

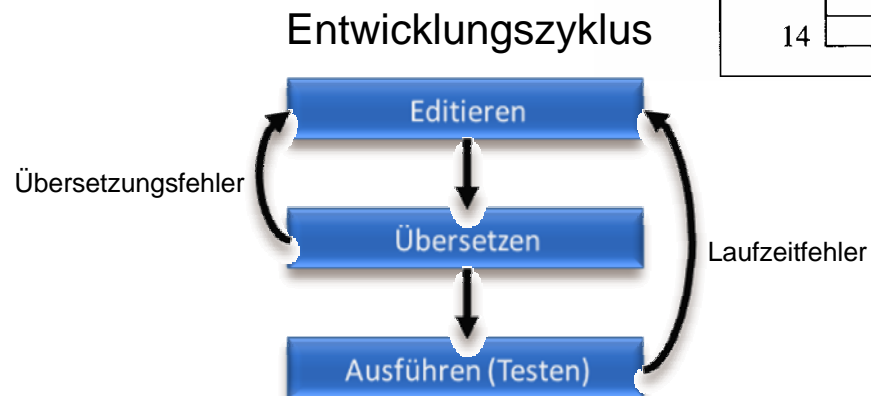
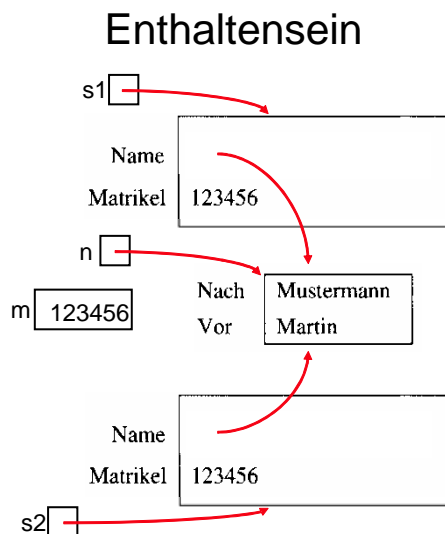
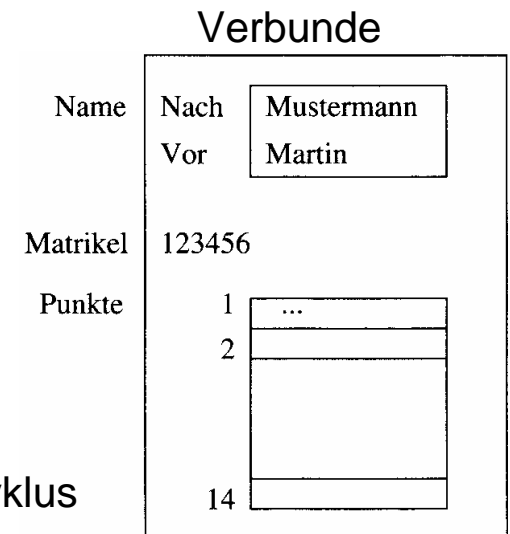
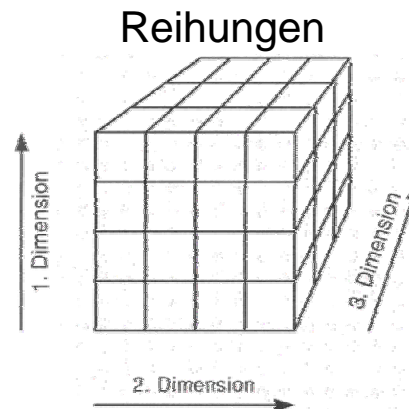
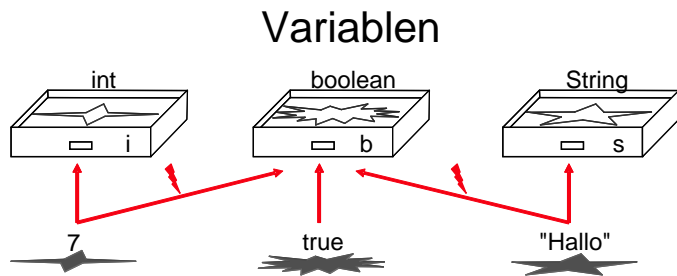
# **Praktische Informatik 1**

