

Vorlesung "Software-Engineering"

Prof. Ralf Möller, TUHH, Arbeitsbereich STS
Übung: Miguel Garcia

⌘ Vorige Vorlesung

- ☒ Pflichtenheft (requirements specification document)
Charakterisierung von Software-Qualität
- ☒ Detaillierte Anforderungsanalyse (detailed requirements engineering)

⌘ Heute:

- ☒ Spezifikation mit UML
- ☒ Strukturdiagramme Teil 1

Bücher über UML2

- ⌘ UML 2 glasklar. Praxiswissen für die UML-Modellierung und Zertifizierung
- ⌘ Chris Rupp, Jürgen Hahn, Stefan Queins



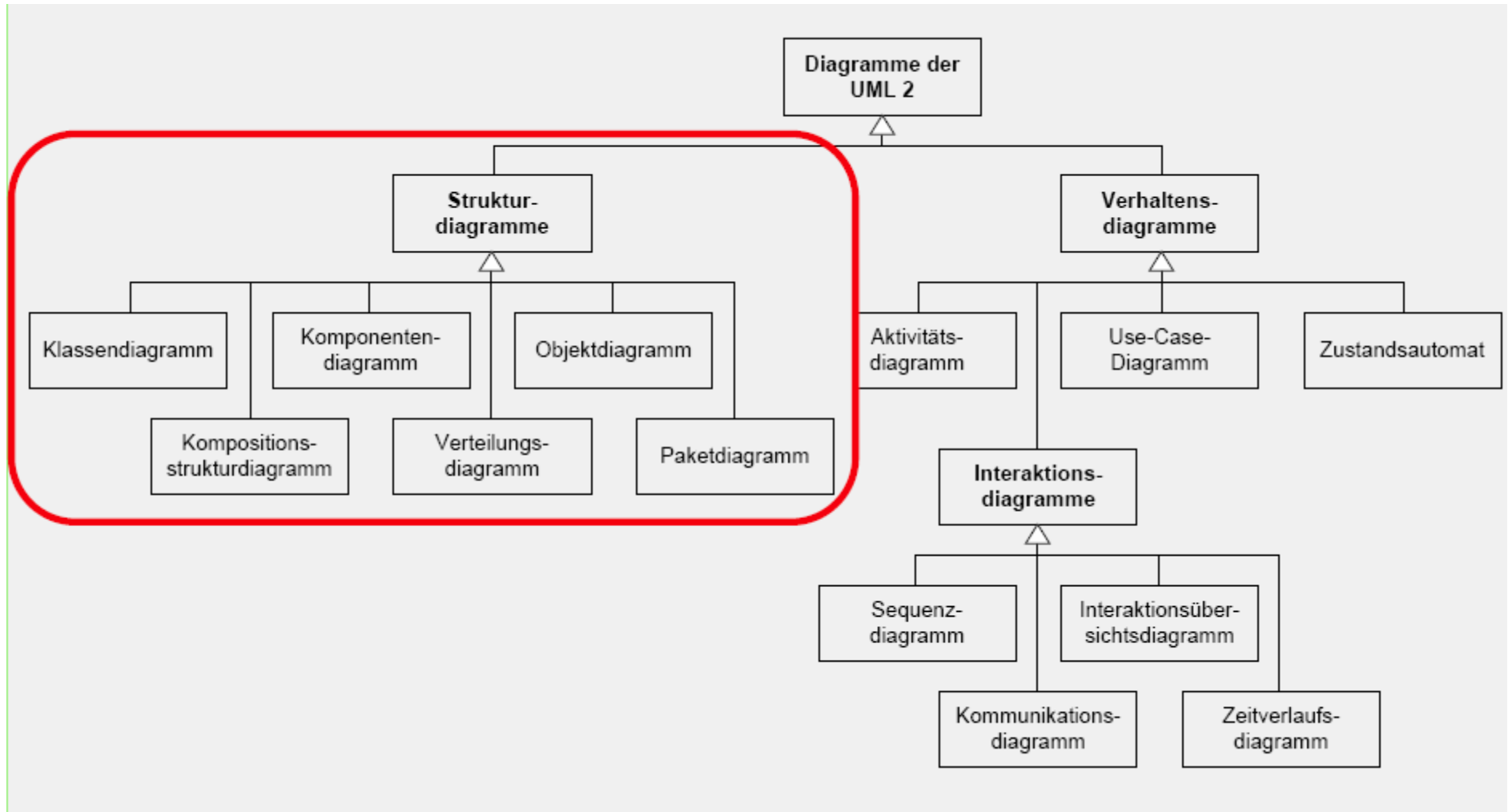
- ⌘ UML 2 und Patterns angewendet - Objektorientierte Softwareentwicklung
- ⌘ Craig Larman



