Techniken der Projektentwicklung

Klassendiagramme

Franz Kummert, Gerhard Sagerer

Termin 6

Motivation

Objektorientierte Modellierung
- Denken in Klassen
- Schnittstellen
- Was nun?

UML-Klassendiagramme
- Klassen
- Beziehungen
- Zusammenfassung

Einführung

Bisher kennengelernt:

- Modellierung auf Konzeptlevel
- Usecase-Diagramme
- Domänenmodelle

Jetzt: Übergang zu Spezifikation und Implementierung!

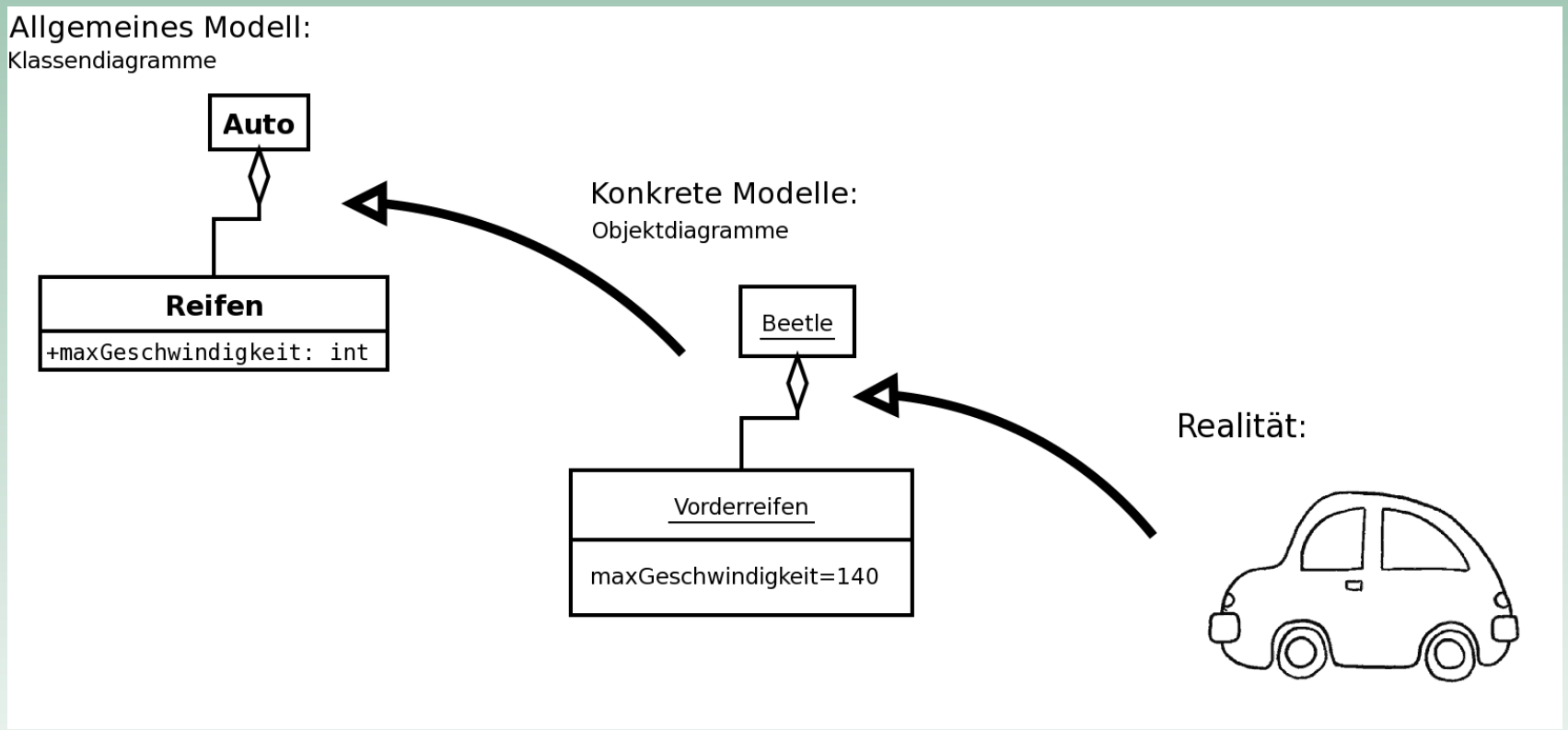
- Vom Domänenmodell zum Klassendiagramm

Statische Modellierung

Stärkere Detailierung der statischen Modellierung nötig, z.B. durch:

- *Attribute* von Objekten und Klassen
- Spezifikation von *Verantwortlichkeiten* durch Methoden
- genauerer Spezifikation von *Beziehungen* zw. Klassen
- *Schnittstellen*
- Gruppierung und Strukturierung durch *Pakete*

Modellentwicklung



Abstraktionen als Schlüssel zur Modellbildung!