## **Objektorientierte Software-Konzepte und UML**

Thomas Röfer

- Funktionale Dekomposition vs. Objektorientierter Ansatz
- Objekte und Klassen
- Objektbeziehungen
- Beispiel
- Musterlösung, Übungsblatt

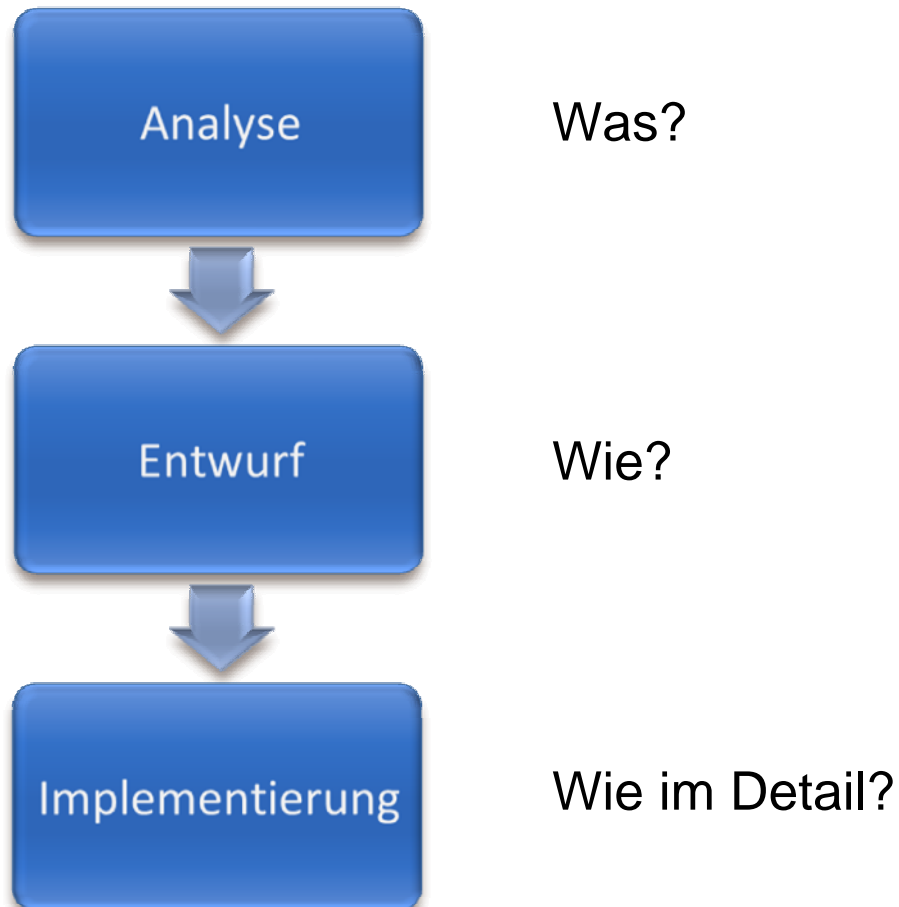
# Rückblick „Konzepte benutzerdefinierter Datenstrukturen“



# Funktionale Dekomposition

- Schritte
  - Abstrakte Spezifikation der Funktion
  - Hierarchisch absteigende Zerlegung der Funktion in immer elementarere Teilfunktionen
  - Ausprogrammieren der Funktionen
- Beispiel
  - Verwaltung der PI-1 Studierenden
    - Einfügen, Ändern, Löschen
    - z.B. Einfügen: Leeren Datensatz erzeugen, ändern, in Datenbank einfügen
- Nachteil
  - Datenstrukturen sind nicht Teil des Modellierungsprozesses

# Objektorientierter Ansatz der Software-Entwicklung





# Objektorientierter Ansatz

## Analyse

- Suche nach Objekten und Objektbeziehungen in der realen Welt
- Wie soll Software genutzt werden?
- Welche Funktionalität soll wem zur Verfügung gestellt werden?
- **Was** geschieht **warum** mit Objekten?