- ⌘ Methodische objektorientierte Softwareentwicklung
- ⌘ Mario Winter



Diagramme der UML2



M. Jeckle: UML 2.0. Modellierung 2004, Marburg, 2004-03-24

Klassendiagramm (1 / 2)

- ⌘ Eine *Klasse* beschreibt Objekte mit gemeinsamen Eigenschaften. Diese Eigenschaften umfassen Struktur, d.h. Attribute und Beziehungen, als auch Verhalten.
- ⌘ Die von einer Klasse beschriebenen Objekten werden als Instanzen dieser Klasse bezeichnet.
 - ☒ Jede Klasse (für sich allein betrachtet) stellt die gültige Wertebereiche ihrer Instanzen fest, z.B. „jede Person muss einen Vornamen und ein Namen besitzen“
 - ☒ Außerdem, Assoziationen und Kompositionen machen Aussagen über die gültigen Beziehungsausprägungen (Verbindungen) zwischen Objekten
z.B. „jedes Auto verfügt über einen Motor, d.h. es kann keine Instanzen von Motor geben die isoliert stehen“.

Klassendiagramm (2 / 2)

- ⌘ Eine UML Klasse wird als Typ in einer Objektorientierten Programmiersprache abgebildet.
- ⌘ OCL (Object Constraint Language) kann auch ins Spiel kommen, um die Aussagen über gültige Wertebereiche und Verbindungen weiter zu verschärfen, z.B.

```
context Person
  inv: isFahrer implies age >= 18
```