Denken in Klassen

Wiederholung: Was sind Klassen?

- Eine Klasse beschreibt *Objekte gleicher Struktur und gleichen Verhaltens*:

```
class Point {
  int x = 0, y = 0;

  void move (int dx, int dy) {
    x += dx;
    y += dy;
  }
}
```

- Klassendefinitionen: Schnittstellen mit *Operationen und Attributen* der Klasse
- Klassen dienen als „*Bauanleitung*" für die Konstruktion konkreter Objektinstanzen

Denken in Klassen

Ziele objektorientierter Modellierung:

- bessere Abbildung der Realität (Modellierung) als bei rein prozeduralen Methoden
- Wartbarkeit
- Erweiterbarkeit
- Wiederverwendbarkeit

Konzepte dafür:

- Vererbung
- Polymorphie
- Kapselung

Schnittstellen

Design-Prinzip: Schnittstellen-orientierter Entwurf

- legen öffentliche Methodensignaturen fest
- aber: legen keine Implementierung fest
- stellen einheitlichen Zugriff auf Funktionalität sicher
- fassen Klassen zu Funktionalitätsgruppen zusammen

Prinzip des wiederverwendbaren objektorientierten Entwurfs:

Programmiere auf eine Schnittstelle hin, nicht auf eine Implementierung.
(Gamma, 1996)

Schnittstellen

Warum Schnittstellen?

- fördert lose Kopplung zwischen Komponenten
- Schnittstelle ist ,,Vertrag" zwischen unabhängigen Komponenten
- Implementierungsdetails werden gekapselt
- ermöglicht Anwendung dynamischer Polymorphie (Substitutionsprinzip)

Übergang zur Implementierung

Was fehlt im Domänenmodell?

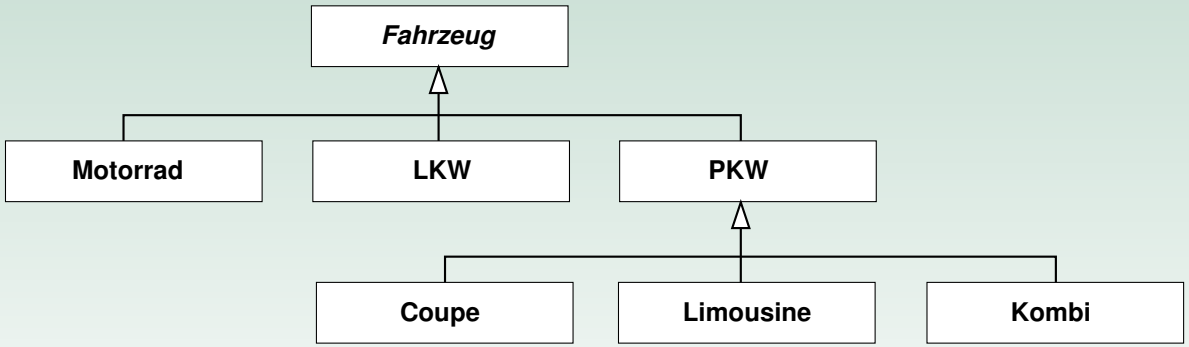
- synthetische Konzepte
- Vererbung nicht konkret formuliert
- i.d.R keine Zuständigkeiten festgelegt
- keine Schnittstellen

Lösung: Klassendiagramme

Klassendiagramme

- Konkretisierung des Domänenmodells
- Einzelne *Klasse*: Rechteck mit Klassenname

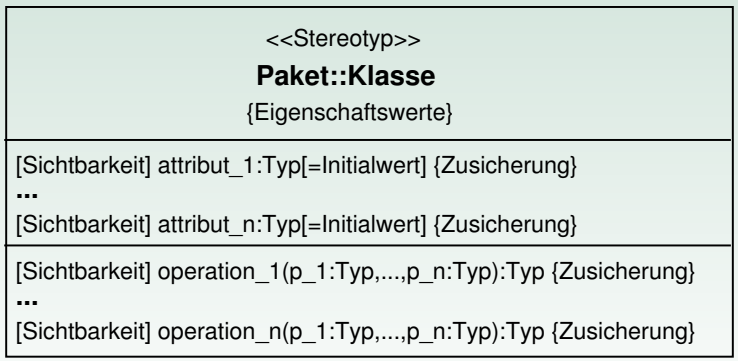
- weitere Angaben möglich: Attribute, Methoden, Typen, ...
- Beschreibung von *Beziehungen* zwischen Klassen:



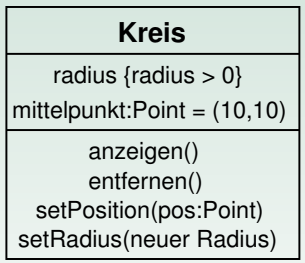
Klassen

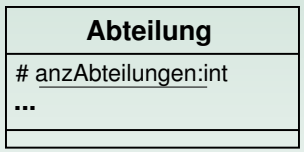
Spezifikation aufgeteilt in drei Bereiche:

1. Klassenname (evtl. Zusatzinfos)
   Notation: `Paket::Klasse`
2. Attribute (Variablenbeschreibungen)
   Notation: `attribut:[Typ[=Initialwert]]`
3. Methoden (Verantwortlichkeiten)
   Notation: `operation([p1[:Typ],...,pn[:Typ]])`



p_i stehr für den i–ten Parameter der jeweiligen Methode





Sichtbarkeit

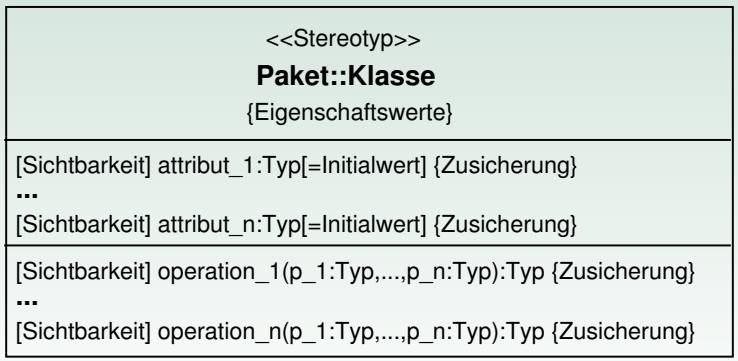
Access-Modifier

`+` `public`

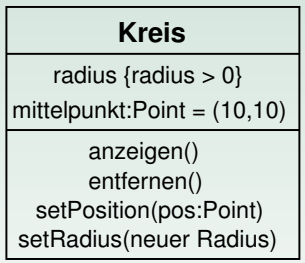
`#` `protected`

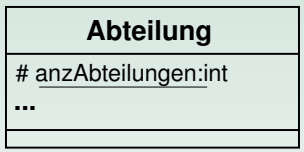
`-` `private`

- So öffentlich wie nötig, so privat wie möglich
- Attribute (fast) immer `private`



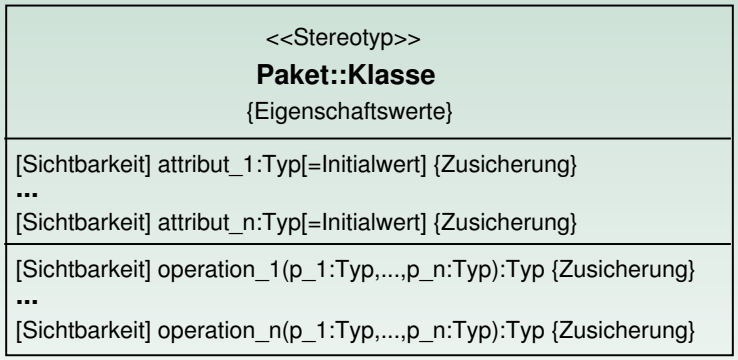
p_i stehr für den i–ten Parameter der jeweiligen Methode



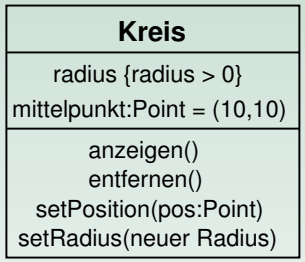


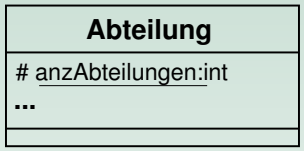
Stereotypen

- Spezifikation neuer Modellelemente
- durch Erweiterung der Basiselemente
- neue Semantik der abgeleiteten Elemente
- Beispiele: `<<interface>>` abgeleitet von Element ,,Class"