# Objektorientierter Ansatz Entwurf

- Übertragung in die Welt der Software
- Ergänzung und Modifikation aus programmiertechnischen Notwendigkeiten
- **Wie** soll etwas **im Prinzip** geschehen?
- Modell der Software-Architektur

# Objektorientierter Ansatz

## Implementierung

- Software-Architektur wird zum lauffähigen Programm konkretisiert
- Objektzustände werden durch Datenstrukturen repräsentiert
- Objektfunktionalität wird durch Algorithmen realisiert und ausprogrammiert
- Festlegung, **wie** alles **im Einzelnen** geschieht

# Objekte

- Objekt

- Gedankliche oder reale Einheit in der Umwelt oder Software
- Ein Objekt hat einen **Zustand** und eine **Funktionalität**

<u>:Name</u>
Zustand
Funktionalität

- Objekt-Zustand

- **Zustandsvariablen**, auch **Attribute** des Objekts
- Attribute sollten nicht nach außen **sichtbar** sein (**Geheimnisprinzip**, **Kapselung**), sondern nur über Methoden (**Selektoren**, set-/get-Methoden) gesetzt oder gelesen werden können

# Objekt-Funktionalität

- Nach innen: Veränderung des Objektzustandes
- Nach außen: Wirkung auf andere Objekte, zu denen **Objektbeziehungen** bestehen
- Wird realisiert durch die **Methoden** eines Objekts
- Interaktion mit Objekt geschieht ausschließlich über dessen Methodenschnittstelle

<u>:Name</u>
Zustand
Funktionalität

# Objekte und (Objekt-)Klassen

- Objekte sind Einzelstücke
- Objekte mit gleichen Attributen und Methoden gehören derselben (Objekt-)Klasse an
- Eine Klasse ist der **Typ** eines Objekts
- Objekte sind **Instanzen** einer Klasse

Personal
n : Name; d : Geburtstag; o : Geburtsort;

Klasse

<u>a : Personal</u>
n = "Peter"; d = "29.2.1981"; o = "Schilda";

<u>b : Personal</u>
n = "Paul"; d = "31.6.1979"; o = "Fulda";

<u>c : Personal</u>
n = "Maria"; d = "31.4.1982"; o = "Werra";

Instanzen

# Beispiele für Klassen

## TV2000

```
z : EinAus;  
s : Seriennummer;  
k : Kanal;  
l : Lautstärke;  
  
ein();  
aus();  
wähle_Kanal();  
wähle_Lautstärke();  
get_Seriennummer();
```

## Motor

```
z : EinAus;  
v : Vorwärtslauf;  
r : Rückwärtslauf;  
d : Drehzahl;  
  
ein();  
aus();  
vorwärts();  
rückwärts();  
set_Drehzahl();  
get_Drehzahl();
```

## Robot

```
z : EinAus;  
p : Greifposition;  
  
ein();  
aus();  
bewege_Punkt_zu_Punkt();  
bewege_linear();
```

# Objektbeziehungen

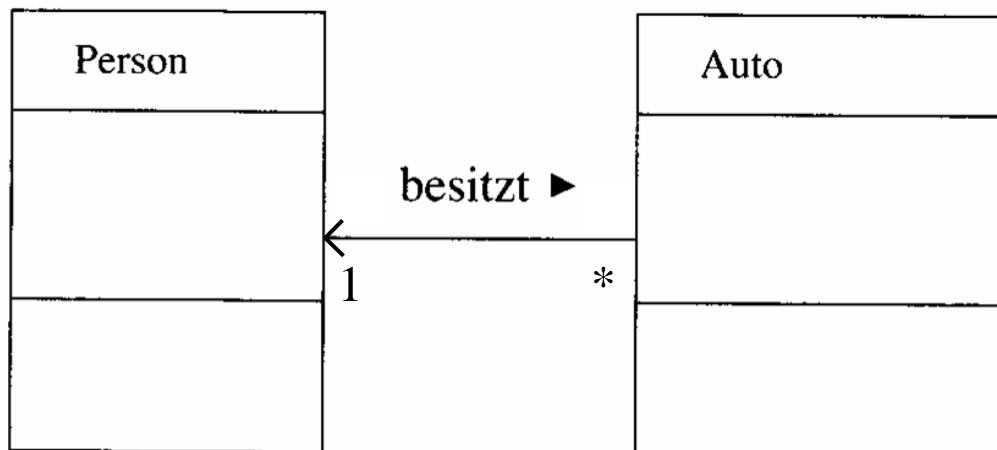
- Verhaltensbezogene Beziehungen
  - Informationsfluss oder Nachrichten (**message**)
  - Funktionsaufruf oder Kunde/Lieferant (**client/server**)
- Strukturelle Beziehungen
  - Einschluss (**has-a**)
  - Subtyp oder Vererbung (**is-a**)



