


## Code-Erzeugung aus UML-Klassendiagrammen

Theorie und Praxis



*REConf 2009  
München*



**Dominik Gessenharter** | Dipl.-Inf.  
Institut für Programmiermethodik  
und Compilerbau

James-Franck-Ring | 89081 Ulm | Germany  
Tel.: +49 731 50-24252 | Fax.: +49 731 50-1224252  
dominik.gessenharter@uni-ulm.de  
<http://www.uni-ulm.de/in/pm/mitarbeiter/gessenharter.html>



## Code-Generierung - wozu?

Abstraktion, Produktivität, Qualität, Wiederverwendbarkeit  
(Forschungsgegenstand an der Universität Ulm)  
Traceability von Anforderungen bis zum Code

## Warum dieser Vortrag?

Code-Generierung ist  
theoretisch eine feine Sache,  
in der Praxis oft nicht befriedigend

## Was kann man tun?

## Code-Generierung und RE

Anforderungen bilden die Grundlage von Modellen

Modelle bilden die Grundlage für Code

Traceability von Anforderungen bis zum Code wünschenswert  
(vollständige Generierung aus UML Modellen kaum machbar)

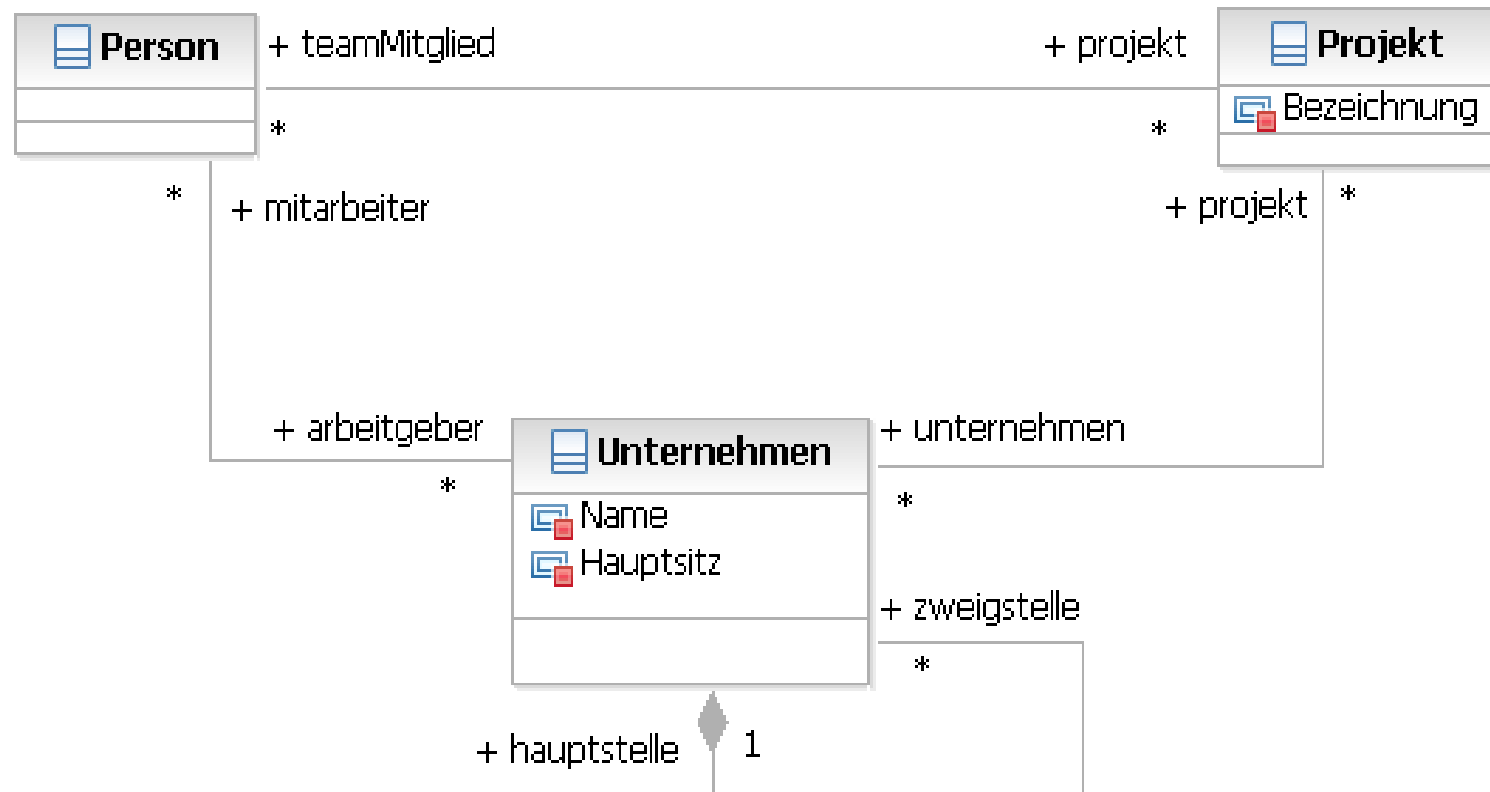
Bei Traceability von Anforderungen zum Modell kann Generator diese bis zum Code erweitern