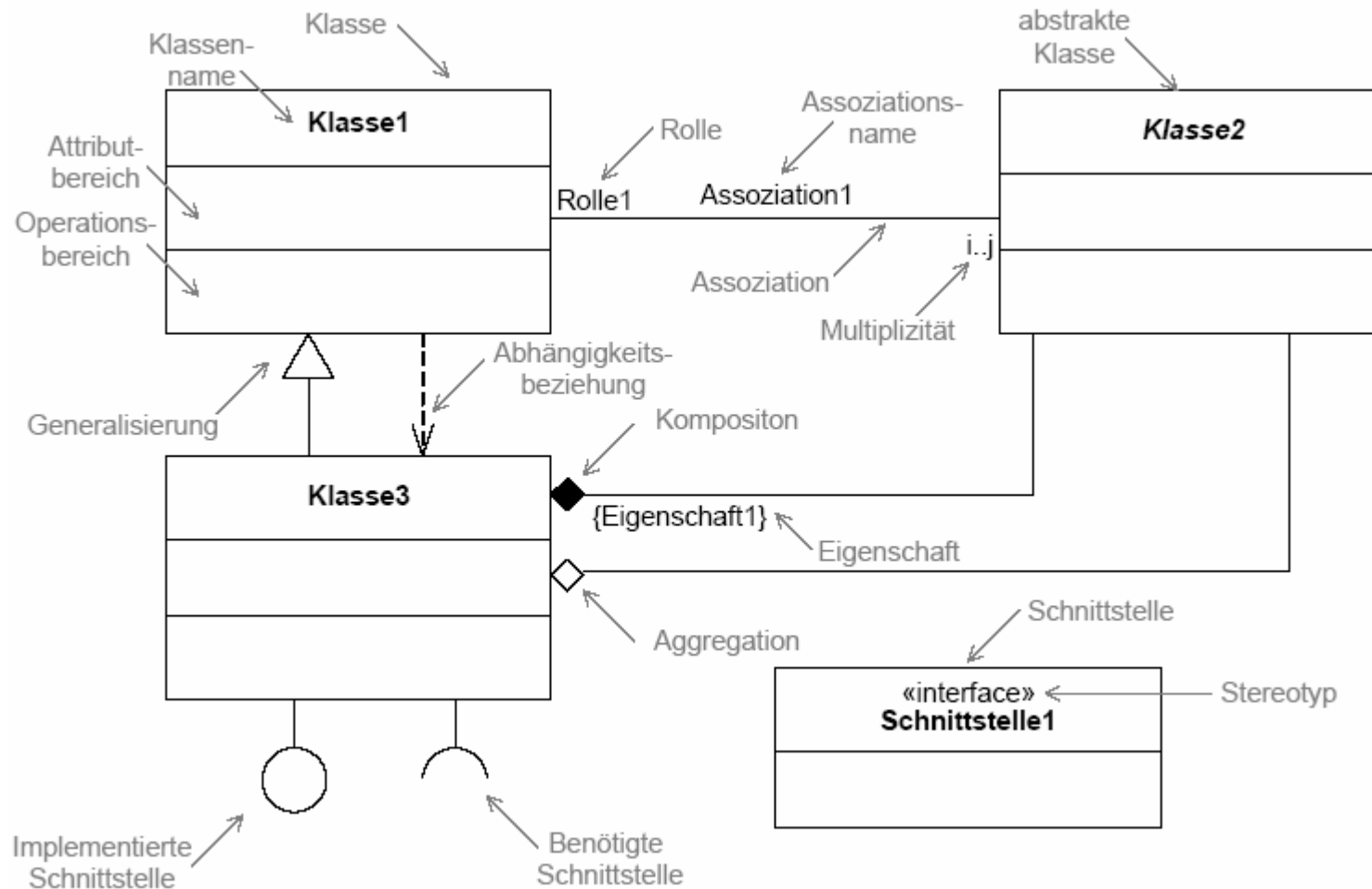
d.h. obwohl nach dem Typ vom Attribut `age` (Integer) Werte unter 18 zulässig sind, wird OCL eingesetzt, um zu dokumentieren, dass diese Werte in Zusammenhang mit `isFahrer = true` nicht zulässig sind. Das System darf nicht solche Welt Darstellungen abbilden. Die Implementierung sollte sich an diese Regeln halten. Das wird durch automatisierte Generierung aus Modellen, Tests oder Verifikation sichergestellt.

Klassendiagramm: Aufgabe in Projekt

⌘ Aufgabe im Projekt:

- ☒ Variierend ...
 - ☒ ... von der ersten Darstellung konzeptueller Dateninhalte
 - ☒ ... über plattformunabhängige logische Modelle
 - ☒ ... bis hin zu „Implementierungsbauplänen“
(„Bilder-für-Java“, „Kästchen-und-Strichchen-statt-C++“)
- ⌘ Ersetzt oftmals das klassische, in Entity Relationship-Notation abgefasste, Datenmodell. In diesem Fall, haben wir mit ORM (Object-Relational Mapping) zu tun.