# Objektbeziehungen

## Beziehungen zwischen Klassen

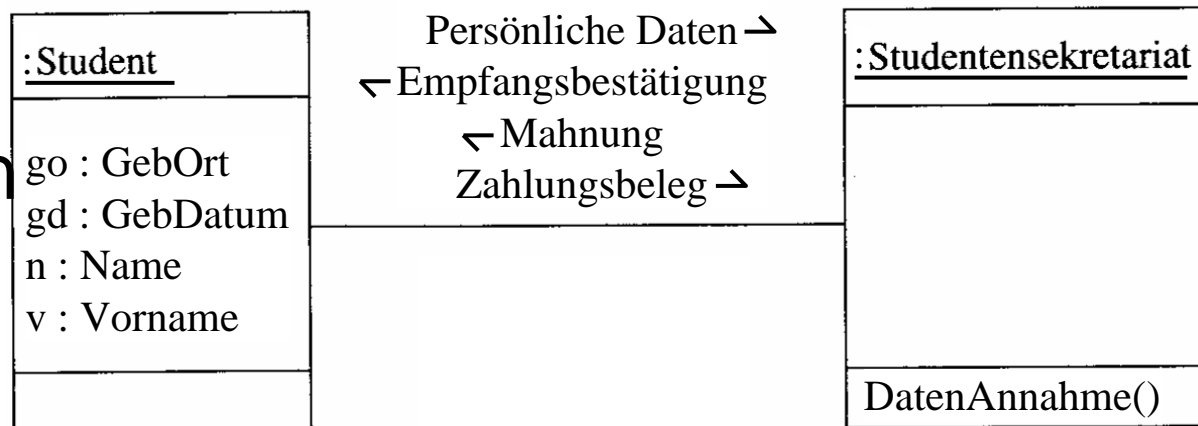
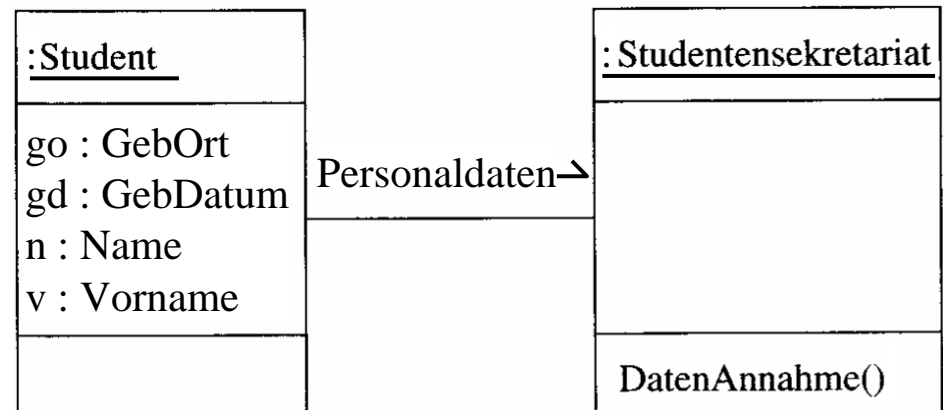
- Darstellung durch Verbindungspfeil
- Vielfachheit, z.B. 0..2, \*, 1..\*
- \* steht für „beliebig viele“



