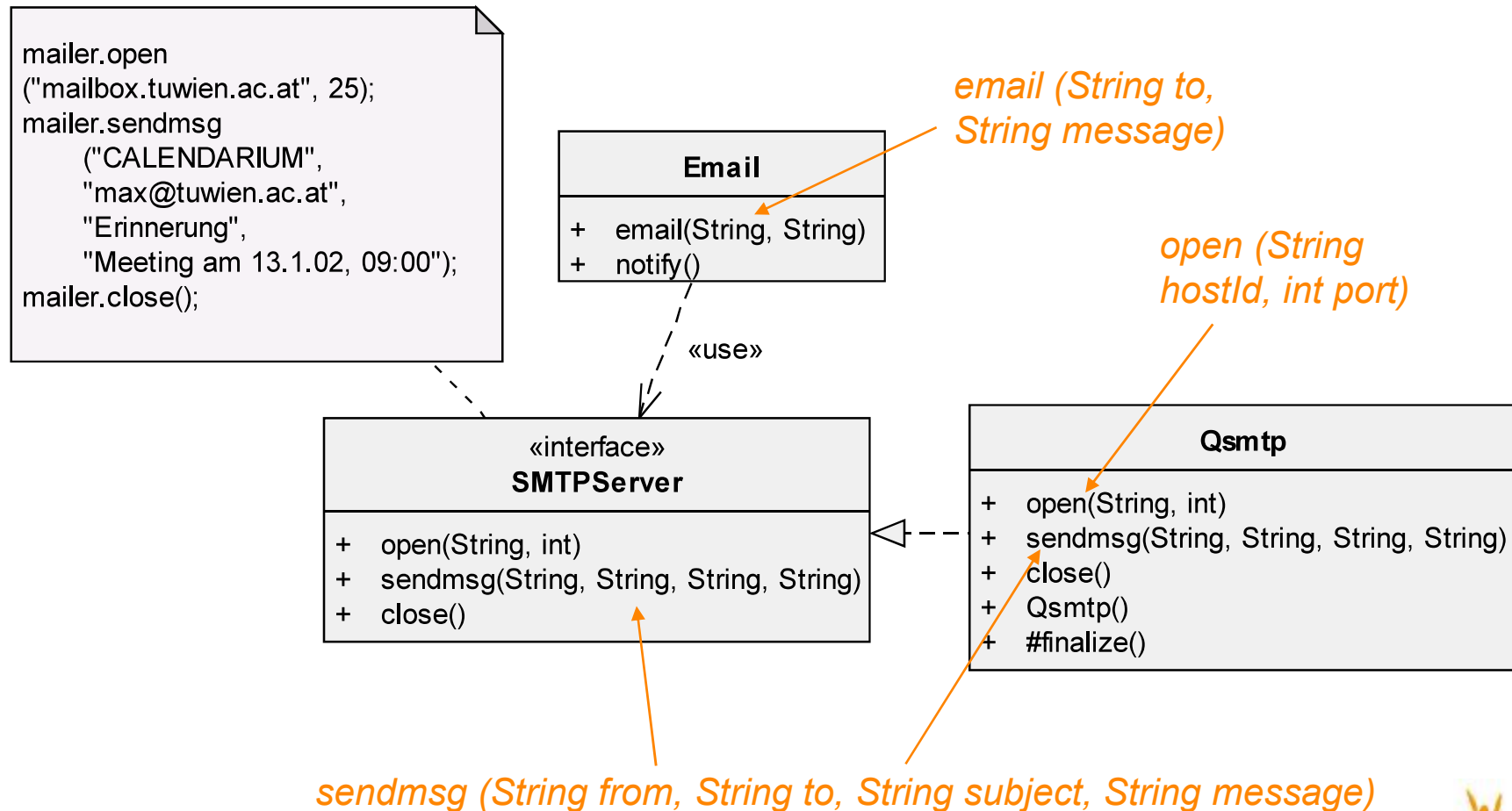


Interface

- Ein Interface spezifiziert **gewünschtes Verhalten** durch **Zusammenfassung der Operationen**
 - einer Klasse
 - einer Komponente
 - eines Paketsund kann auch Attribute aufweisen
- **Unterschied zur abstrakten Klasse**
 - Abstrakte Klasse kann nur durch Subklassen realisiert werden, Interfaces durch beliebige Klassen
- Klassen, die ein **Interface realisieren - Anbieter** - können noch zusätzliche Operationen aufweisen
- Klassen, die ein **Interface benutzen - Klienten** - müssen nicht alle angebotenen Operationen tatsächlich nutzen