Visuelle Notation von Klassendiagramme



M. Jeckle: UML 2.0. Modellierung 2004, Marburg, 2004-03-24

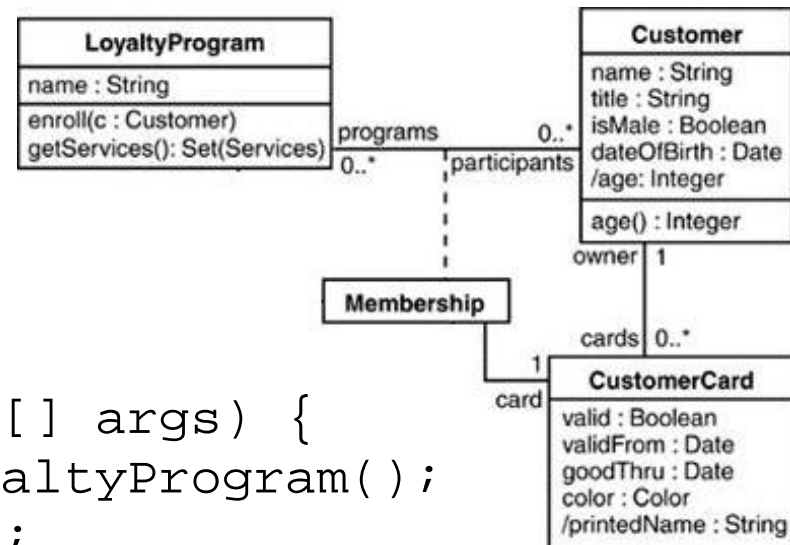
Attribute

- ⌘ Attribute können mit den in Programmiersprachen üblichen Typen typisiert werden:
 - ☑ Boolean, Integer, Real, Date
- ⌘ aber auch mit anwendungsspezifischen Datentypen:
 - ☑ Adresse, Bestellungsnummer
- ⌘ Auf Wunsch, kann man die Deklaration eines Attributs ergänzen um:
 - ☑ einen Defaultwert
 - ☑ Sichtbarkeit (`public`, `private`)
 - ☑ Änderbarkeit (`changeable`, `frozen`)

Operationen

- ⌘ Eine Operation kann Attributwerte lesen oder schreiben, Berechnungen ausführen, Verbindungen zu anderen Objekte knüpfen und lösen sowie Operationen anderer Objekte aufrufen.

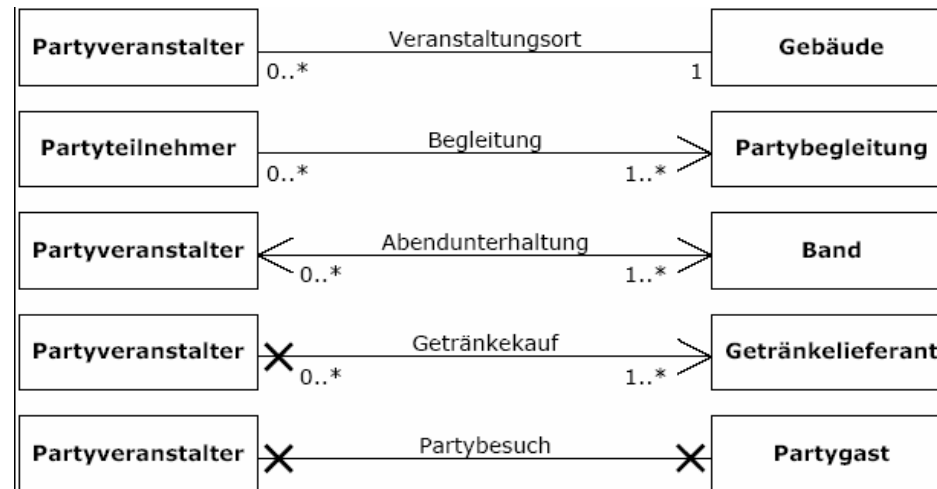
- ⌘ Beispiel:



```
public static void main(String[] args) {
    ILoyaltyProgram lp = new LoyaltyProgram();
    ICustomer c = new Customer();
    c.addToCards(new CustomerCard( true ));
    lp.getParticipants().add(c);
}
```

Assoziationen: Multiplizität, Navigierbarkeit

- ⌘ Die wichtigste Information eines Assoziationsendes ist die Multiplizität, welche die untere und obere Grenze für die Anzahl der bezüglich des Assoziationsendes verbindbaren Instanzen mit einer Instanz der assoziierten Klasse angibt.
- ⌘ Generell sind Assoziationen (in UML 2) mit unspezifizierter Navigierbarkeit ausgestattet.
- ⌘ In Programmiersprachen wird typischerweise ausschließlich unidirektionale Navigierbarkeit unterstützt
- ⌘ Assoziationen mit beidseitigem Navigationsverbot praktisch kaum sinnvoll einsetzbar