# Informationsfluss oder Nachrichten

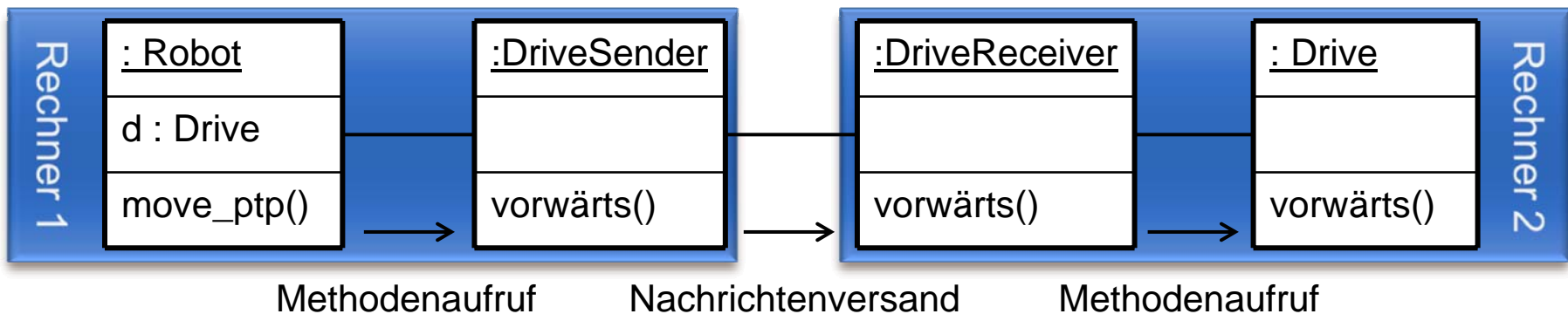
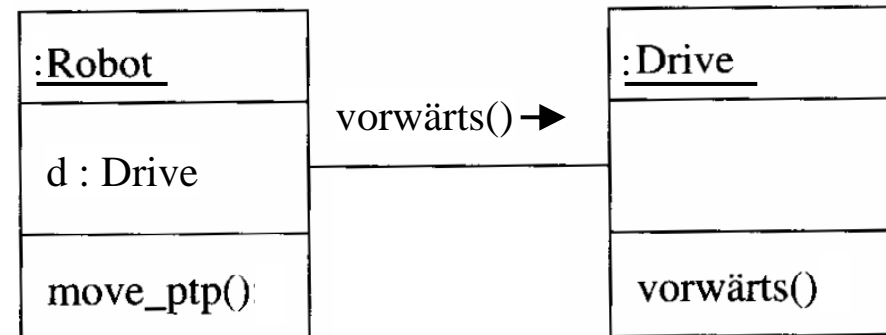
- Nachricht wird von Sender zu Empfänger geschickt
- **Asynchrone** Kommunikation
- Kann auch zwischen Objekten auf verschiedenen Rechnern geschehen

