# System-Design

- Übersicht
- Klassen und ihre Elemente
- Beziehungen zwischen Klassen
  - Assoziationen
  - Aggregation vs. Komposition
  - Vererbung
- Interfaces
- Komponenten
- Vorgehen beim Design (einfache Heuristik)

DHBW Duale Hochschule Baden-Württemberg Mannheim

# Die Unified Modeling Language (UML)

- Die UML ist eine *graphische Notation* für die objektorientierte Entwicklung (Analyse, Entwurf, Implementierung)
- Graphische Notationen sind geeignet
  - um sich einen (abstrakten) Überblick über den Aufbau eines Anwendungsgebiets oder eines Programms zu verschaffen
  - um mit Kunden, Anwendern usw. zu kommunizieren
  - zum "drüber reden" unter Informatikern, für die Diskussion alternativer Implementierungsmöglichkeiten
  - etc.
- Diese Aktivitäten sind z.B. auf (Java-) Codeebene und mit Pseudocode nicht möglich!

# Die Unified Modeling Language (UML)

Die UML bietet viele Diagrammarten

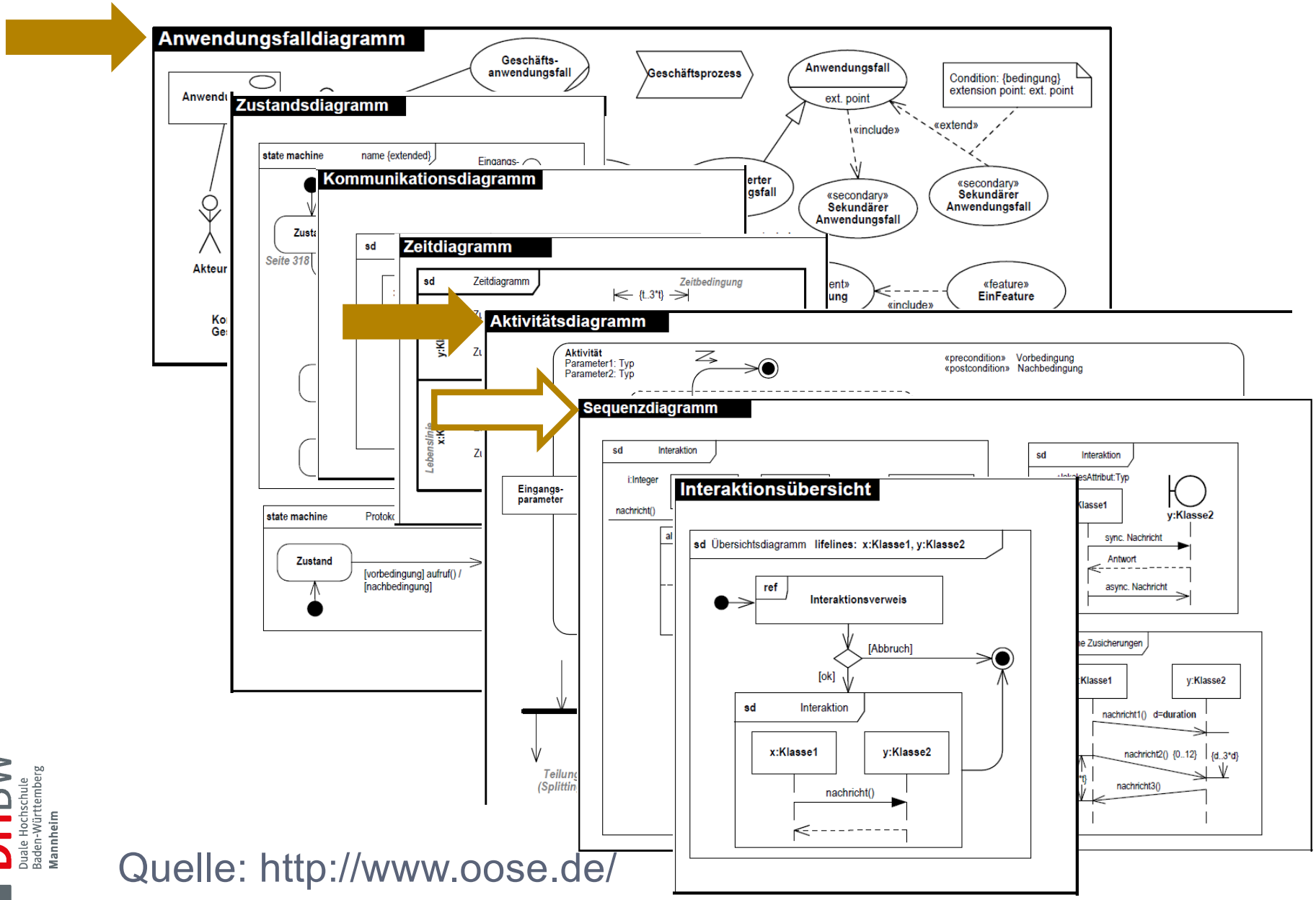
- für die objektorientierte Analyse; darstellen lassen sich
  - Abläufe (hier besonders wichtig)
  - Strukturen