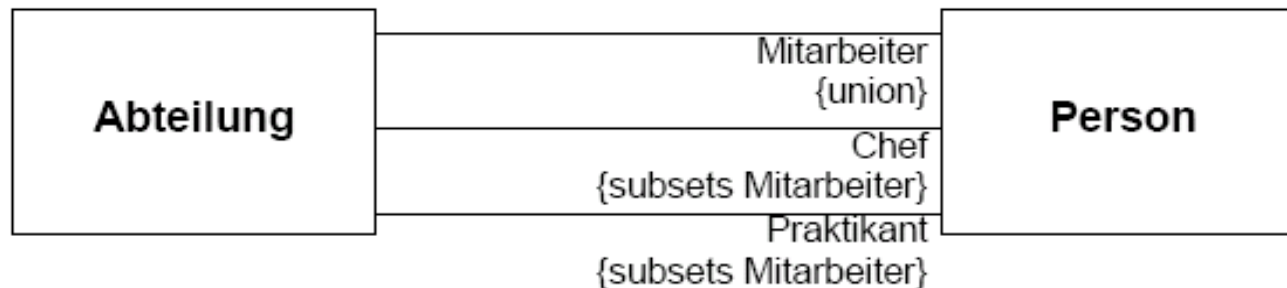


Assoziationsenden

⌘ Den Assoziationsenden wird Detailinformation zugefügt:

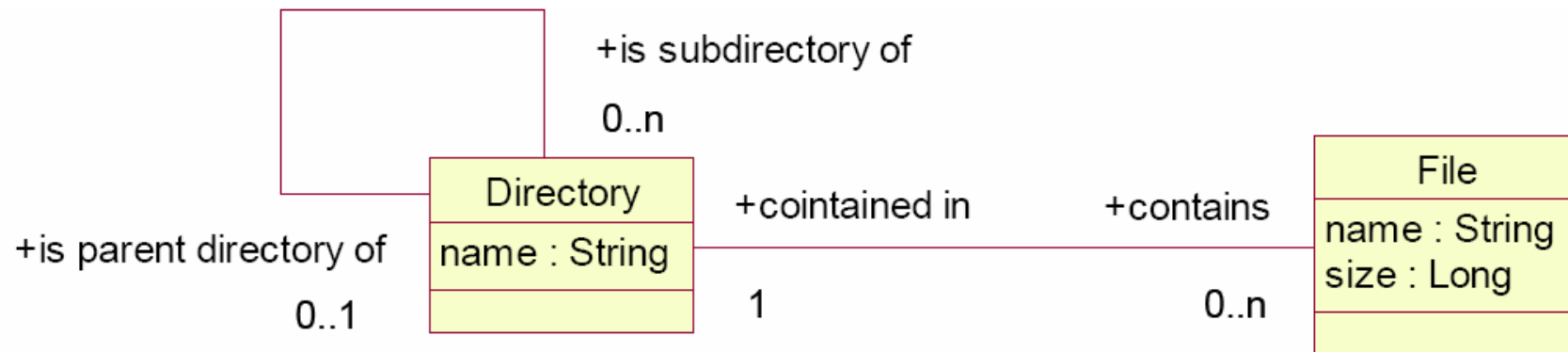
- ☒ Multiplizität
- ☒ Navigierbarkeit
- ☒ {ordered}
- ☒ {unique} (d.h. Duplikate werden vermieden)
- ☒ {subsets} (Beispiel unten)

Um die bezüglich des Assoziationsendes verbindbaren Instanzen zu beschränken. Ist aus semantischer Sicht sinnvoll aber hat keine vorgegebene Abbildung in Programmiersprachen.