## Kollaborationsdiagramme

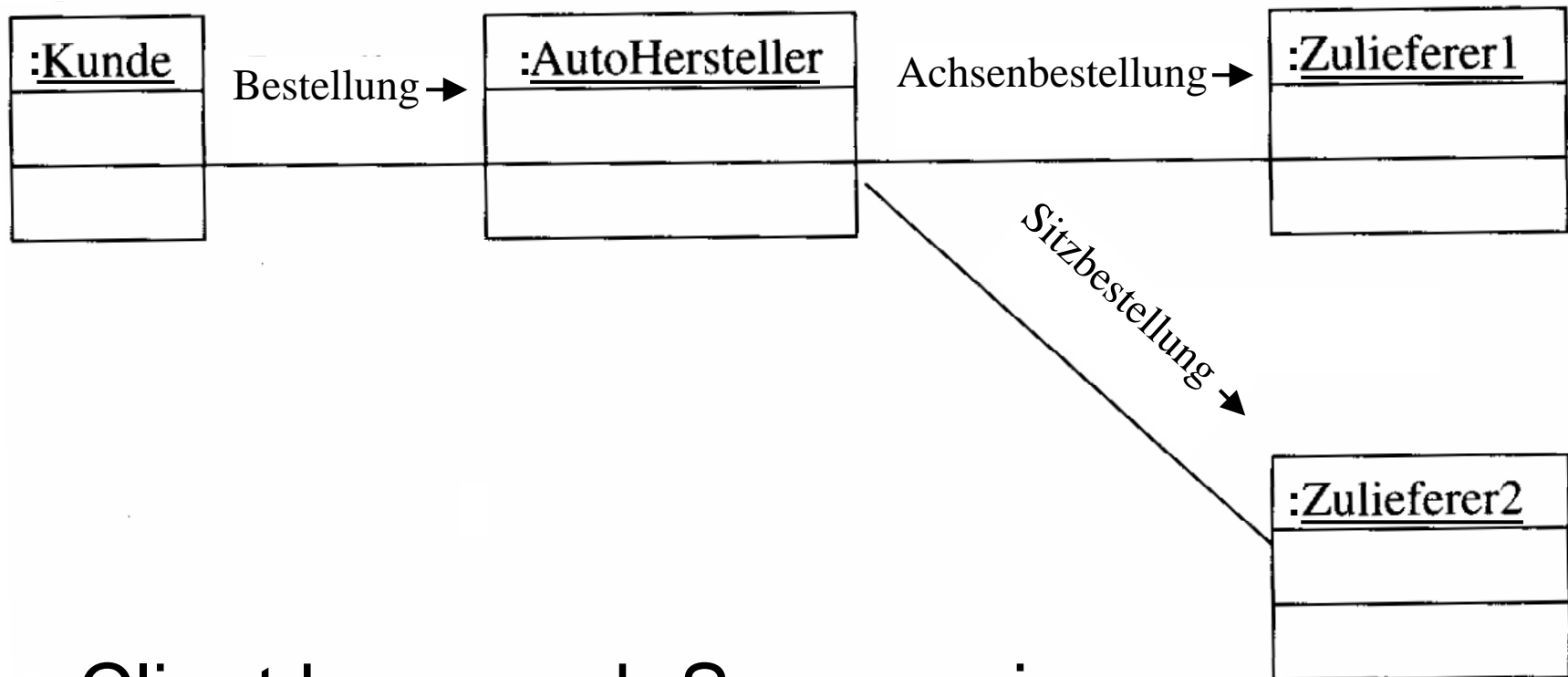


# Client/Server-Beziehung

- Ein Objekt (Client) fordert Dienstleistung von anderem Objekt (Server) an
- **Synchrone** Kommunikation
- **Methodenaufruf**
  - Standard in Java
- Kann nicht unmittelbar auf verschiedenen Rechnern geschehen
  - Ist aber durch Stellvertreterobjekte möglich (entfernter Methodenaufruf, Remote Method Invocation)

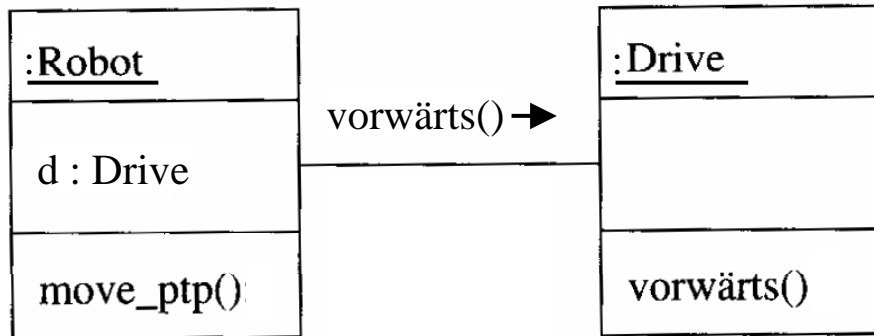


# Client/Server-Beziehung



- Client kann auch Server sein

# Methodenaufruf in Java

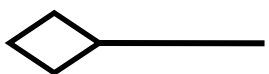
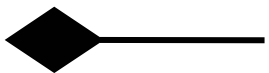


- Bitte wählt eure Bezeichner einheitlicher als die Autoren des Buches
- nur in einer Sprache (normalerweise Englisch)
  - `forward()`;
- keine Unterstriche (werden nur in Konstanten verwendet)
  - `movePTP()`;
  - `final int NUM_OF_TESTS = 12;`

```
class Robot {
    Drive d;
    :
    void move_ptp() {
        d.vorwärts();
    }
}
```

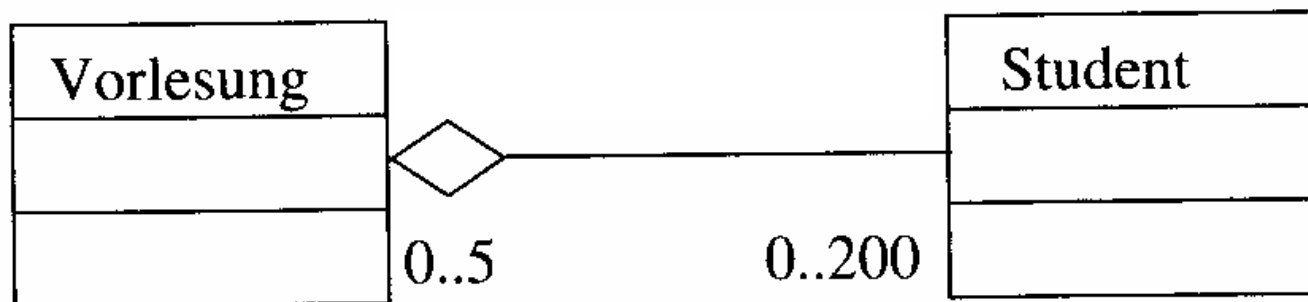
```
class Drive {
    void vorwärts() {
        :
    }
}
```

