

1.4 Statisches Modell

1.4.1 Assoziation

Eine Assoziation beschreibt eine Sammlung von Verknüpfungen mit einer gemeinsamen Struktur und Semantik.

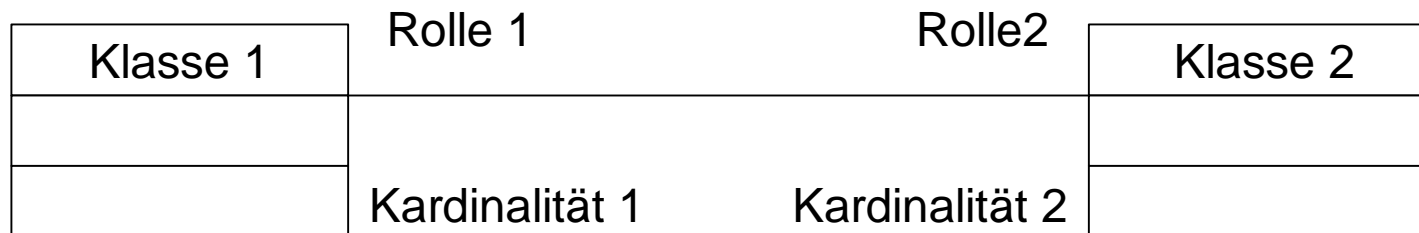
- Eine Assoziation zwischen zwei Klassen A und B drückt eine beliebig geartete Beziehung zwischen diesen Klassen aus.
- Dies kann bedeuten:
 - A und B kennen sich
 - A und B sind miteinander verbunden
 - Zu jedem A gibt es ein B
 - ...
- z.B.:



■ Kardinalität

- Gibt an wie viele Objekte der Klassen in der späteren Verknüpfung einbezogen sind.
- Es können **kein** Objekt, **ein** Objekt und **viele** Objekte in der Verknüpfung auftreten

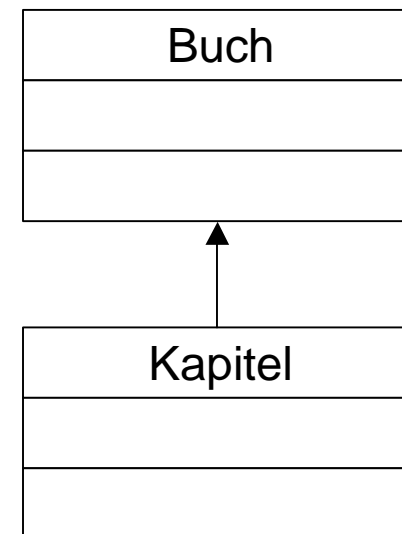
Assoziation



1.4.2 Aggregation

Eine Aggregation bezeichnet eine „Teil-Ganzes-“ oder „ist-Teil-von“-Verknüpfung

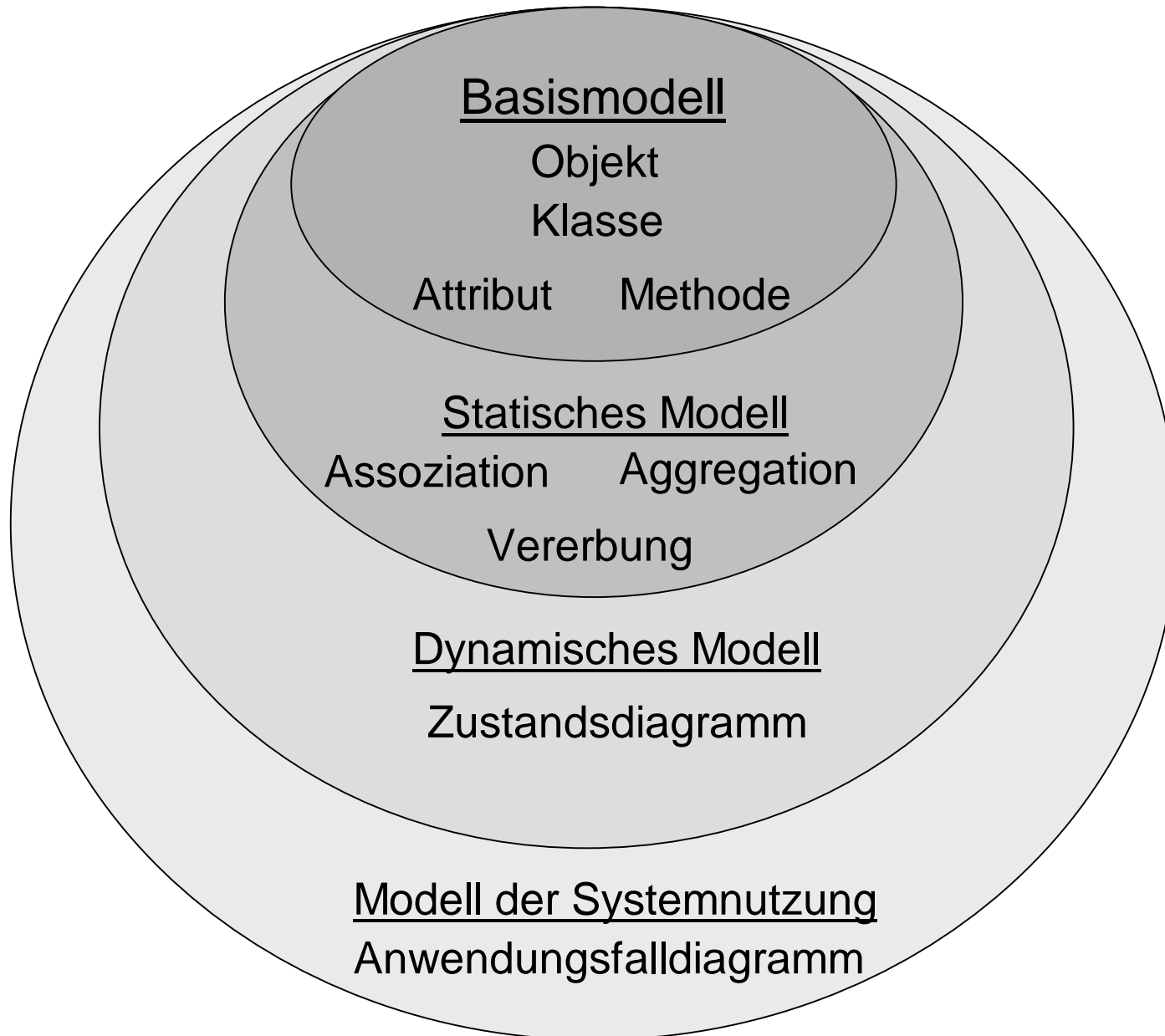
- Eine Aggregation ist eine spezielle Assoziation; sie drückt eine Beziehung zwischen einem Ganzen und seinen Teilen aus.
- Dies sind Beziehungen der Art:
 - A besteht aus B
 - A enthält B
 - B ist Teil von A
 - ...
- Eine Aggregation mit sehr starker Bindung wird als Komposition bezeichnet