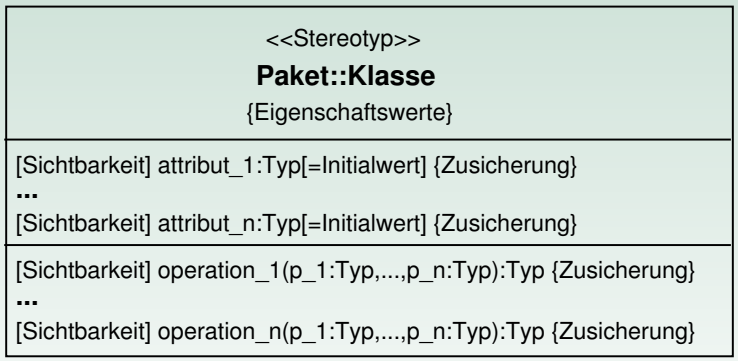
p_i stehr für den i–ten Parameter der jeweiligen Methode



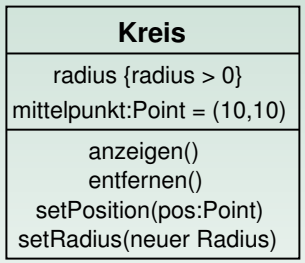


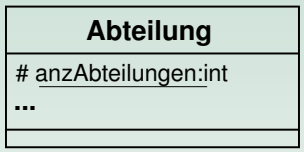
Constraints

- Zusicherungen (Assertions)
- Einschränkungen und Integritätsregeln
- Anreicherung von Spezifikationen
- Notation mit Hilfe der Object Constraint Language
- Beispiel: {`Kreis.radius>0`}



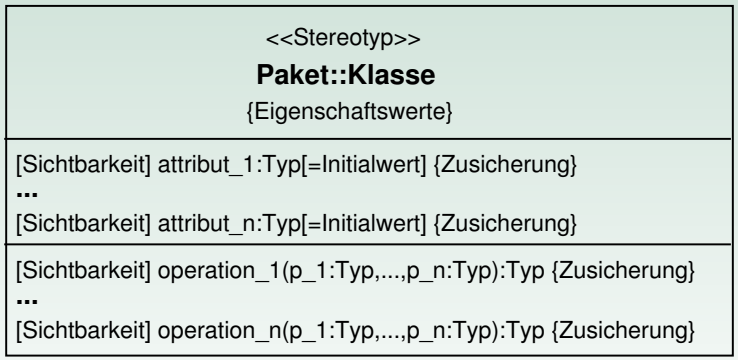
p_i stehr für den i–ten Parameter der jeweiligen Methode



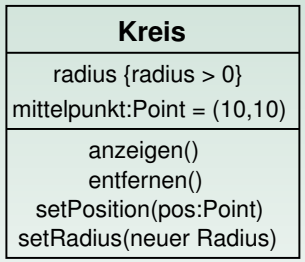


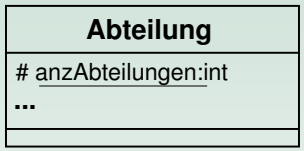
Tagged Values

- semantische Spezifikation
- Schlüssel-Wert-Paare
- Anreicherung von Meta-Information
- Beispiele:
  - {throws=ElevatorStuckException}
  - {abstract}



p_i stehr für den i–ten Parameter der jeweiligen Methode



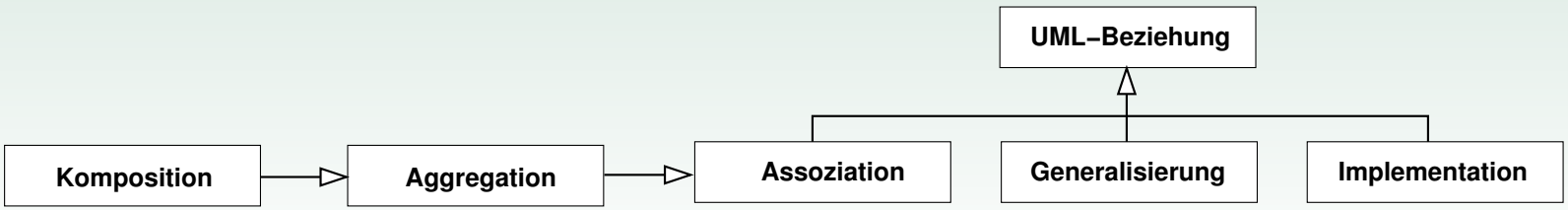


Relationen

UML Relationships:

- *Generalisierung (Generalisation):* Beziehung zwischen speziellen und allgemeinen Konzepten
- *Implementation (Realisation):* Verknüpfung von Spezifikation und Implementierung
- *Assoziation (Association):* Verknüpfung von Objekten
- *Aggregation:* Teil-Ganzes Beziehung
- *Komposition (Composition):* Spezialfall der Aggregation

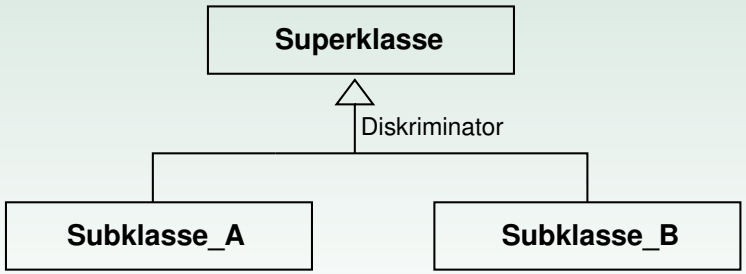
Zusammenhänge zwischen diesen Relationen:

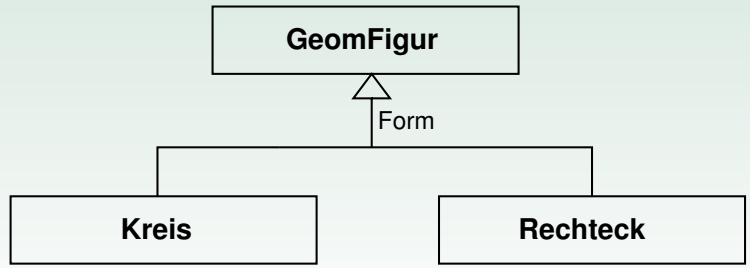


Generalisierung

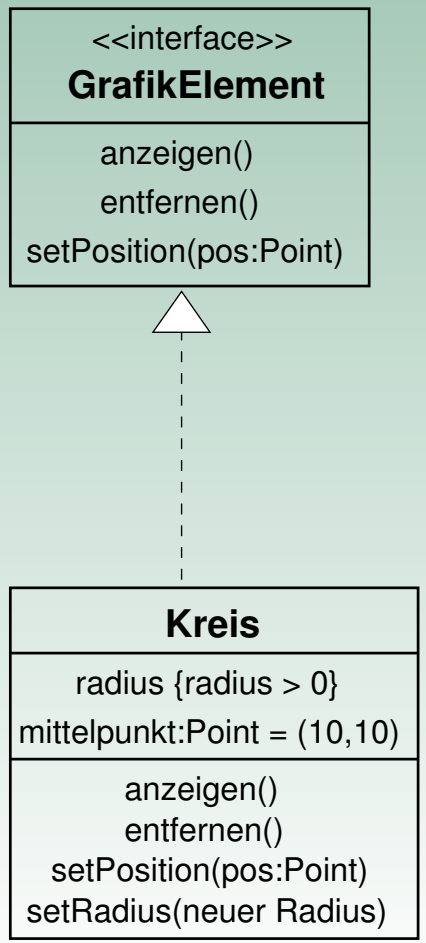
Generalisierung im Kontext der UML:

- gemeinsame Eigenschaften von Klassen extrahieren
- Code-Wiederverwendung
- realisiert durch *Vererbung*
- Unterscheidungsmerkmal wird als ,,Diskriminator" bezeichnet
- definiert durch eine *is-a* Beziehung
- Notation: **geschlossener Pfeil zeigt von Unter- auf Oberklasse**