Bei korrektem Generator entspricht der Code den Anforderungen in gleicher Weise wie das Modell

## Inhalt

- Grundlagen der UML Klassendiagramme
- Objektorientierte Programmierung
- Exkurs: Relationale Datenbanken
- Modellierung in der Praxis
- Code-Generierung in der Praxis
- Theorie: Was ist möglich?
- Zusammenfassung und Ausblick
- Diskussion

## Grundlagen der UML Klassendiagramme



## Grundlage der UML Klassendiagramme

- Klassen
  - Attribute
  - Operationen
  - Verhalten
- Assoziationen
  - Sichtbarkeit
  - Navigierbarkeit
  - Multiplizitäten
  - Stelligkeit
- Besondere Assoziationen
  - Aggregation und Komposition
  - Reflexive Assoziationen → Rollennamen
- Klassen und Assoziationen sind instantiierbar (Objekte und Links)

## Objektorientierte Programmierung

- Klassen
  - Attribute
  - Methoden
- Referenzen
  - Spezialfall eines Attributs
- Kein Konstrukt zur Darstellung einer Assoziation
- Klassen sind instantiierbar, Referenzen sind Attribute in Klassen

## Exkurs: Relationale Datenbanken

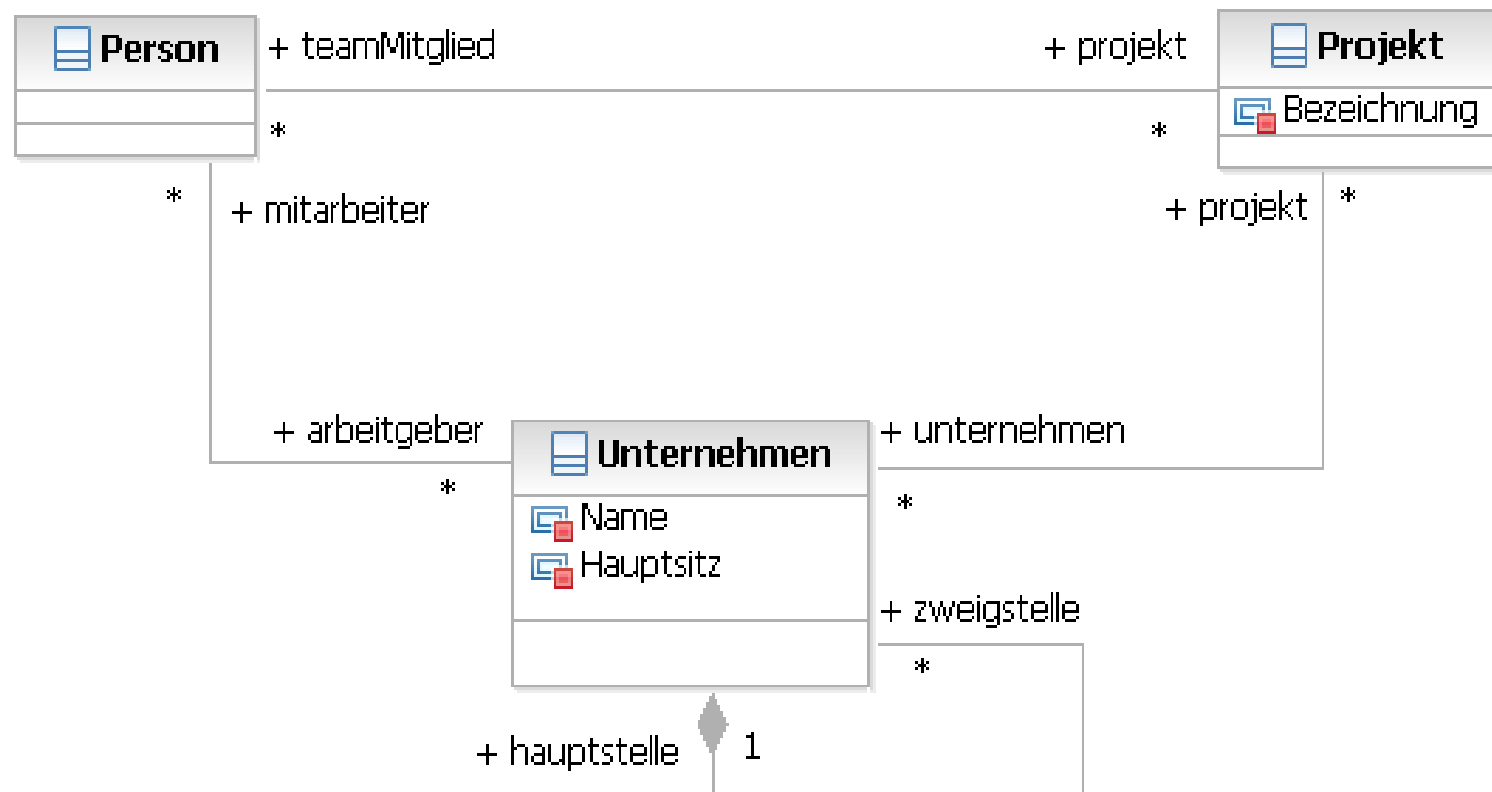