Reflexive Assoziationen

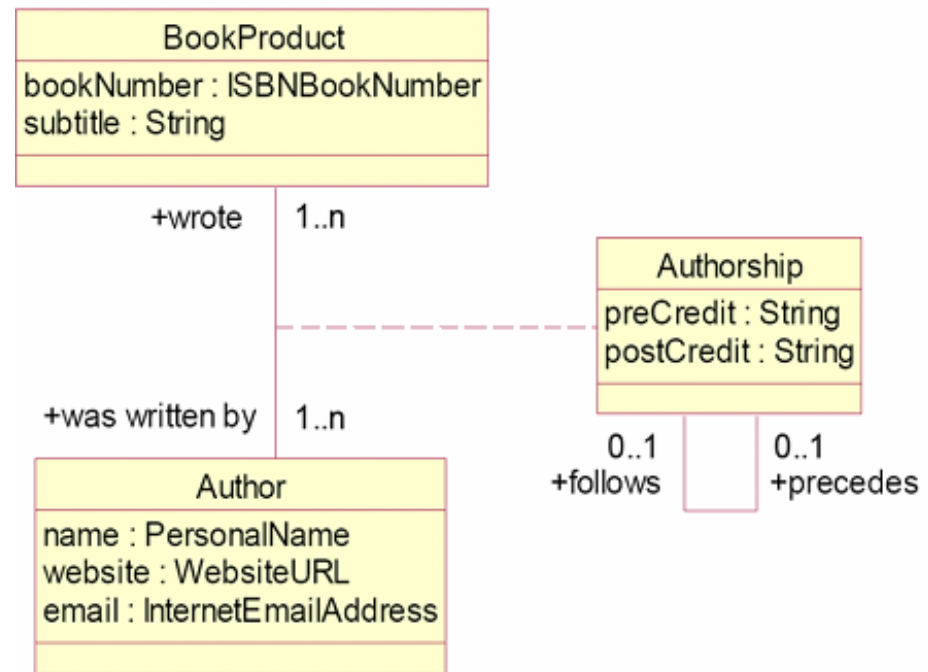
- ⌘ Sollen Verbindungen zwischen Instanzen ein und derselben Klasse möglich sein, so bezieht sich die Assoziation nur auf diese eine Klasse und man spricht von einer *reflexiven Assoziation*
- ⌘ Zu beachten ist, dass sowohl Assoziationen zwischen unterschiedlichen Klassen als auch reflexive Assoziationen zwei Enden besitzen
- ⌘ Beispiel:



Assoziationsklassen

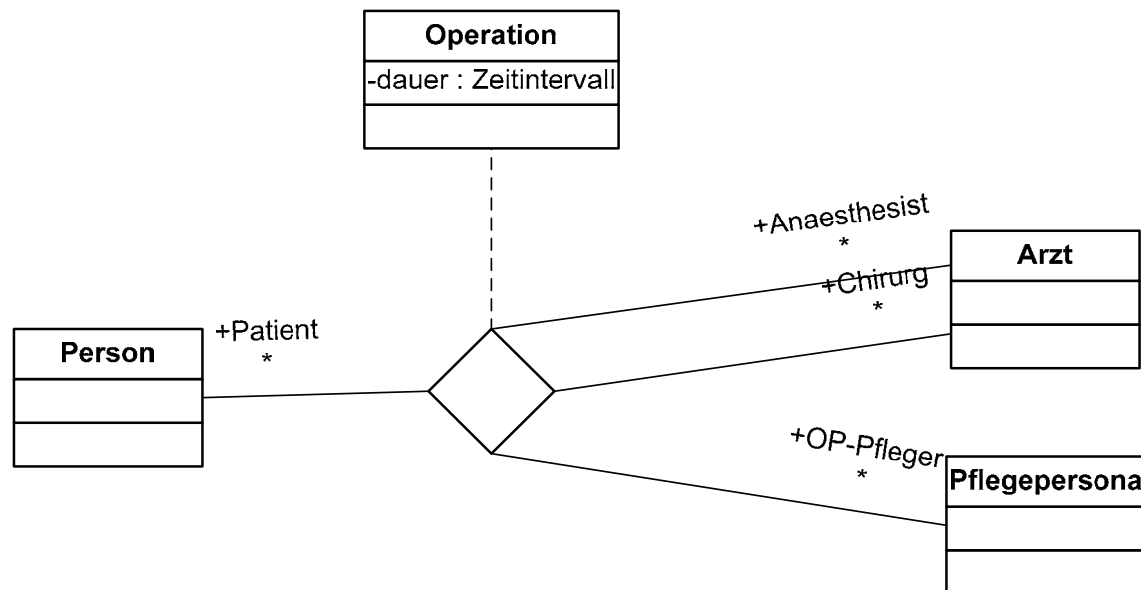
- ⌘ Häufig muss eine Assoziation ausgestattet werden, z.B. mit dem Datum, wann eine Verbindung zwischen zwei Objekten eingegangen wurde.
- ⌘ Eine Assoziationsklasse wird im Klassendiagramm durch eine gestrichelte Linie mit der entsprechenden Assoziationslinie verbunden

- ⌘ Eine Assoziationsklasse kann auch mit Assoziationen und Operationen versehen werden



N-äre Assoziationen (1 / 2)