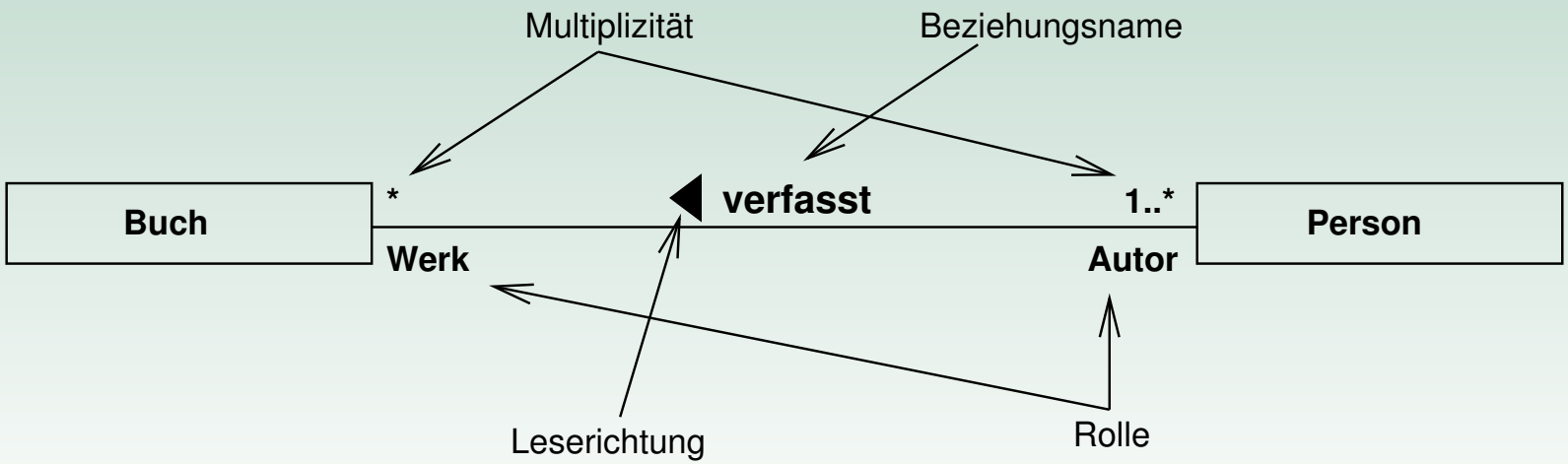
Implementation



- "Ausfüllen" von Interfaces oder abstrakten Klassen
- Spezifikationen der Schnittstelle werden implementiert
- Notation: **gestrichelter Pfeil zeigt von Implementierung auf Interface**

Assoziationen

- Verknüpfungen mit gleicher Semantik
- Gleichwertige Beziehung zwischen Klassen
- Angaben: Rollen, Beziehungsname, Kardinalitäten
- Auf Implementierungsebene: Variablen des entsprechenden Typs

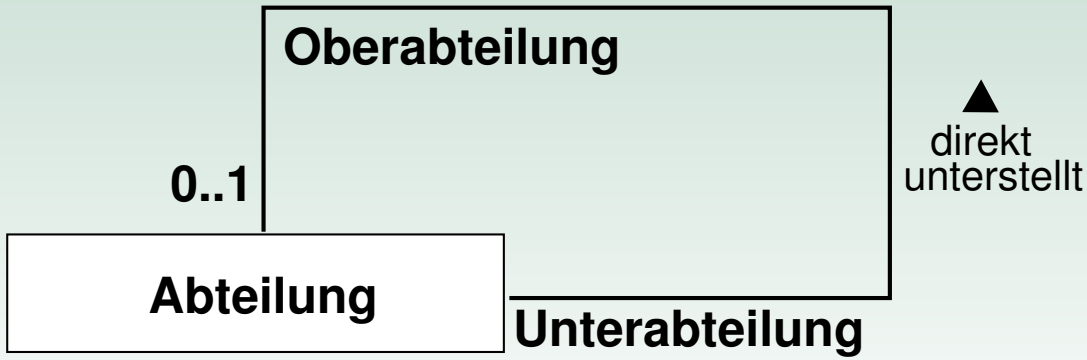


Navigation

Anmerkungen zu Rollen und Navigierbarkeit:

- Beziehungsnamen, wenn Rolle eines Objekts in einer Relation unklar, z.B. in rekursiven Assoziationen
- Leserichtung wird auch als Navigationsrichtung bezeichnet