  des zukünftigen Systems
- für das objektorientierte Design; darstellen lassen sich
  - Abläufe
  - Strukturen

  des zukünftigen Systems

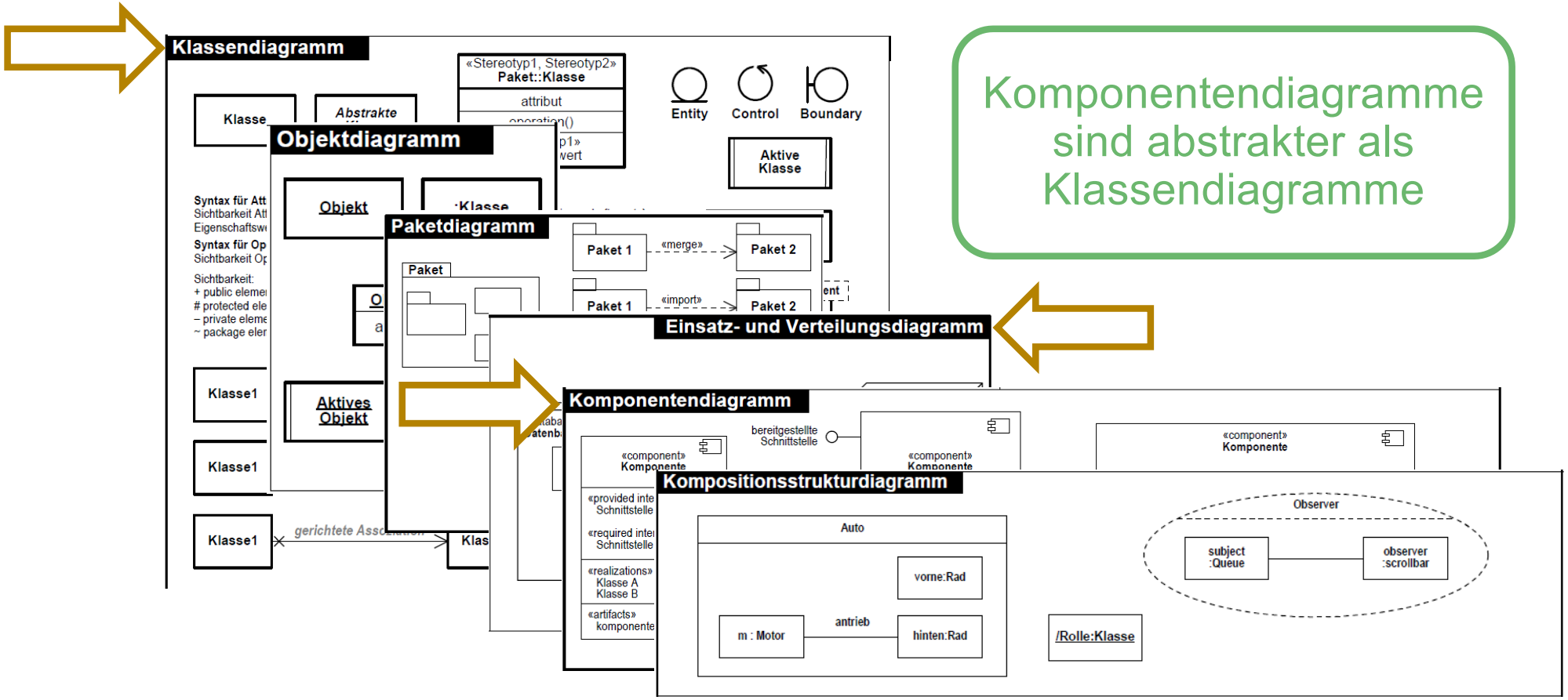
DHBW Duale Hochschule Baden-Württemberg Mannheim