## Einschlussbeziehungen (has-a)

- Objekte einer Klasse schließen Objekte einer anderen Klasse ein
- Ein Objekt „hat“ ein anderes Objekt (**has-a**-Beziehung)
- Zwei Arten
  - Aggregation 
  - Komposition 

# Einschlussbeziehungen (has-a) Aggregation

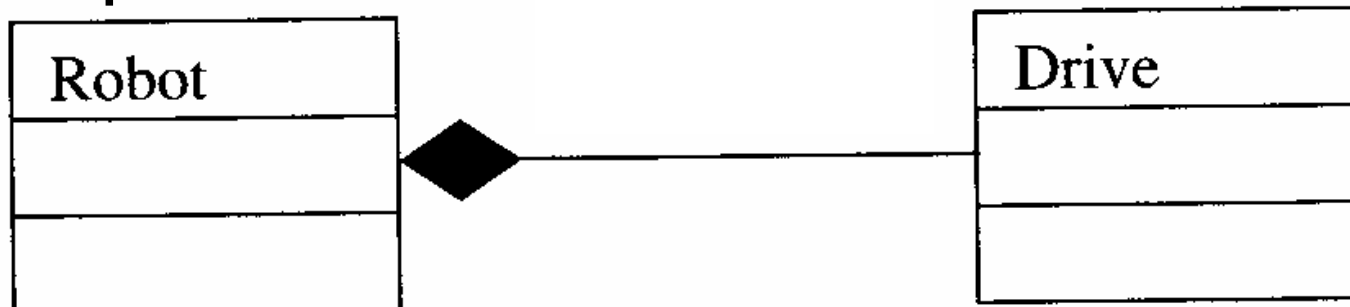
- Ein Objekt ist zu einem anderen zugehörig, aber nicht ausschließlich
- Beispiel: Eine Vorlesung hat bis zu 200 Studenten, die aber auch an anderen Vorlesungen teilnehmen



# Einschlussbeziehungen (has-a)

## Komposition

- Teilobjekt ist fester Bestandteil eines Objektes
  - Können die Teile ohne das Ganze existieren und das Ganze ohne seine Teile?
  - Wenn nein → Komposition
- Beispiel: Ein Roboter hat einen Antrieb

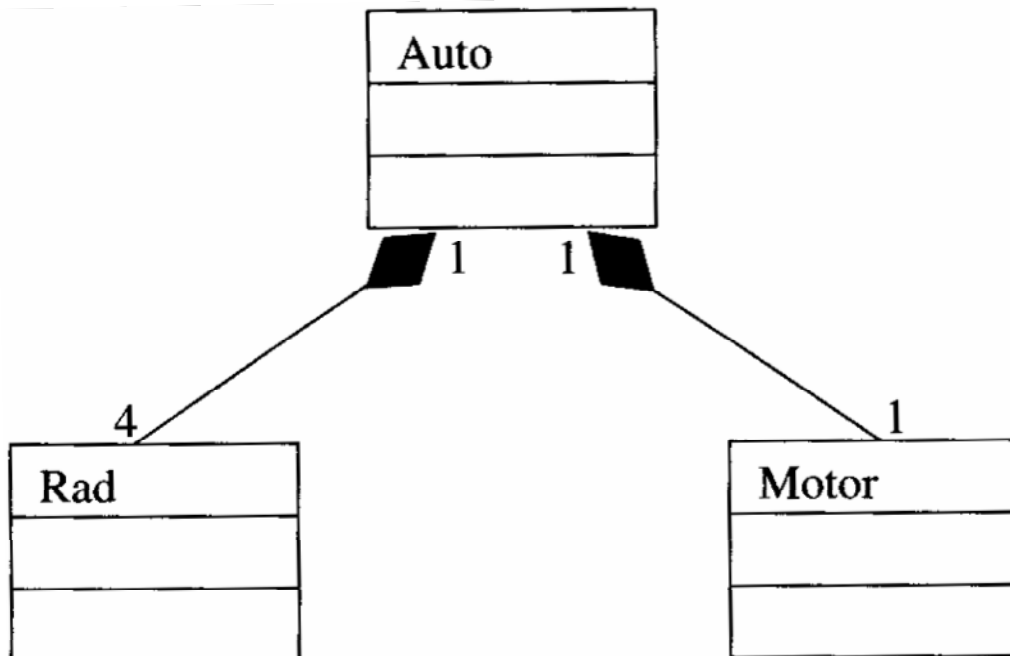




# Einschlussbeziehungen (has-a)

## Beispiel

- Ein Auto hat vier Räder und einen Motor



```
class Auto {
    Rad vr, vl, hr, hl;
    Motor m;
    :
    void starte() {
        m.ein();
    }
}
```