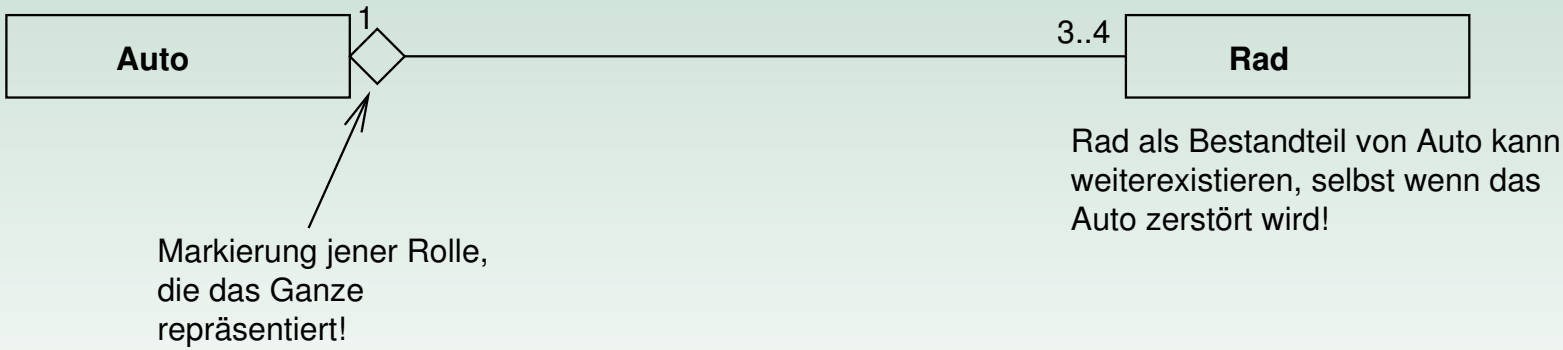
Beispiel: Modellierung einer hierarchischen Abteilungsstruktur



Aggregation

- Sonderform der allgemeinen Assoziation
- *keine gleichwertige Beziehung* zwischen beteiligten Klassen
- repräsentiert eine „Teile-Ganzes-" oder „Besteht-aus-" Beziehung (*part of*)

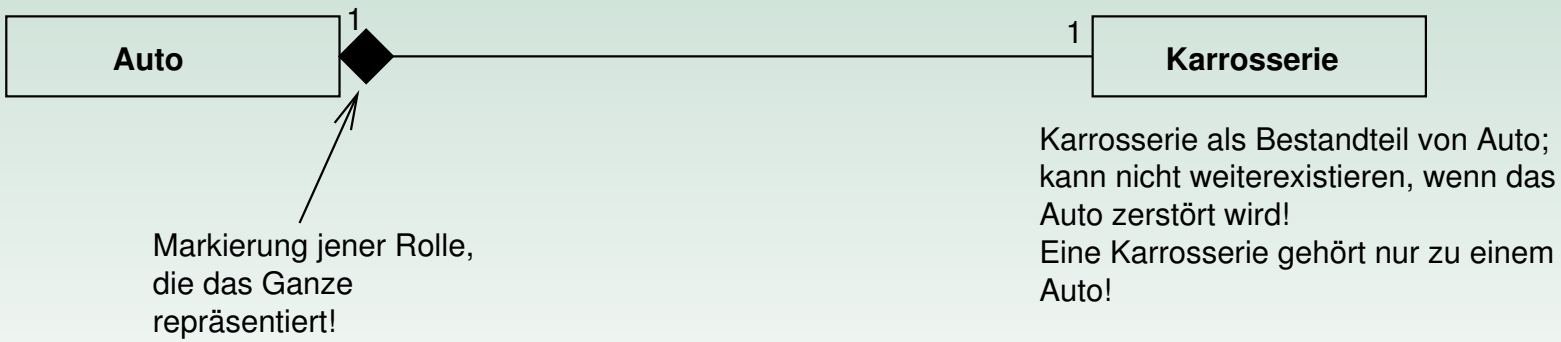
Beispiel: Zu einem Auto gehören mind. 3, maximal 4 Räder



Komposition

- strenge Sonderform der Aggregation
- Exemplare dieser Klassen nicht individuell vorhanden, sondern vom Ganzen *existenzabhängig*
- Modellierungentscheidung!

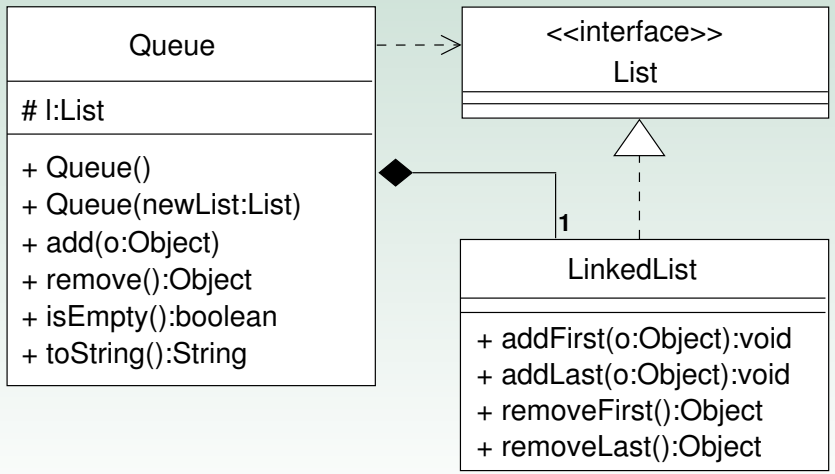
Beispiel: Zu einem Auto gehört immer genau eine Karroserie