- ⌘ Sachverhalte, an denen mehr als zwei Objekte beteiligt sind, lassen sich mit binären Assoziationen nicht adäquat modellieren



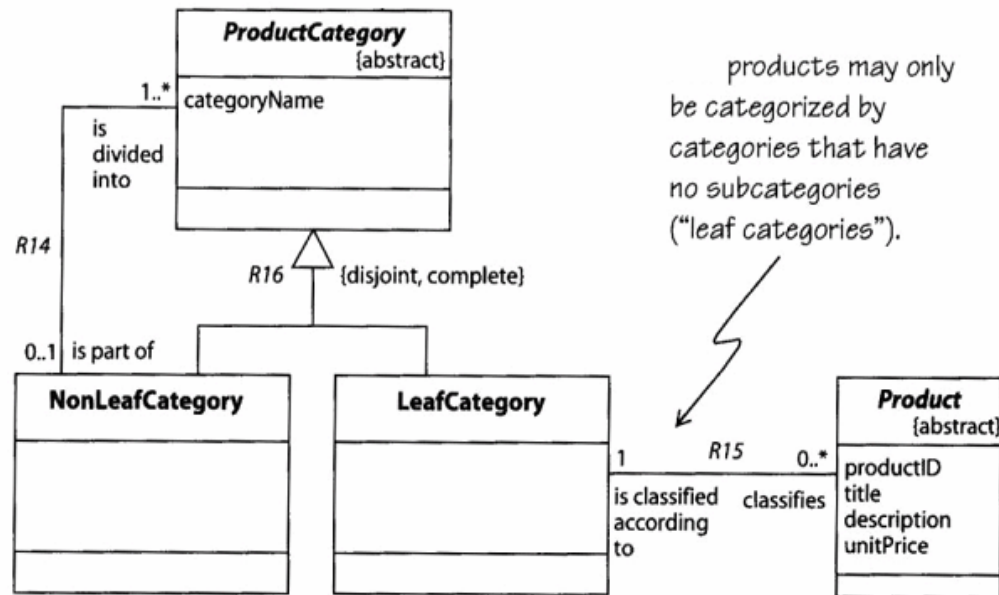
Methodische objektorientierte Softwareentwicklung, Mario Winter

N-äre Assoziationen (2 / 2)