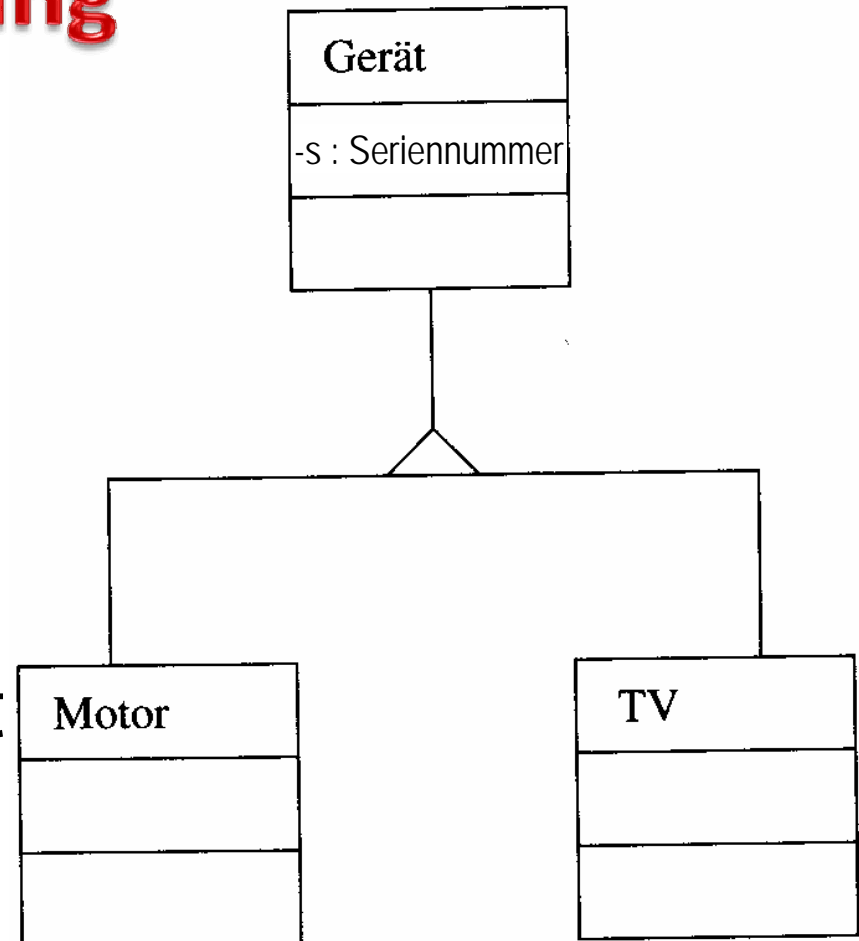
# Subtyp bzw. Vererbungsbeziehung (is-a)

- Eine Klasse besitzt alle Eigenschaften einer anderen Klasse und darüber hinaus noch weitere
- Der Subtyp (subtype) ist (is-a) eine spezielle Abart des Obertyps (supertype)
- Der Subtyp ist eine Spezialisierung des Obertyps
- Der Obertyp ist eine Verallgemeinerung des Subtyps

# Subtyp bzw. Vererbungsbeziehung

## Nutzung der Beziehung

- Subtyp erbt vom Obertyp
- Die gemeinsamen Eigenschaften müssen nur einmal im Obertyp implementiert werden



# Subtyp bzw. Vererbungsbez. (is-a)

## Beispiel

- Kaffeemaschine Cafe2000 hat eine Funktion zur Befüllung von Kaffee und Wasser von bis zu 12 Tassen
- Kaffeemaschine Cafe2000LT hat zusätzlich eine Timerfunktion



```
class Cafe2000 {
    int tasse;
    void befüllung(int x) {
        tasse = x;
    }
}
```

```
class Cafe2000LT extends Cafe2000 {
    Time t;
    void timer(Time time) {
        t = time;
    }
}
```