Abhängigkeiten

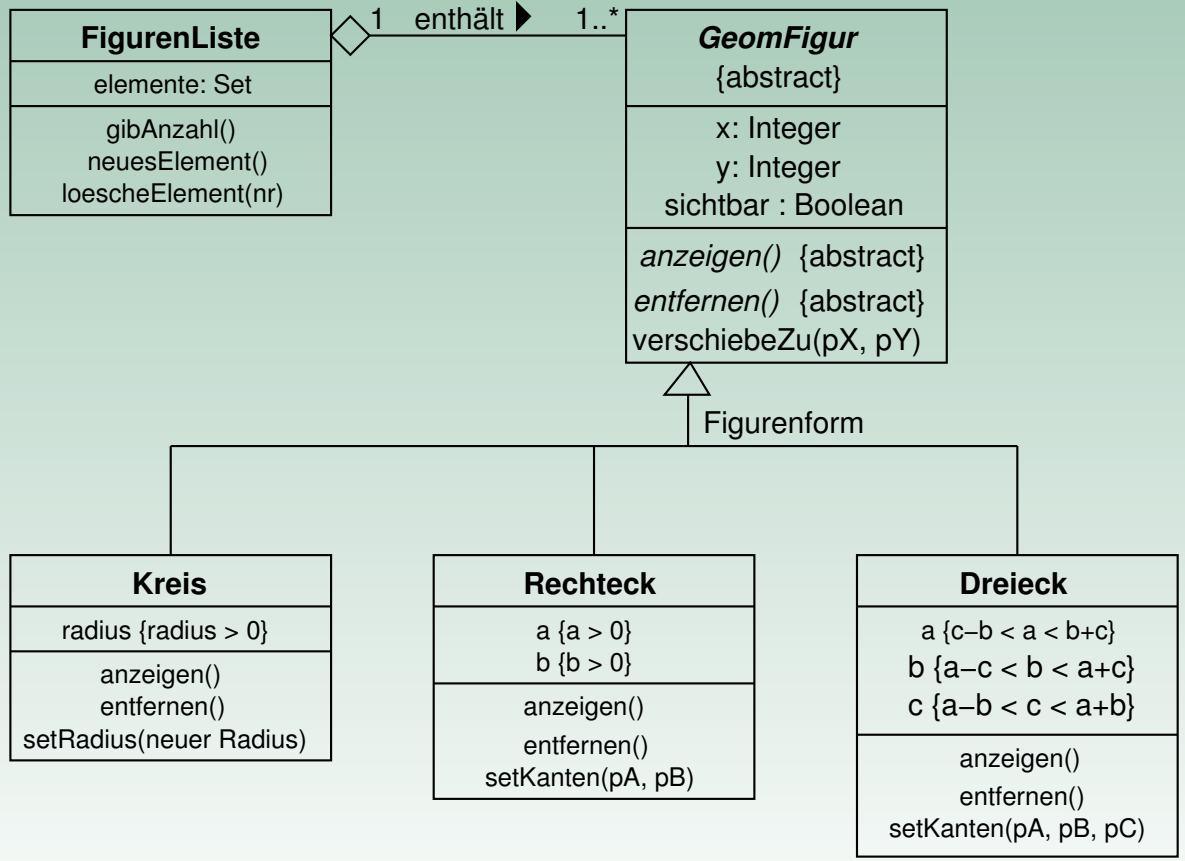
Dependencies zwischen Klassen in UML:

- Abhängigkeit: Semantische Beziehung zwischen Modellelementen
- Änderung des Einen berührt die Semantik des Anderen
- Notation einer Abhängigkeit: Gestrichelter Pfeil mit offener Spitze, optional mit Stereotyp, z.B. <<uses>>



Beispiele

Weiteres Beispiel: Modellierung geometrischer Figuren



Zusammenfassung

- Klassen enthalten Attribute und Methoden
- Schnittstellen als wichtiges OO-Konzept
- Klassendiagramme stellen Klassen und die Beziehungen unter ihnen dar
  - Assoziationen, Aggregation, Komposition
  - Generalisierung, Implementation
  - Abhängigkeiten

Vielen Dank für die Aufmerksamkeit