1.4.3 Dynamisches Modell

- Für die Beschreibung der Dynamik eines objektorientierten Systems gibt es verschiedene Ansätze
 - z.B.: durch die Angabe von Automaten
 - Beschreibt welche Zustände angenommen werden können
 - Beschreibt die Übergänge von Zuständen
- Beispiel: Buchungssystem im Internet
 - Endliche Menge von Zuständen
 - Buchung erstellt
 - Buchung ausgefüllt
 - Buchung reserviert
 - Buchung storniert
 - ...

1.4.4 Objektorientiertes Modell



1.5 Wofür Objektorientierung?

1.5.1 Qualität von Software

- **Korrektheit:** Die Fähigkeit von Softwareprodukten, ihre Aufgaben genau so auszuführen, wie sie in der Spezifikation angegeben wurden.
- **Robustheit:** Die Fähigkeit eines Softwaresystems, angemessen auf außergewöhnliche Bedingungen zu reagieren.
- **Erweiterbarkeit:** Die Leichtigkeit, ein Softwareprodukt an Änderungen der Spezifikation anzupassen.
- **Wiederverwendbarkeit:** Die Fähigkeit eines Softwareelements, bei der Konstruktion von vielen verschiedenen Anwendungen verwendet werden zu können.
- **Kompatibilität:** Die Einfachheit, mit der Softwareelemente miteinander kombiniert werden können.

- **Effizienz:** Die Fähigkeit eines Softwaresystems, möglichst wenig Anforderungen an Hardwareressourcen, wie Rechenleistung, Speicherplatzbedarf und Bandbreite, zu stellen.
- **Portabilität:** Die Leichtigkeit, mit der Softwareprodukte an unterschiedliche Hard- und Softwareumgebungen angepaßt werden kann.
- **Benutzbarkeit („ease of use“):** Die Einfachheit, mit der Menschen unterschiedlicher Qualifikation und mit unterschiedlichem Hintergrund lernen können, Softwareprodukte zu benutzen und sie zur Problemlösung einzusetzen. Dies schließt Einfachheit bei der Installation, beim Betrieb und bei der Überwachung mit ein. Ein wichtiges Element der Benutzbarkeit stellt eine **ergonomische Benutzungsoberfläche** dar.

1.6 Der Softwarelebenszyklus

1.6.1 Phasen der Softwareentwicklung

- **Analysieren**
 - Analyse des Basisprozesses
 - Ergebnis: Aufgabenstellung zur Softwareentwicklung
- **Spezifizieren**
 - Dokumentation der Funktionen des Softwareproduktes
 - Ergebnis: funktionelle Spezifikation
- **Entwerfen**
 - Dokumentation der Problemlösung
 - Ergebnis: logische Gliederung der Funktionen und Daten und Ablaufstruktur
- **Implementieren**
 - Codierung der Problemlösung in einer Programmiersprache
 - Ergebnis: lauffähiges Softwareprodukt