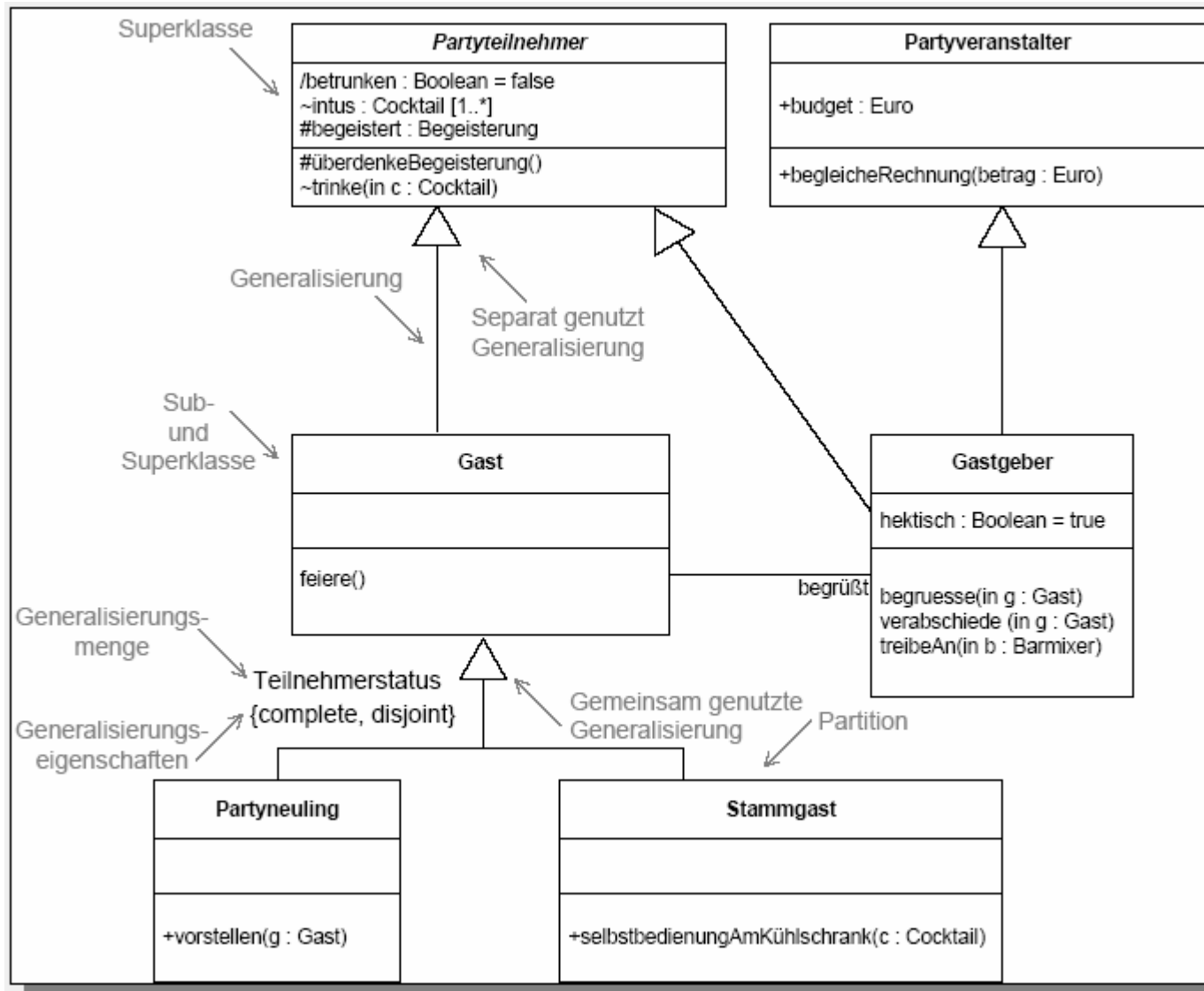
- ⌘ Bei n-ären Assoziationen ($n > 2$) darf die Navigierbarkeit nicht eingeschränkt werden
- ⌘ Ist die obere Grenze der Multiplizität an einem Ende größer als eins, so sind die verbundenen Instanzen der diesbezüglichen Klasse jeweils durch ein $(n-1)$ -Tupel von Instanzen der anderen an den Assoziation beteiligten Klassen eindeutig identifizierbar.

Generalisierung (1 / 2)

- ⌘ Die objektorientierte Modellierung bietet die Möglichkeit, Gemeinsamkeiten z.B. bei Attributen, Operationen oder Assoziationen unterschiedlicher spezieller Klassen mit Hilfe einer allgemeineren Klasse zusammenzufassen. Es wird dadurch Redundanz vermieden.



Generalisierung (2 / 2)



Interface

- ⌘ Man bezeichnet den Operationsnamen zusammen mit den Wertebereichen der Parameter und des Rückgabewerts als die *Signatur* einer Operation.
- ⌘ Ein *Interface* (Schnittstelle) ist eine Sammlung von Signaturen von Operationen (ohne Implementierung). Klassen, die ein Interface unterstützen, sind dazu verpflichtet, alle Operationen des Interfaces zu implementieren.
- ⌘ Solche Abhängigkeit wird auch als *Vertrag* zwischen den involvierten Klassen bezeichnet.
- ⌘ Interfaces besitzen weder Attribute noch Assoziationen. Sie dürfen aber Ziel einseitig navigierbarer Assoziationen sein.

Die andere Strukturdiagramme in UML2

- ⌘ Objektdiagramm:
Enthält Objekte und Beziehungsausprägungen
- ⌘ Paketdiagramm:
Stellt die logische Organisation von Modellelementen und deren Abhängigkeiten dar
- ⌘ Komponentendiagramm:
Zeigt die Organisation und Abhängigkeiten von Komponenten
- ⌘ Kompositionsstrukturdiagramm:
Zeigt die interne Struktur eines Classifiers sowie seine Möglichkeiten zur Interaktion mit anderen Systemkomponenten
- ⌘ Verteilungsdiagramm:
Zeigt die Ausführungssicht des Systems