- Entitäten
  - Attribute
- Relationen
  - Multiplizitäten
  - Stelligkeit
- Datenbanksysteme erlauben die Definition der Integrität der Daten im Datenmodell
- Applikationen können keine inkonsistenten Daten speichern
- Datenintegrität wird an **einer** Stelle definiert

*Es herrscht Ordnung in der Datenbank*

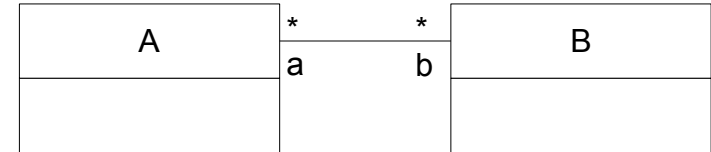
## Ordnung im Arbeitsspeicher

- Konzepte der Datenbanken als Vorbild
- Fehler nicht beim Versuch, zu speichern, sondern beim Versuch, inkonsistenten Zustand im Speicher herzustellen
- Einhaltung aller im Modell formulierten Bedingungen durch Code erzwingen
- Bei Verstößen Fehler signalisieren
- Entwickler werden gezwungen, auf mögliche Fehler zu reagieren

## Modellierung in der Praxis



## Assoziationen mittels Referenzen

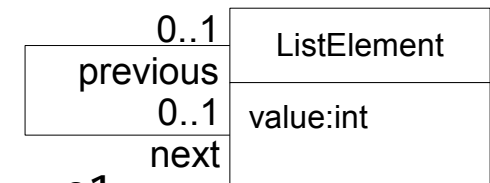


- A referenziert B
  - `public B b;`
- B Referenziert A
  - `public A a;`
- Allgemein:  
für `o:A`  
 $(o.b.size > 0) \Leftrightarrow o.b[i].a.contains(o) == true$   
 mit  $i \leq b.size - 1$