- **Testen**

- Verwenden von Testmethoden – Fehlerfindung
- Ergebnis: Fehlerprotokoll

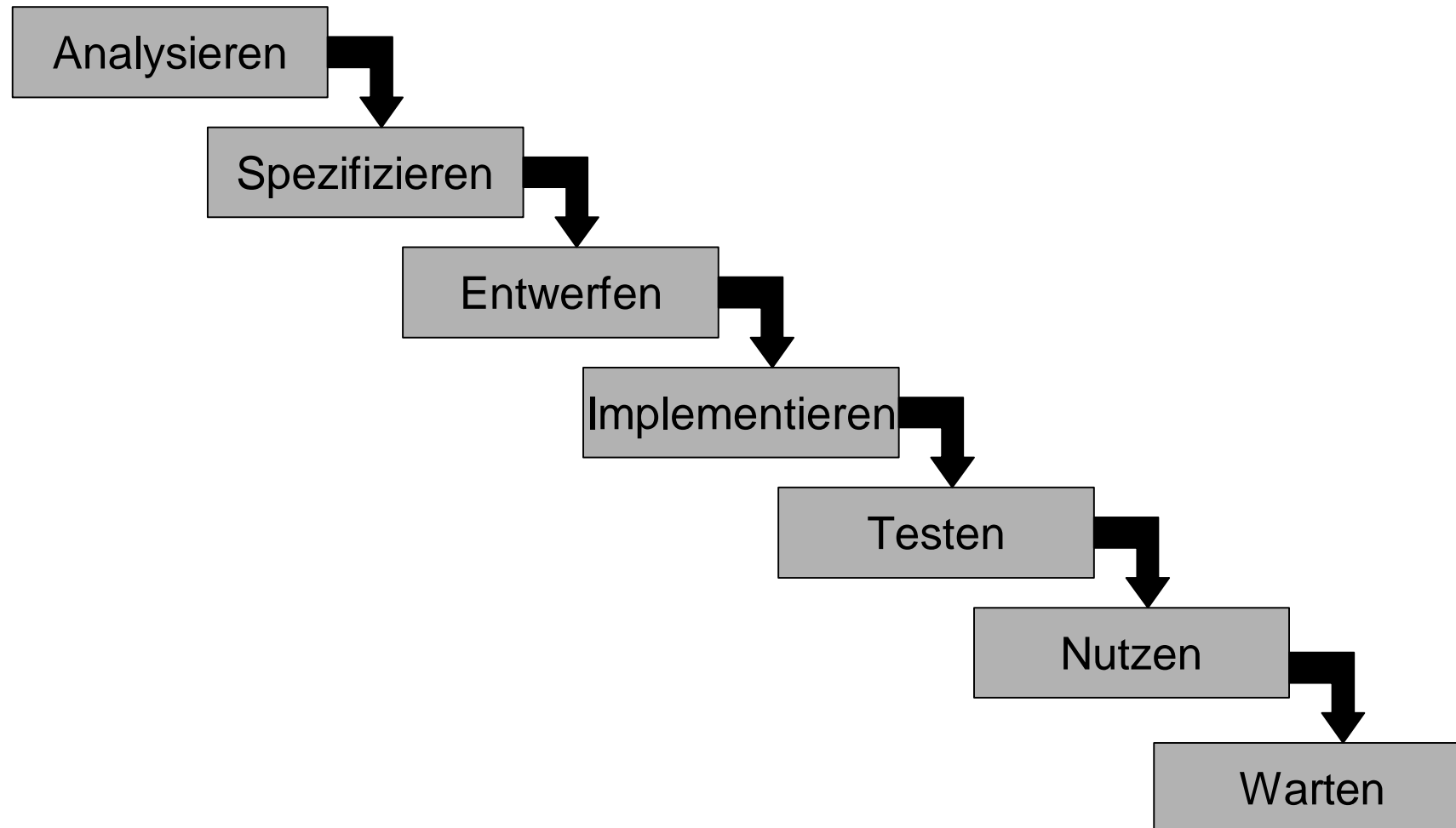
- **Nutzen**

- Anwendung des Softwareprodukts
- Ergebnis: laufendes Softwareprodukt

- **Warten**

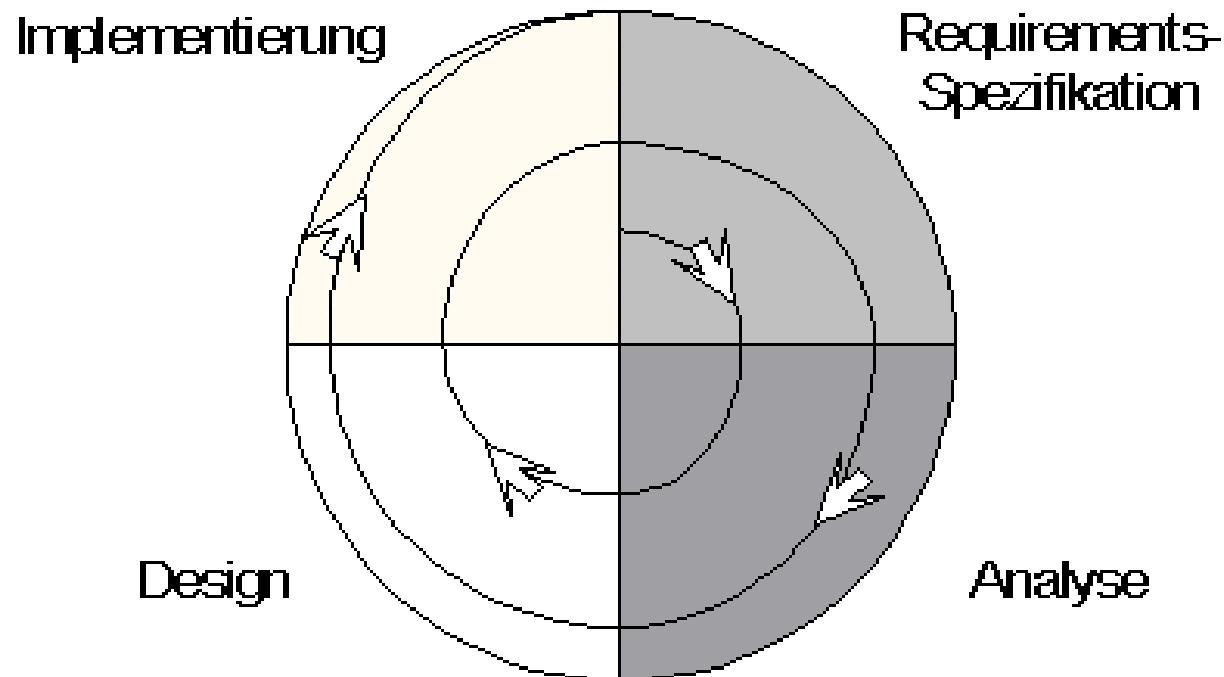
- Ändern und Anpassen des fertigen Softwareprodukts
- Ergebnis: aktualisiertes Softwareprodukt

1.6.2 Klassisches Wasserfallmodell



1.6.3 Spiralmodell nach Boehm

- Iteratives und inkrementelles Vorgehen
- Phasen können mehrfach durchlaufen werden