# UML-Diagrammtypen: Abläufe




Quelle: http://www.oose.de/

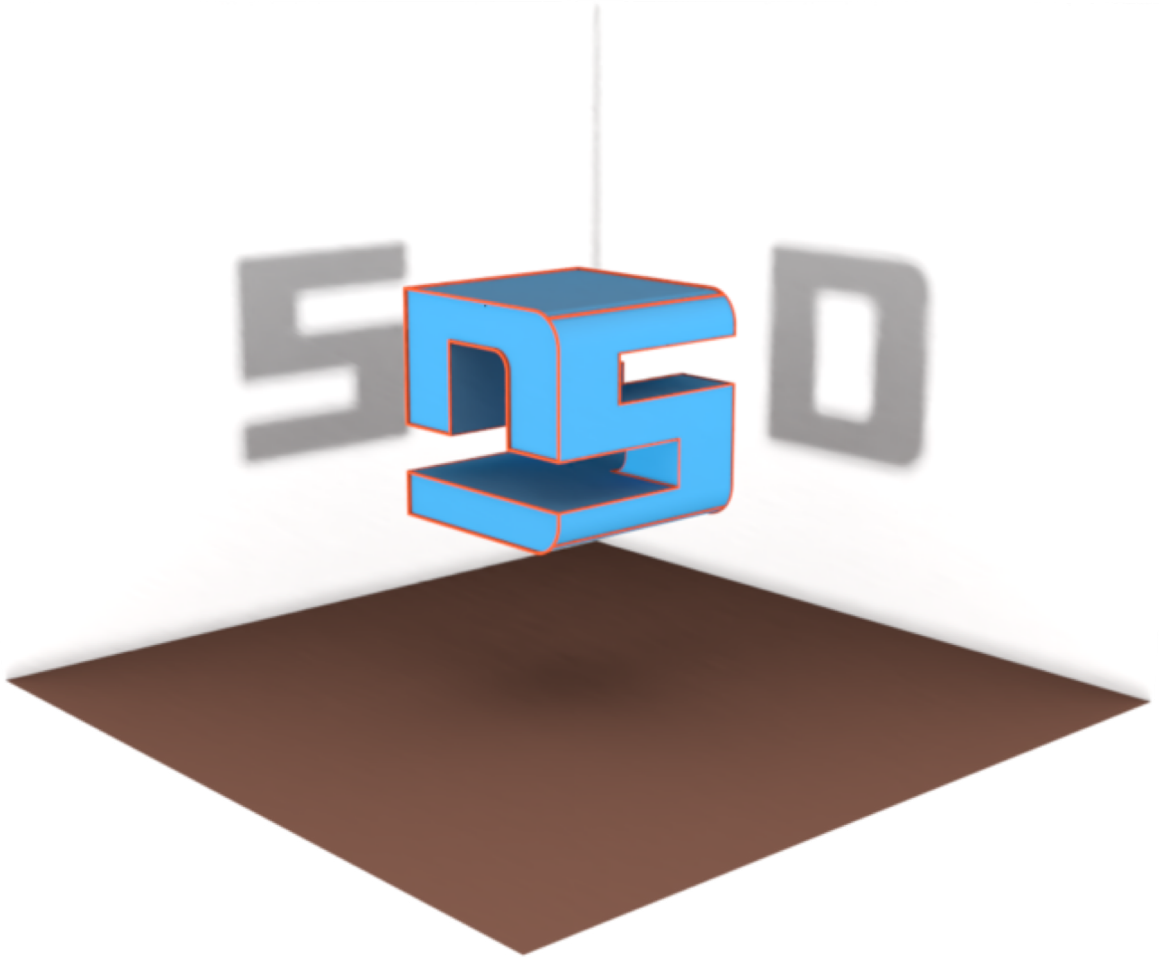
# UML-Diagrammtypen: Strukturen




Komponentendiagramme sind abstrakter als Klassendiagramme

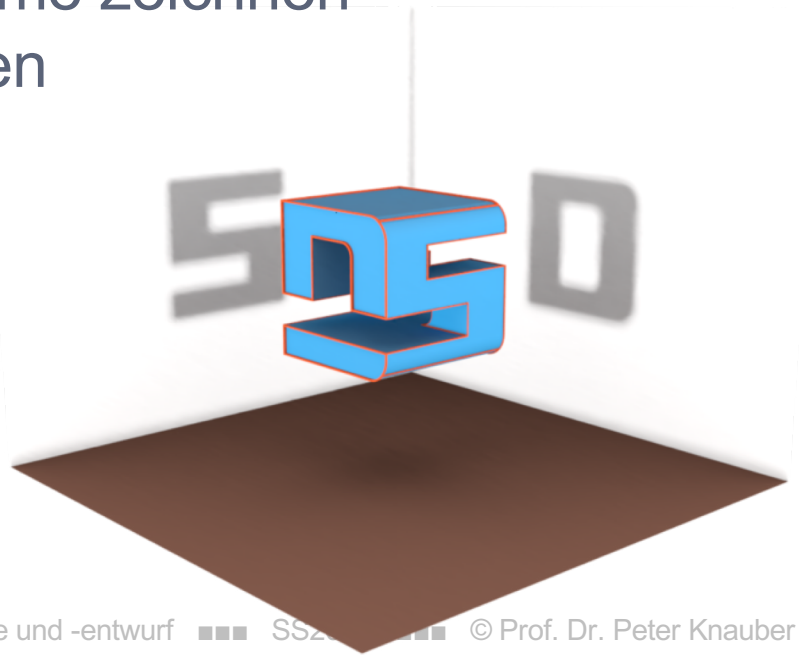
Quelle: http://www.oose.de/

# Statische und dynamische Sicht: die selbe Architektur aus mehreren Perspektiven



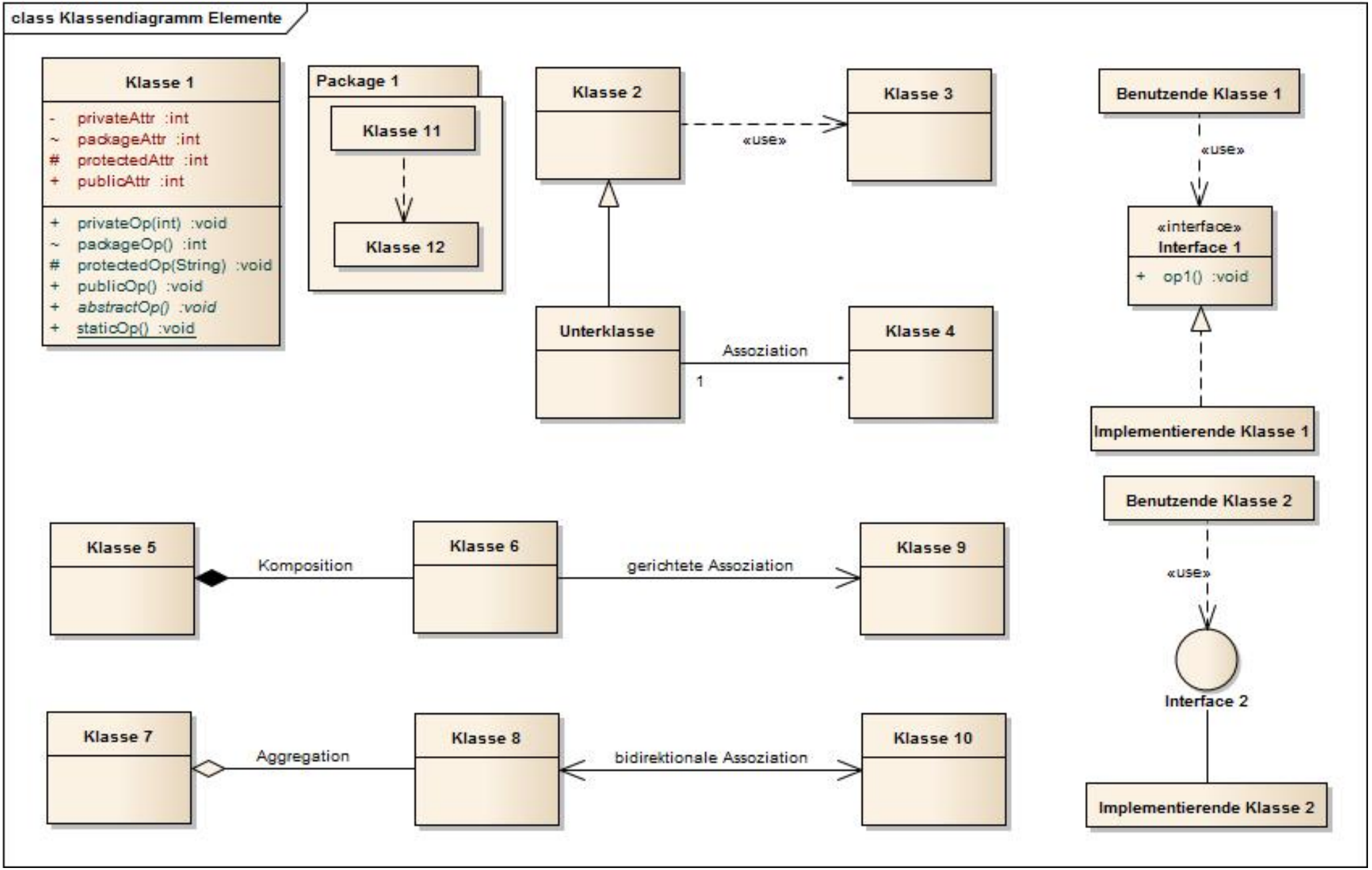
# Vorgehen beim Design (OOD)

1. Wir überlegen uns die wichtigsten / offensichtlichen Klassen, die wir benötigen