- Bei Multiplizitäten von jeweils 0..1 gilt:

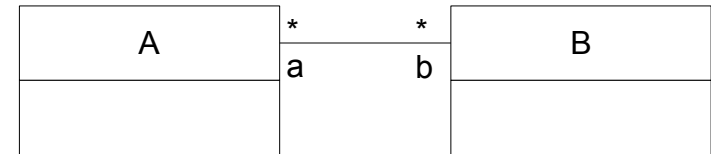
für `e1:ListElement`

$(e1.next \neq null) \Leftrightarrow e1.next.previous == e1$



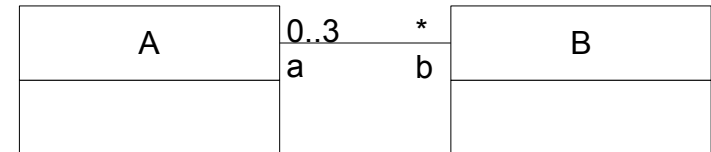
- Assoziationen sind symmetrisch

## Assoziationen mittels Referenzen



- Symmetrie meistens nicht Bestandteil des generierten Codes
- Implementierung durch wechselseitige Aufrufe möglich
- Problem:
  - Sichtbarkeit → ggf. kein Aufruf auf Gegenseite möglich
- Spezialfall:
  - Bei Navigierbarkeit in nur einer Richtung keine Rückreferenz nötig
  - Aber: Multiplizitäten beachten
- Referenzen nur für einige Spezialfälle geeignet
- Referenzen nicht geeignet für mehrstellige Assoziationen

## Assoziationen mit Multiplizitäten



- Obergrenzen als Länge eines Arrays
- Untergrenzen ohne Prüfung
- Die meisten Werkzeuge unterstützen Multiplizitäten nicht

## Mehrstellige Assoziationen

- Nicht, oder nur unzureichend unterstützt.

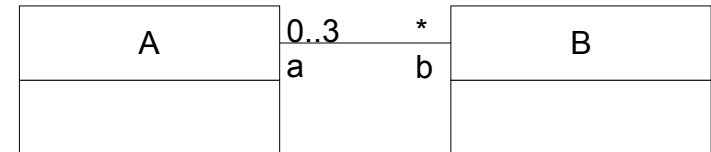
## Assoziationen als Komposition

- Nicht unterstützt
- Bei Java nicht ohne Weiteres möglich

## Was ist möglich?

- Multiplizitäten sind implementierbar
  - Auch bei Implementierung mittels Referenzen
- n-stellige Assoziationen sind implementierbar
- Kompositionen sind implementierbar

## Assoziationen mit Multiplizitäten



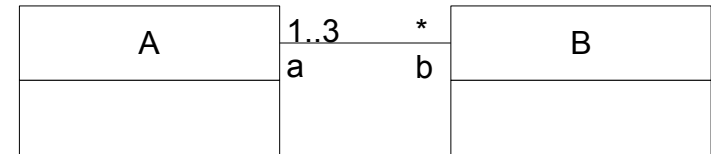
- Obergrenzen leicht prüfbar  
Beim Erstellen neuer Links

```
public class B{

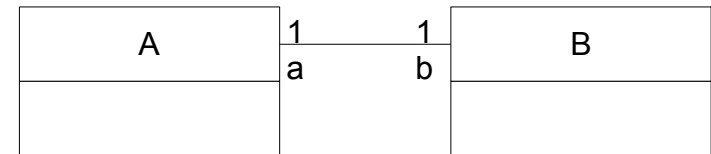
    private List<A> a;

    public void addA(A a) throws Exception{
        if (this.a.size() > 2)
            throw new Exception();
        else
            this.a.add(a);
    }
}
```