Interface: Beispiel CALENDARIUM (1/2)

- Notationsvariante 1:



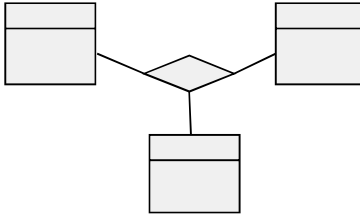
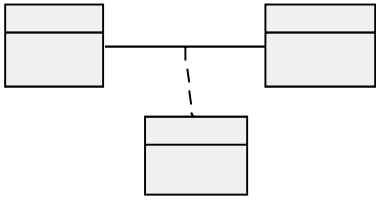
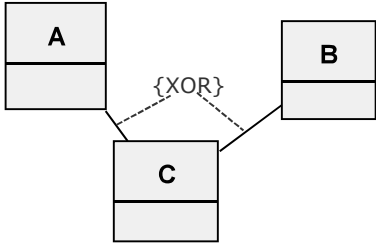
Interface: Vorteile

- Durch die Verwendung von Interfaces wird die **Vererbung von Implementierungen** von der **Vererbung von Interfaces** getrennt
 - Insbesondere **Frameworks** können größtenteils auf der Basis von Interfaces gebaut werden
- Eine **Klasse** kann als **Menge von Rollen** angesehen werden
 - Jedes **Interface** repräsentiert eine **Rolle**, die die Klasse spielt
 - Verschiedene **Klienten** verwenden nur jene Rollen, die für sie interessant sind
 - Mit Interfaces können **Sichten** auf eine Klasse für verschiedene Klienten realisiert werden
 - **Kopplung** wird reduziert, Flexibilität in Bezug auf Wartbarkeit und Erweiterbarkeit steigt




Basiselemente (1/3)

Name	Syntax	Beschreibung
Klasse	<div> <div>Klassenname</div> <div> - Attribut1: Typ - Attribut2: Typ </div> <div> + Operation1() : void + Operation2() : void </div> </div>	Beschreibung der Struktur und des Verhaltens einer Menge von Objekten
abstrakte Klasse	<div> <div>Klassenname</div> <div></div> </div> oder <div> <div>{abstract}</div> <div>Klassenname</div> <div></div> </div>	Klasse, die nicht instanziiert werden kann
Assoziation	<div> <div>_____</div> <div> <-----> </div> <div> X-----> </div> </div>	Beziehung zwischen Klassen: keine Angabe über Nav.-r.; mit Navigationsrichtung; in eine Richtung nicht navigierbar.

Basiselemente (2/3)

Name	Syntax	Beschreibung
n-äre Assoziation		Beziehung zwischen n Klassen
Assoziations- klasse		nähere Beschreibung einer Assoziation
xor-Beziehung		Entweder steht Klasse A oder Klasse B in Beziehung zu C, nicht aber beide

Basiselemente (3/3)

Name	Syntax	Beschreibung
schwache Aggregation		"Teil-Ganzes"-Beziehung
starke Aggregation = Komposition		exklusive "Teil-Ganzes"-Beziehung
Generalisierung		Vererbungsbeziehung zwischen Klassen

Zusammenfassung

- Sie haben diese Lektion verstanden, wenn Sie wissen ...
- was der Unterschied zwischen einer Klasse und einem Objekt ist.
- was der Unterschied zwischen abstrakten und konkreten Klassen ist.
- was Attribute und Operationen einer Klasse sind und welche Eigenschaften diese haben können.
- dass Klassen durch Assoziationen miteinander verbunden werden.
- warum Assoziationen mit einer Multiplizität versehen werden.
- was Generalisierung ist und wann diese eingesetzt wird.
- was der Unterschied zwischen starker und schwacher Aggregation ist.
- wie Sie aus einer textuellen Angabe ein UML-Klassendiagramm erstellen.
- wie Sie die wichtigsten Elemente des Klassendiagramms in Java umsetzen können.
- wozu ein Meta-Modell benötigt wird.
- wie der Zusammenhang zwischen Klassen- und Objektdiagramm ist.
- was Interfaces sind.
- wozu Pakete eingesetzt werden.
- wie Abhängigkeiten in UML dargestellt werden können.