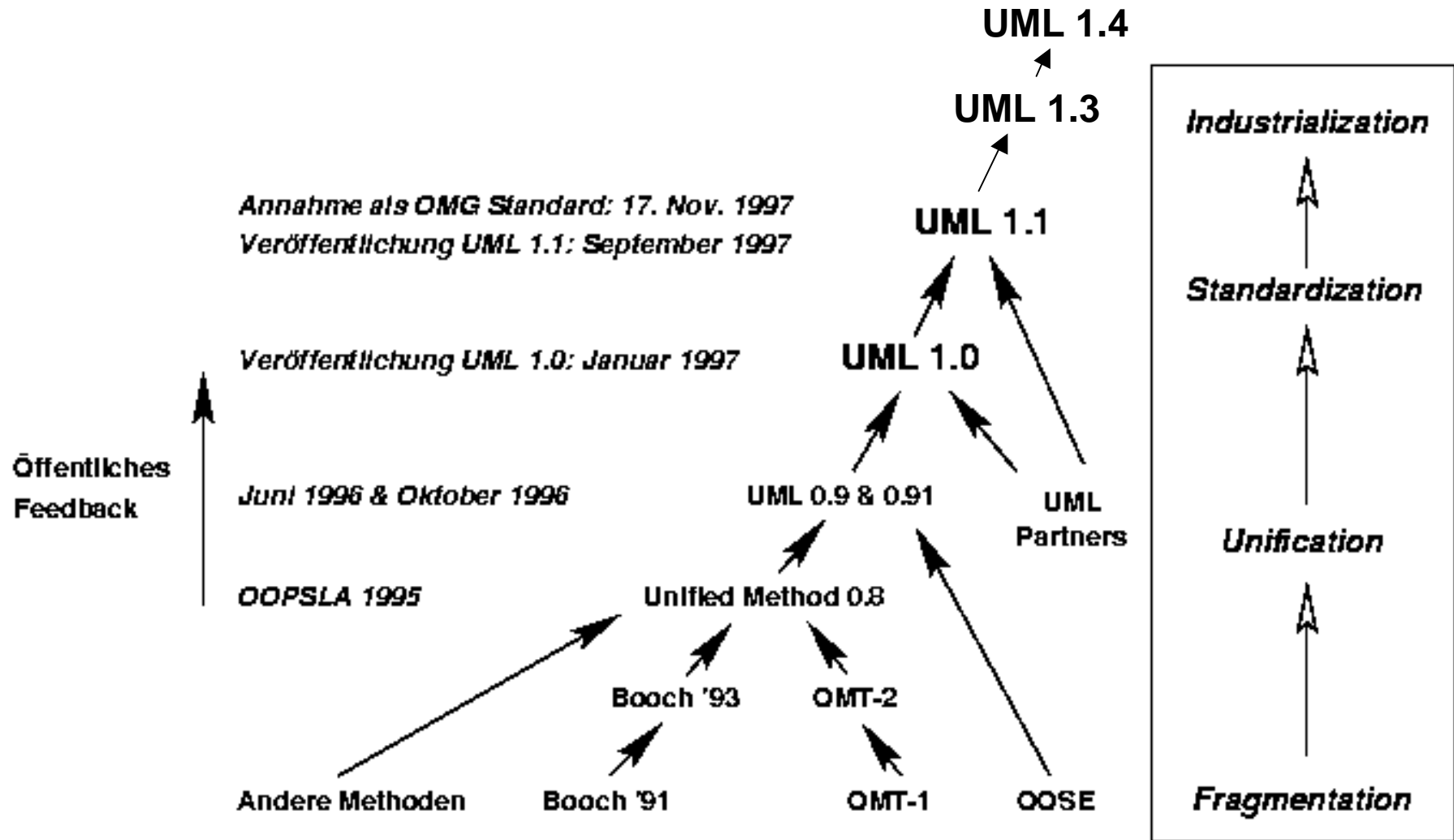
2 UML – Unified Modeling Language

2.1 Geschichte der UML

Wichtige Vorarbeiten für OO-Methoden, waren:

- Shlaer/Mellor,
- Coad/Yourdon,
- Booch,
- OMT (von J. Rumbaugh et al.) und
- OOSE (von I. Jacobson).

- 1994 war für die UML - und wohl auch für die Geschichte der OO-Methoden - ein bedeutsames Jahr. Jim Rumbaugh verließ General Electric und ging zu Rational, wo bereits Grady Booch tätig war. Die beiden wollten ihre Methoden zusammenführen und erklärten, daß der Methodenkrieg nun vorbei sei.
- 1995 gaben sie die erste Veröffentlichung ihrer "Unified Method 0.8" heraus.
- Viel wichtiger in diesem Jahr war jedoch die Bekanntgabe, daß Rational die Firma Objectory gekauft hatte, und daß nun auch Ivar Jacobson für Rational arbeitete. Im Volksmund werden die drei "Amigos" genannt.
- 1996 begannen die Arbeiten an der UML
- Im Jahr 1997 wurde die Version 1.0 bei der *Object Management Group (OMG)* als Standardisierungsvorschlag eingereicht, und eine Beschreibung der Version 1.0 wurde veröffentlicht.
- Im September 1997 wurde die Version 1.1 bei der OMG eingereicht, alle anderen Gegenvorschläge wurden zurückgezogen - in der Version 1.1 wurden Konzepte der Gegenvorschläge berücksichtigt.



2.2 UML und Prozeßmodelle:

- UML beinhaltet
 - nur Notation
 - kein Prozeßmodell (z.B.: RUP)
- es wurden aber die Erfordernisse mehrerer Prozeßmodelle berücksichtigt, so daß UML bei deren Anwendung relativ problemlos benutzt werden kann
- UML umfaßt
 - neun Diagrammarten (!)
 - die aber nicht bei jedem Prozeßmodell und
 - noch weniger bei jedem Prozeß verwendet werden (müssen)

2. 3 Klassendiagramm

Das Klassendiagramm zeigt:

- Die statische Struktur des Modells
- Die Beziehungen der Klassen untereinander