## Assoziationen mit Multiplizitäten

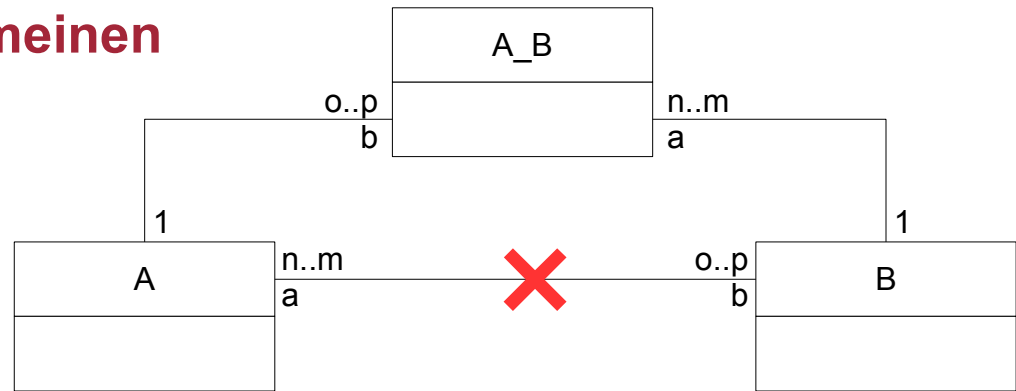


- Untergrenzen leicht prüfbar
  - Beim Aufheben bestehender Links



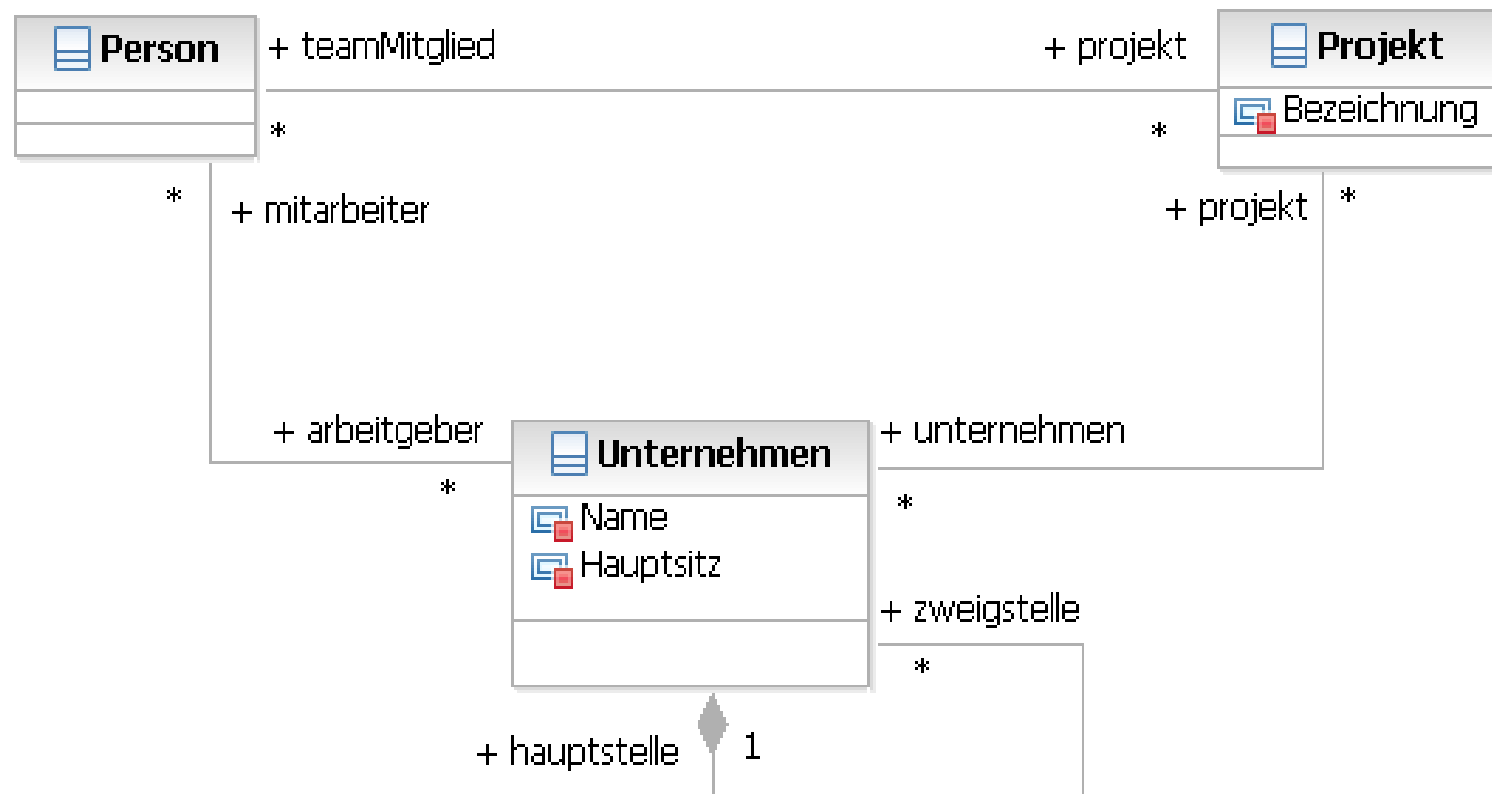
- Bei der Erzeugung
  - Parameter  $p_1, \dots, p_n$  für Konstruktor von A
  - Parameter  $q_1, \dots, q_n$  für Konstruktor von B
  - Konstruktoren für A:
    - $A(p_1, \dots, p_n, q_1, \dots, q_n) \{ \dots; b = \text{new } B(q_1, \dots, q_n, \text{this}); \}$
    - $A(p_1, \dots, p_n, B b)$
  - Konstruktoren für B:
    - $B(q_1, \dots, q_n, p_1, \dots, p_n) \{ \dots; a = \text{new } A(p_1, \dots, p_n, \text{this}); \}$
    - $B(q_1, \dots, q_n, A a)$
- Verwendung von Factories ggf. sinnvoller