   -> als (ein) Klassendiagramm zeichnen
2. Wir überlegen uns, wie die wichtigsten / schwierigsten Aufgaben zwischen diesen Klassen abgewickelt werden; der Fokus liegt auf der *Interaktion*, nicht den Algorithmen
   → als (mehrere) Sequenzdiagramme zeichnen
   → die dafür benötigten Operationen in das Klassendiagramm übernehmen



# F R A G E N

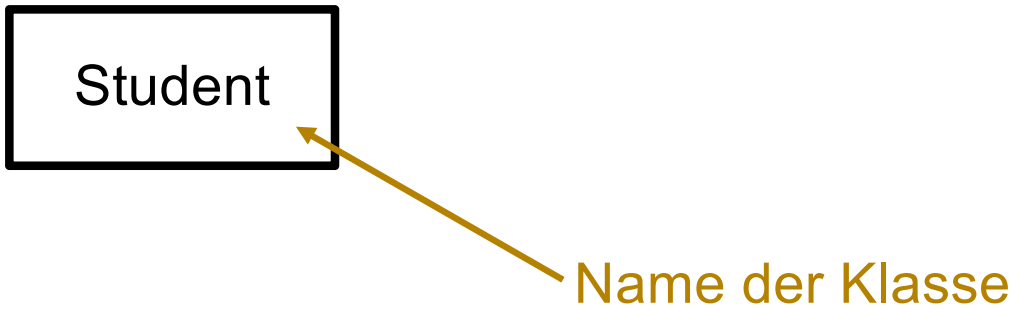
# Elemente von Klassendiagrammen



# Klassen und Objekte

*Klassen*...