 - Generalisierung/Spezialisierung (Vererbung)
 - Assoziation
 - Aggregation
 - Abhängigkeit
 - Verfeinerung
- Die Vererbungsstruktur

2.3.1 Darstellung der Klasse

- Eine Klasse wird durch ein Rechteck mit drei Feldern dargestellt.
 - das obere Feld enthält den Klassennamen
 - das mittlere Feld enthält die Liste der Attribute mit den zugehörigen Datentypen (Klassen)
 - das untere Feld enthält die Liste der Operationen (mit Argumentliste und Typ des Rückgabewertes)
- Beispiel Bankkonto

Bankkonto
+Kontoinhaber: string -Kontostand: integer -Mindestbetrag: float #Zinssatz: float
+Einzahlen (Betrag) +Abheben (Betrag)

Symbole zur Charakterisierung der Sichtbarkeit von Attributen und Operationen

- **+**
 - Public (öffentlich):
 - Der Zugriff ist nicht eingeschränkt
- **-**
 - Private (privat):
 - Nur Methoden innerhalb der Klasse haben Zugriff
- **#**
 - Protected (geschützt):
 - Methoden können zugreifen, die von Klassen, die direkt oder indirekt von der Klasse erben, in der das Attribut definiert wurde

Klassendiagramm allgemein

«stereotyp» Name
+öffentlichesAttribut : Typ #geschütztesAttribut : Typ -privatesAttribut : Typ
+öffentlicheOperation() : Typ +öffentlicheOperationOhneRückgabewert() +öffentlicheOperationMitArg(einParameter : Typ) : Typ #geschützteOperation() : Typ -privateOperation() : Typ

Stereotyp und Merkmale

- Stereotyp
 - Ein Stereotyp klassifiziert Elemente (z.B.: Klassen, Operationen) des Modells.
 - Die UML enthält vordefinierte Stereotypen
z.B:
 <<utility>>
 <<metaclass>>
 <<type>>
 <<boundary>>
 <<enumeration>>
 - Eigene Stereotypen können definiert werden
- Merkmale
 - Ein Merkmal (property) beschreibt Eigenschaften eines bestimmten Elements des Modells
 - Können in einer Liste zusammengefasst werden
 - Form {Schlüsselwert = Wert, ...}

Beispiel:

<<Stammdaten>>

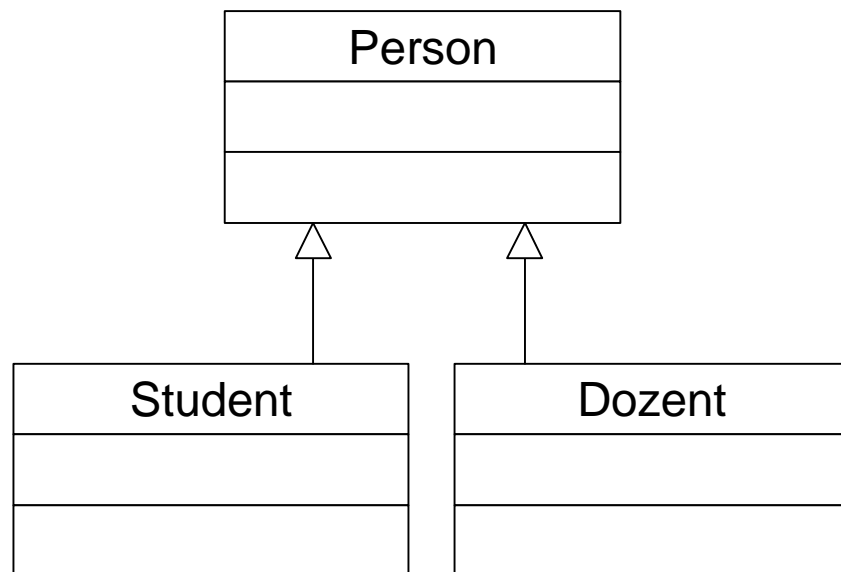
Bankkonto

{Autor=Balzert,
Version=1.0}

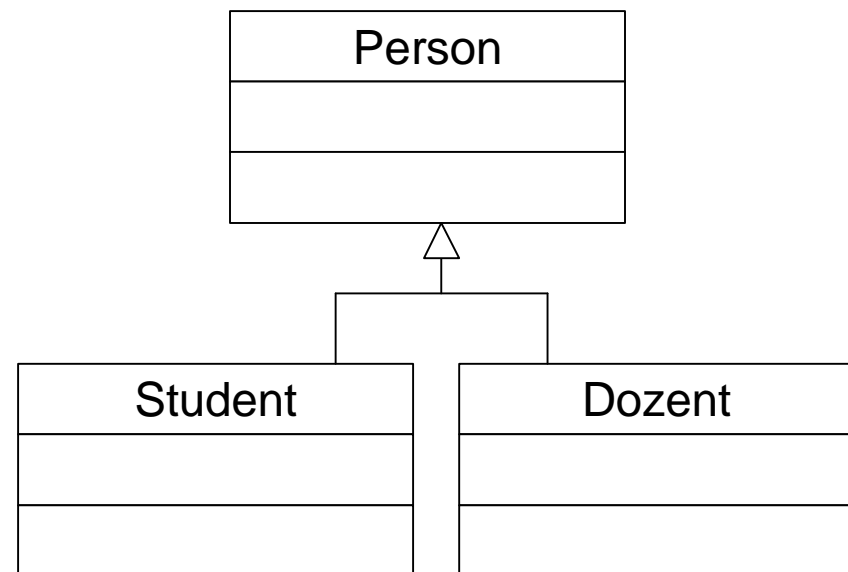
+Kontoinhaber: string
-Kontostand: integer
-Mindestbetrag: float
#Zinssatz: float

+Einzahlen (Betrag)
+Abheben (Betrag)

2.3.2 Darstellung von Einfachvererbung (Single inheritance)



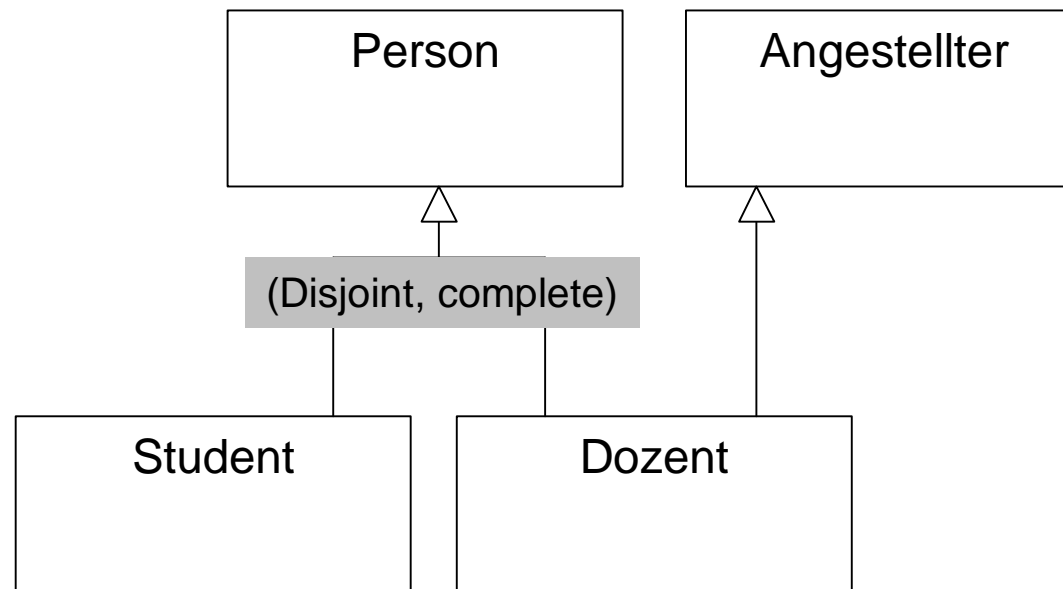
Einfachvererbung



Einfachvererbung (shared target notation)