## Assoziationen im Allgemeinen

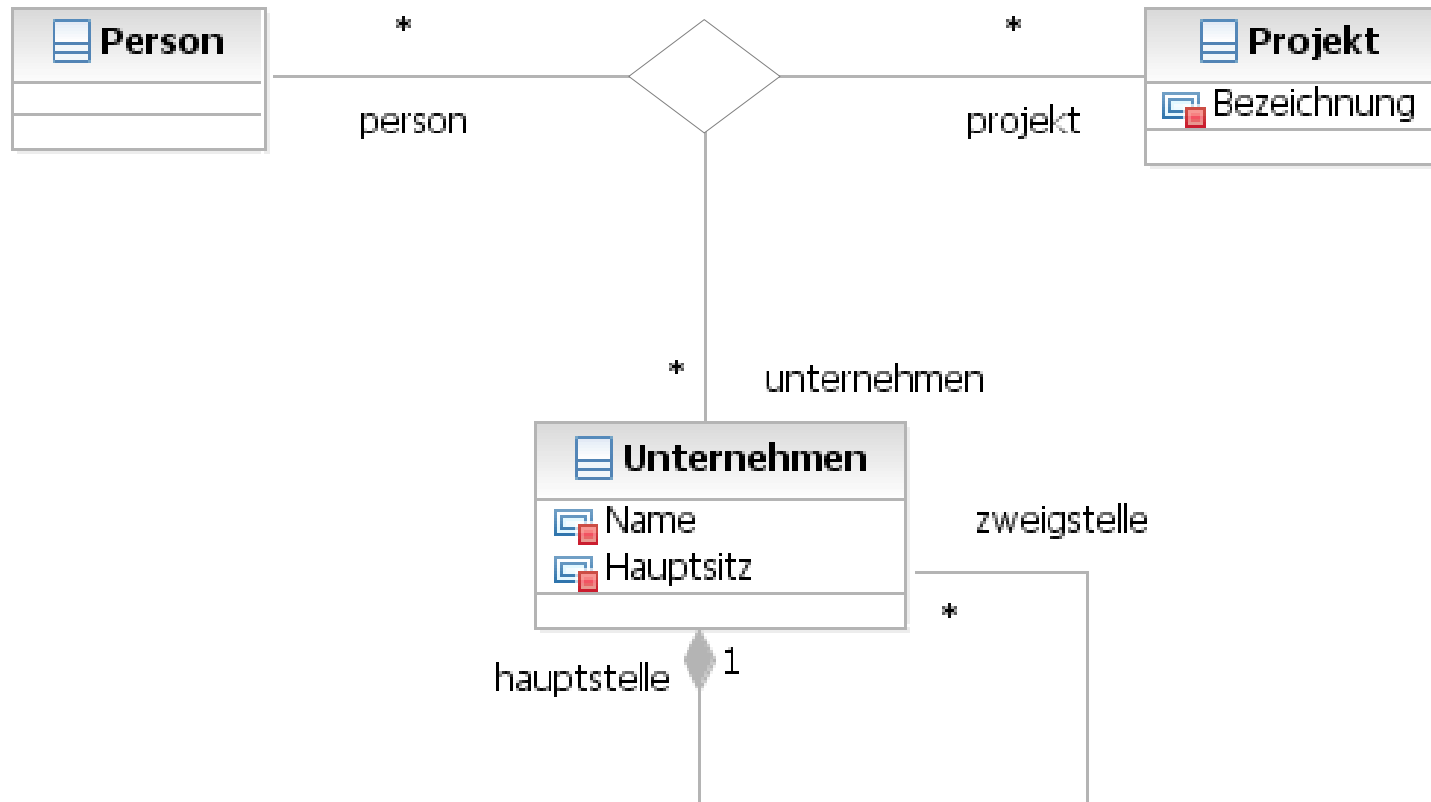


- Implementierung der Assoziation als eigene Klasse
- Sichtbarkeit definiert die Möglichkeit des Zugriffs auf Instanzen der Implementierungsklasse (Links)
- Link-Objekt ermöglicht Zugriff nur auf Instanzen an navigierbaren Assoziationsenden
- Stelligkeit, Navigierbarkeit und Sichtbarkeit frei kombinierbar
- Entscheidend ist die Konsistenz der Referenzen auf Link-Objekte und Instanzen der an der Assoziation beteiligten Klassen

## Modellierung mit binären Assoziationen



## Exaktere Modellierung



## Mehrstellige Assoziationen

- Link-Objekte können auf beliebig viele Instanzen verweisen
- Die Referenzen auf alle beteiligten Instanzen werden zentral in der Link-Instanz gehalten
- Kein Koordinationsaufwand
- Aufwand zur Erzeugung der Link-Objekte





## Zusammenfassung

- Allgemein benötigen Assoziationen zur Laufzeit Instanzen
- In OO-Sprachen derzeit nur durch Klassen realisierbar
- Assoziationsklassen damit einfach realisierbar
- Vererbung bei Assoziationen realisierbar
- Auch bei Sprachen mit Garbage-Collection sind Kompositionen implementierbar
- Aufwand zur Erzeugung von Link-Objekten
- Symmetrie der Assoziation in Link-Objekten realisiert
  - Keine wechselseitigen Aufrufe
- Freie Kombinierbarkeit der Konzepte der Assoziation

## Ausblick

- Unsere Arbeit
  - Implementierung eines Code-Generators mit voller Unterstützung von Assoziationen der UML 2.2
  - Messung des Potenzials der Code-Erzeugung
    - Produktivität
    - Qualität
- Modellierung und Code-Erzeugung
  - Vermehrter Einsatz schwierig zu implementierender Konzepte
    - Mehrstellige Assoziationen
    - Vererbung bei Assoziationen
    - Abstrakte Assoziationen
    - Assoziationen zwischen Assoziationen
  - Bessere Modelle
  - Bessere Modellierer
  - Echte Fortschritte im Bereich MDD...

## Diskussion

Vielen Dank für

Ihre Aufmerksamkeit

Ihr Feedback

Ihre Fragen