- repräsentieren den Bauplan (Blaupause, Template, Schablone, Vorlage, ...) für eine Menge gleicher ***Objekte***;
  Beispiele
  - Bankkonten, Autos (einer Marke), Studenten
- werden in der UML als Rechteck repräsentiert

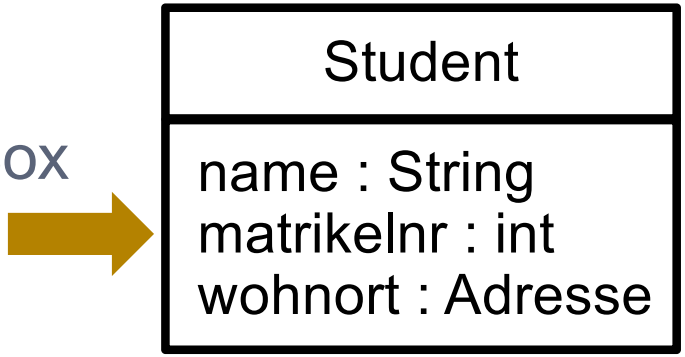



*Objekte* (Instanzen) sind die konkreten Ausprägungen von jeweils einer Klasse

# Attribute

*Attribute*...

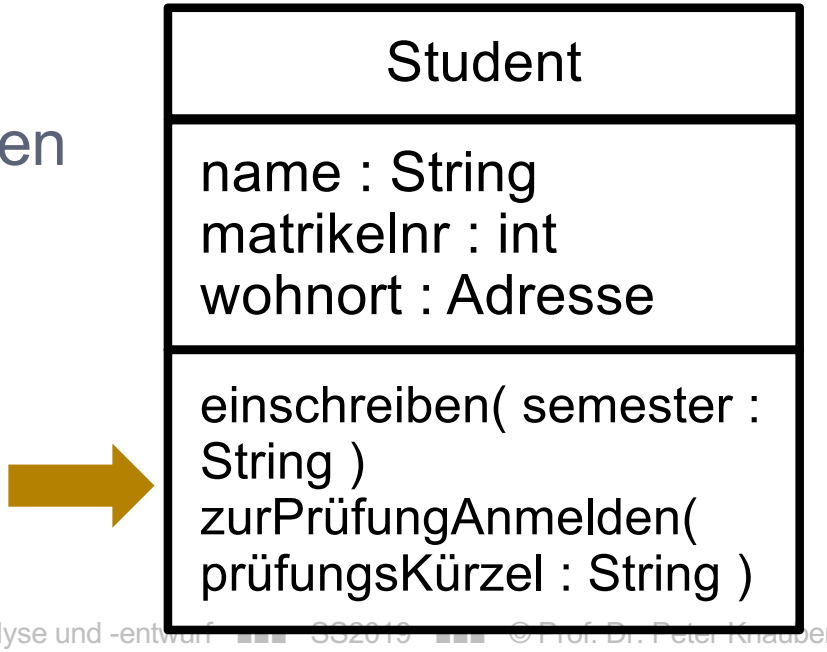
- sind die Eigenschaften einer Klasse
  - Jedes Objekt einer Klasse besitzt *alle* Attribute der Klasse (nicht mehr, aber auch nicht weniger)
  - Objekte einer Klasse enthalten einen Wert für jedes Attribut
  - Alle Attributwerte zusammen stellen den *Zustand* des Objekts dar
- haben einen Typ, der bestimmt, welche Werte sie annehmen können
- können einen Default-Wert haben
- werden in der UML in einer separaten Box unter dem Klassennamen dargestellt