- Nicht ausgefüllter Pfeil zeigt in Richtung der Superclass
- Details, wie Attribute und Methoden müssen nicht angegeben werden

2.3.3 Darstellung von Mehrfachvererbung (Multiple Inheritance)



Mehrfachvererbung

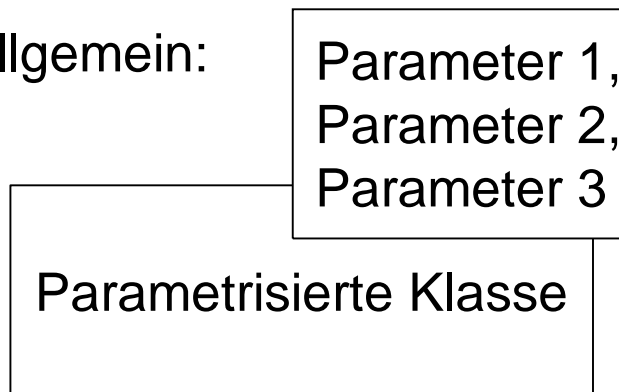
2.3.4 Charakteristiken von Vererbungsbeziehungen

- Vollständigkeit
 - Complete (vollständig)
Alle Unterklassen sind spezifiziert, unabhängig davon, ob Sie im konkreten Diagramm alle angezeigt werden.
 - Incomplete (unvollständig)
Die Liste der Unterklassen im Modell ist unvollständig.
- Zusammenhang zwischen Vererbungsbeziehungen
 - Disjoint (wird standardmäßig angenommen)
Eine weitere Unterklasse kann nur Nachfolger **einer** der modellierten Unterklassen sein.
 - Overlapping
Eine weitere Unterklasse kann Nachfolger **mehrerer** der modellierten Unterklassen sein.

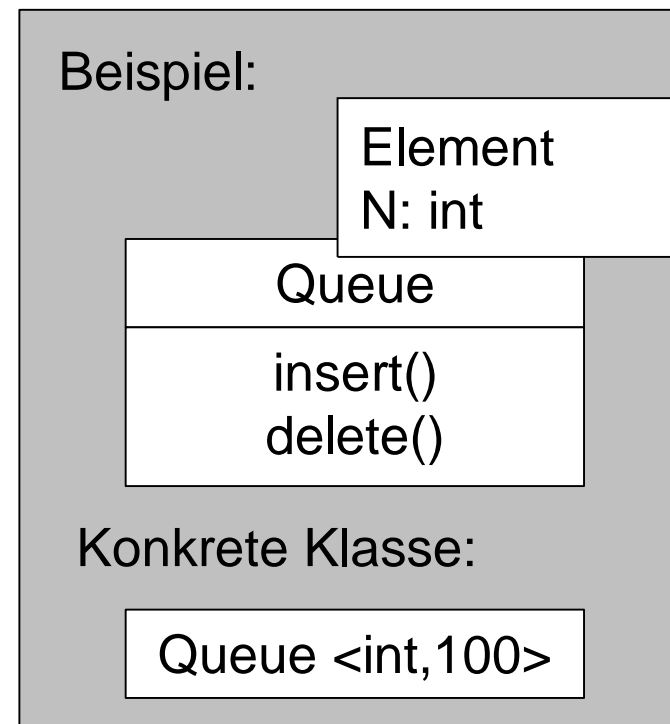
2.3.5 Generische Klassen (Templates)

- UML ermöglicht die Darstellung von generischen Klassen (Templates)
- Templates sind „Klassen von Klassen“
- Müssen vor Ihrer Benutzung mit einem Parameter versehen werden

Allgemein:



Beispiel:

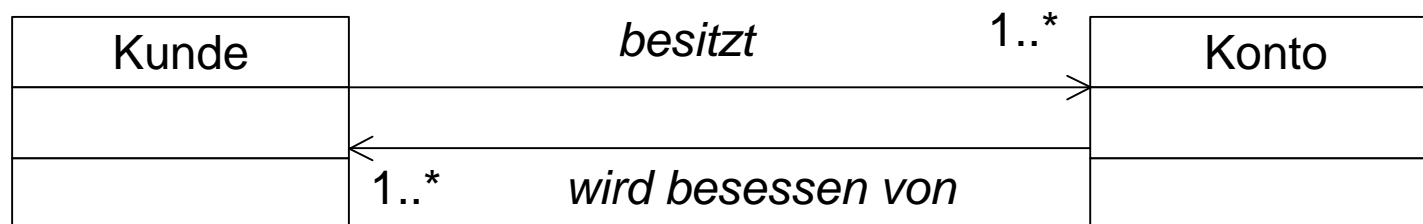


2.3.6 Darstellung von Assoziationen

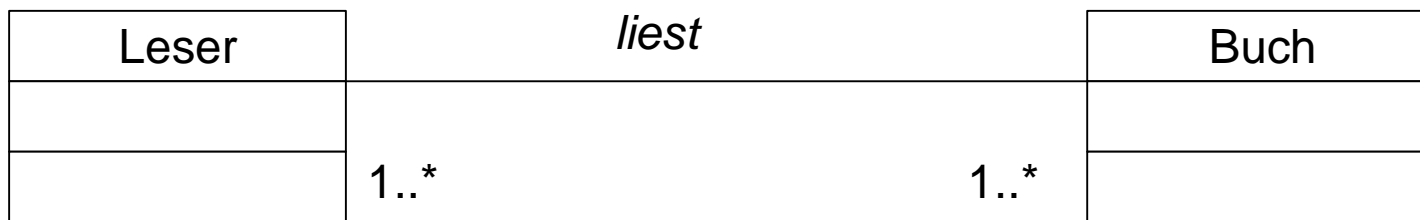
- Unidirektionale Assoziation zwischen Student und Buch
 - Ein Student besitzt kein bis zu beliebig vielen Büchern



- Unidirektionale Assoziation zwischen Kunde und Konto
 - Ein Kunde besitzt mindestens ein Konto
 - Ein Konto kann einem oder mehreren Kunden gehören



- Assoziation zwischen Student und Buch
 - Ein Student besitzt kein bis zu beliebig vielen Büchern
 - Assoziation kann in beide Richtungen gelesen werden

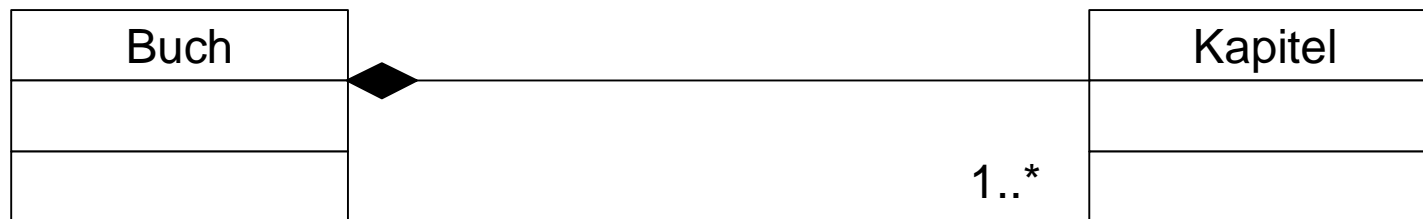


- Bidirektionale Assoziation zwischen Kunde und Konto
 - Ein Kunde besitzt mindestens ein Konto
 - Ein Konto kann einem oder mehreren Kunden gehören

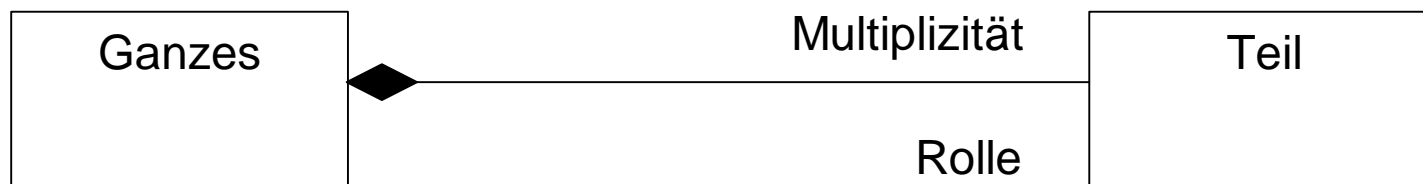


2.3.7 Aggregation vs. Komposition

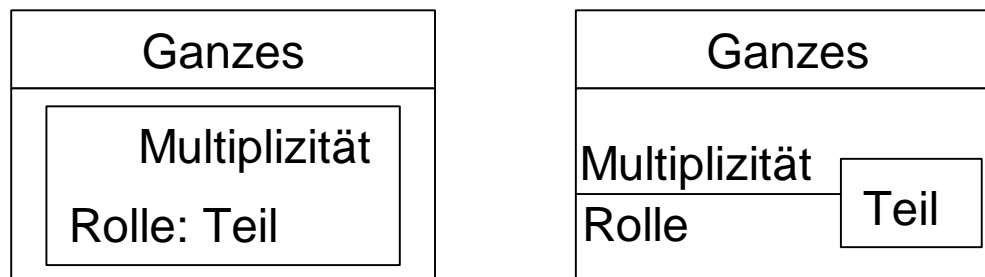
- Komposition
 - Unzertrennbare Objekte werden als Komposition modelliert
d.h. das eine kann ohne das andere nicht existieren



Allgemein:



Oder alternative Darstellung:



- Aggregation
 - Eng verknüpfte Objekte werden als Aggregation modelliert