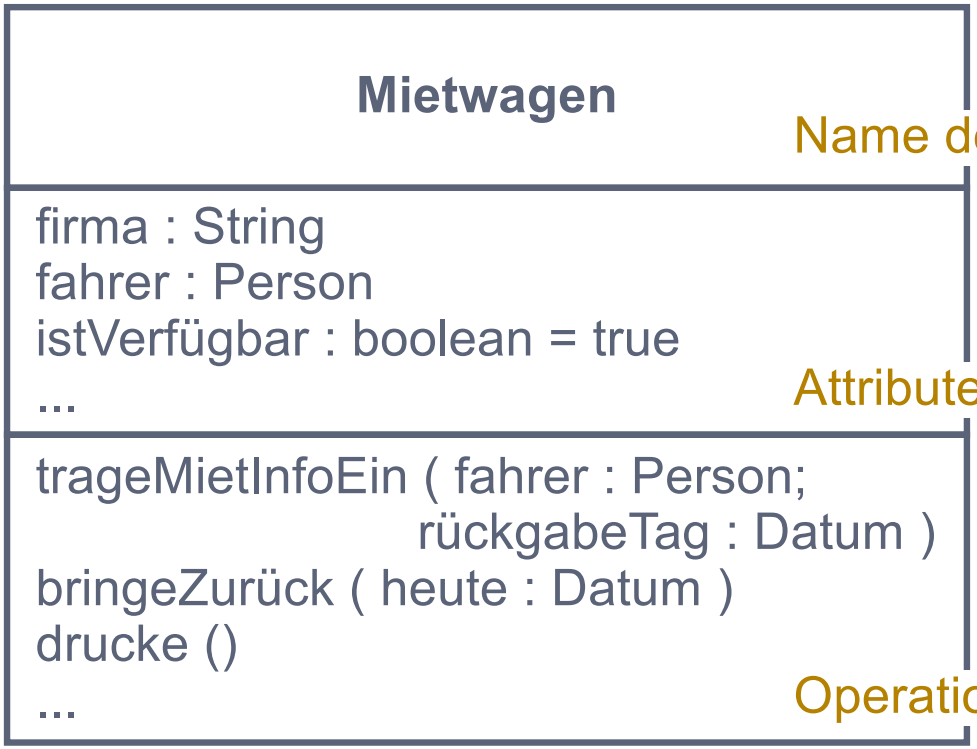
# Operationen

*Operationen*...

- dienen dazu, den Zustand von Objekten abzufragen oder zu verändern; Beispiele
  - (Student) immatrikulieren, Fläche (einer geometrischen Figur) berechnen, (Motor) einschalten
- heißen in Programmiersprachen zum Beispiel Methode, (virtuelle) Funktion, ...
- werden in der UML in einer separaten Box unter den Attributen dargestellt



# Klassen in der UML: Beispiel




Klassennamen beginnen mit einem **G**roßbuchstaben, weil...

Klassen repräsentieren Dinge aus der realen Welt und werden daher mit Hauptwörtern benannt

# Sichtbarkeit von Attributen und Operationen

Spezielle Markierungen werden benutzt, um die Sichtbarkeit von Attributen und Operationen anzugeben

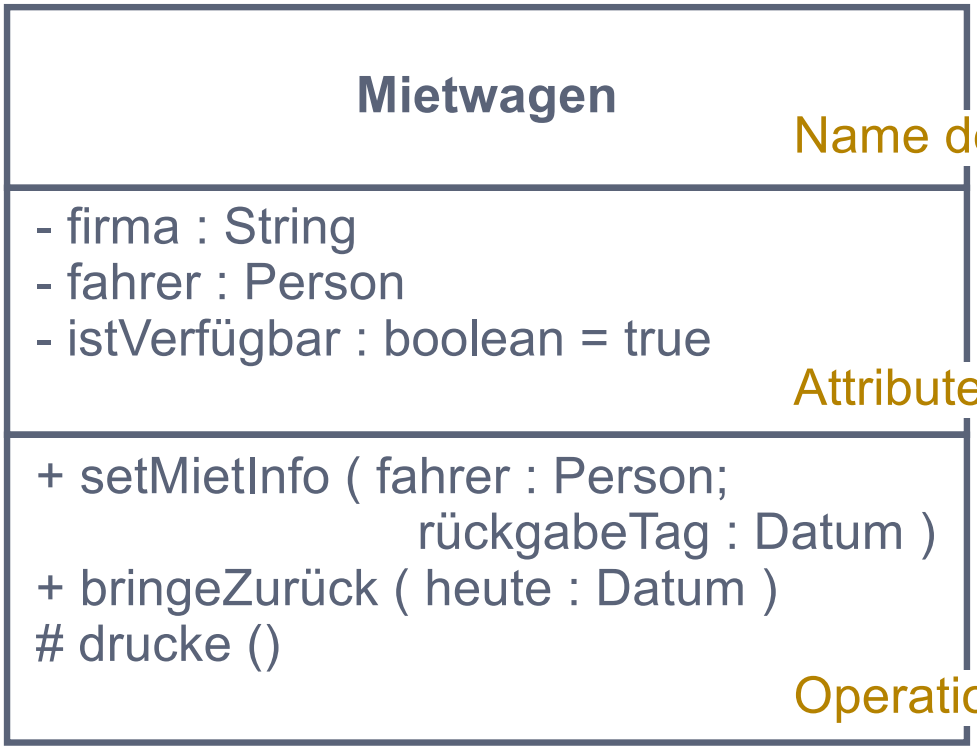
\+ öffentlich (*public*): von überall in einem Programm sichtbar / zu benutzen
-> das ist gefährlich!

\- private (*private*): nur innerhalb seiner Klasse sichtbar
-> Das ist für (alle) Attribute zu empfehlen!

\# geschützt (*protected*): nur innerhalb seiner Klasse und deren Erben (s. später) sichtbar

~ ... (*package*): nur innerhalb seiner Klasse und in deren Paket sichtbar

# Klassen in der UML: Beispiel mit Sichtbarkeitskennzeichen




Klassennamen beginnen mit einem **G**roßbuchstaben

# Darstellung von Klassen in der UML

Klassendiagramm

vollständig:


Kurzformen


Das entspricht der Darstellung einer „Komponente“

Attribut: *attributname* ":" *Attributtyp*
*attributname* ":" *Attributtyp* = *Attributwert*

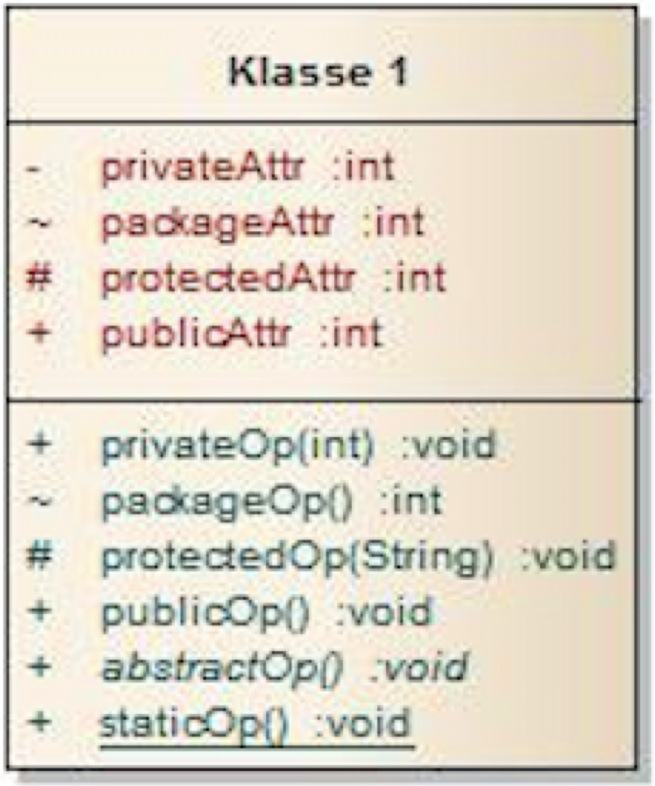
Operation: *operationsname* "(" … ")“ : Ergebnistyp

Konvention:

- Klassennamen beginnen mit Großbuchstaben
- Attributnamen und Operationsnamen beginnen mit kleinbuchstaben
- Es werden keine Trenn- oder Unterstriche verwendet

Java hat diese Konventionen von der UML übernommen

# Klassen




## Klassen haben...

- Attribute mit...
  - Sichtbarkeit
  - Namen
  - Typ
  - optional: Initialwert

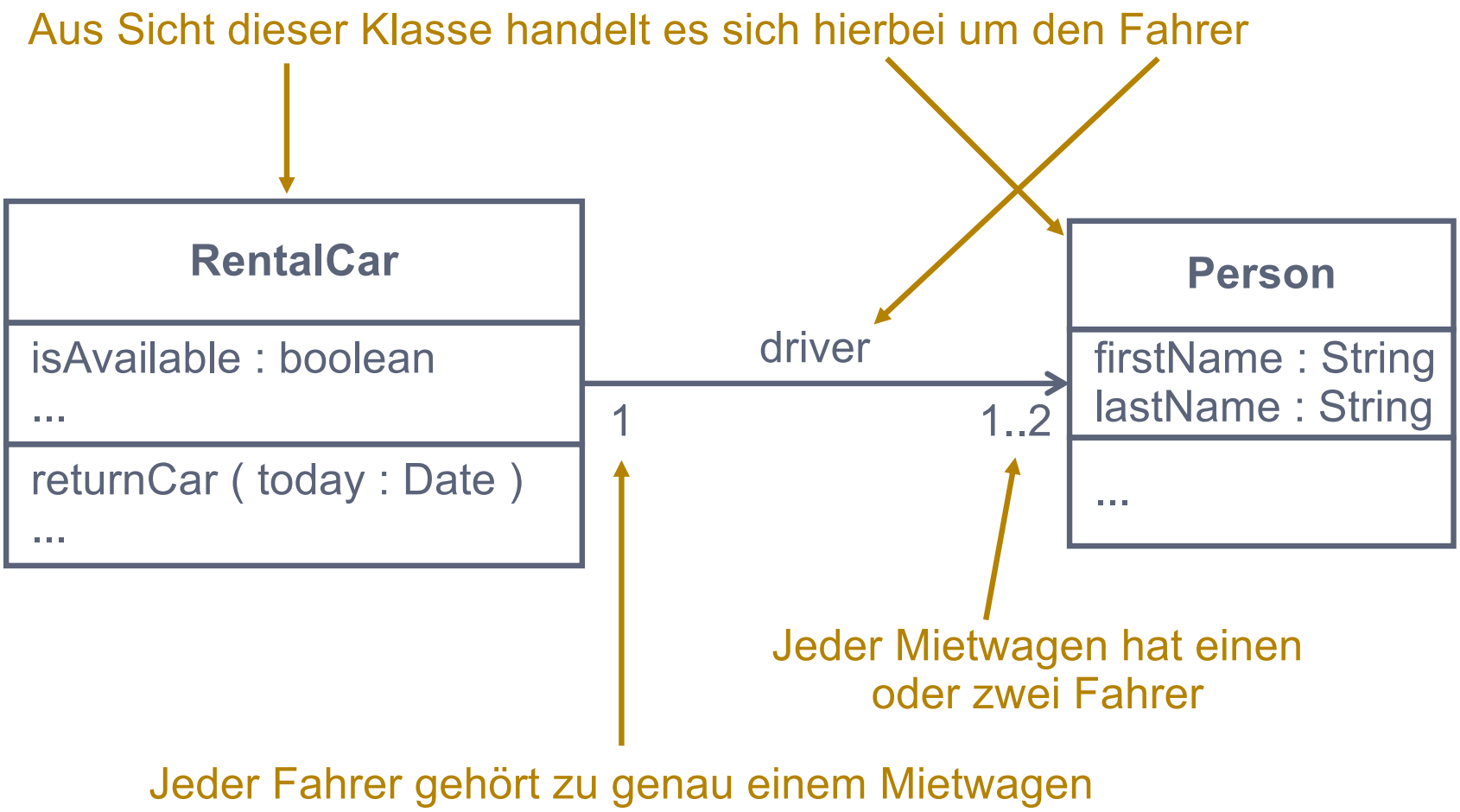
  Attribute können statisch sein
- Operationen (Methoden) mit...
  - Sichtbarkeit
  - Namen
  - Formaler Parameterliste
  - Ergebnistyp

  Operationen können abstrakt sein
  - Die zugehörige Klasse kann nicht instanziiert werden!

  Operationen können statisch sein

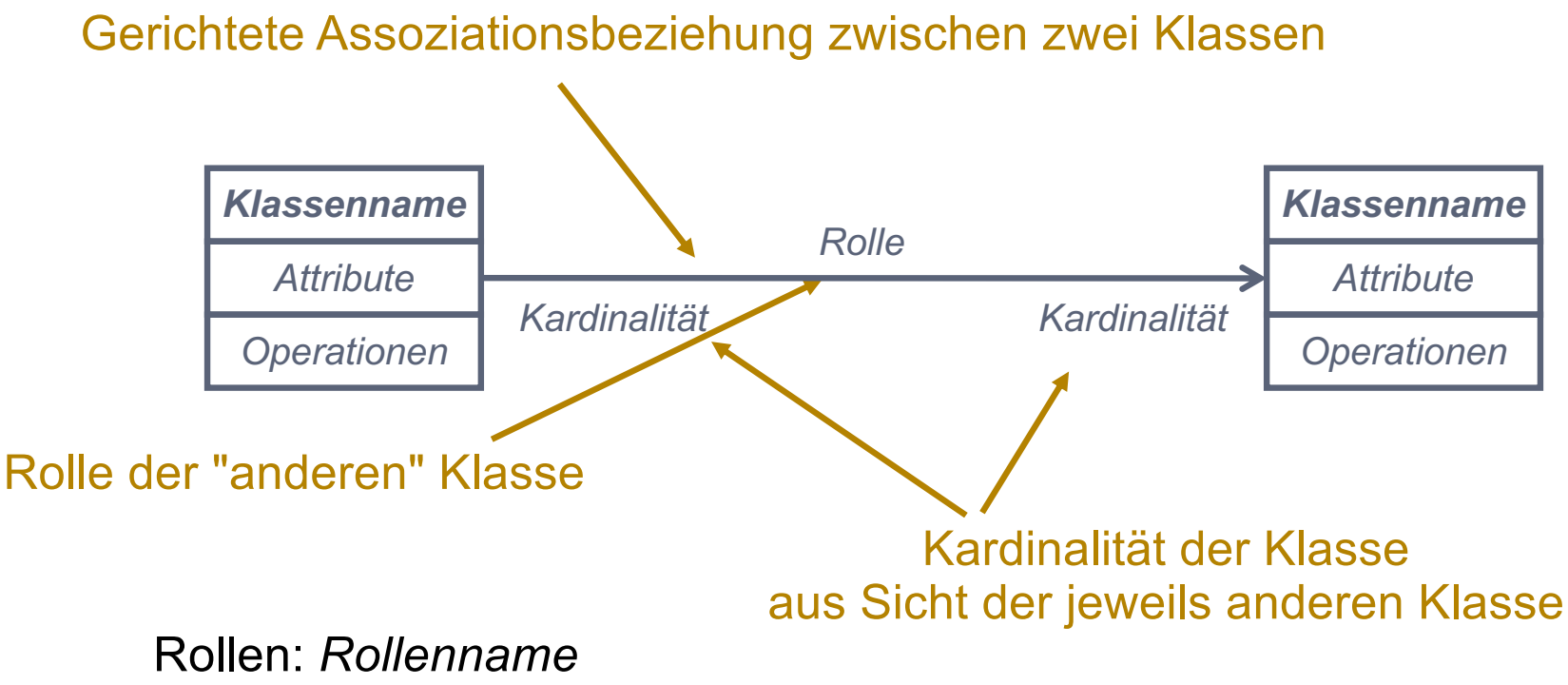
„protected“ ist anders als z.B. in Java definiert!

# Beziehungen zwischen Klassen in der UML: Beispiel




Können Sie sich den (Java-)Code dazu vorstellen?

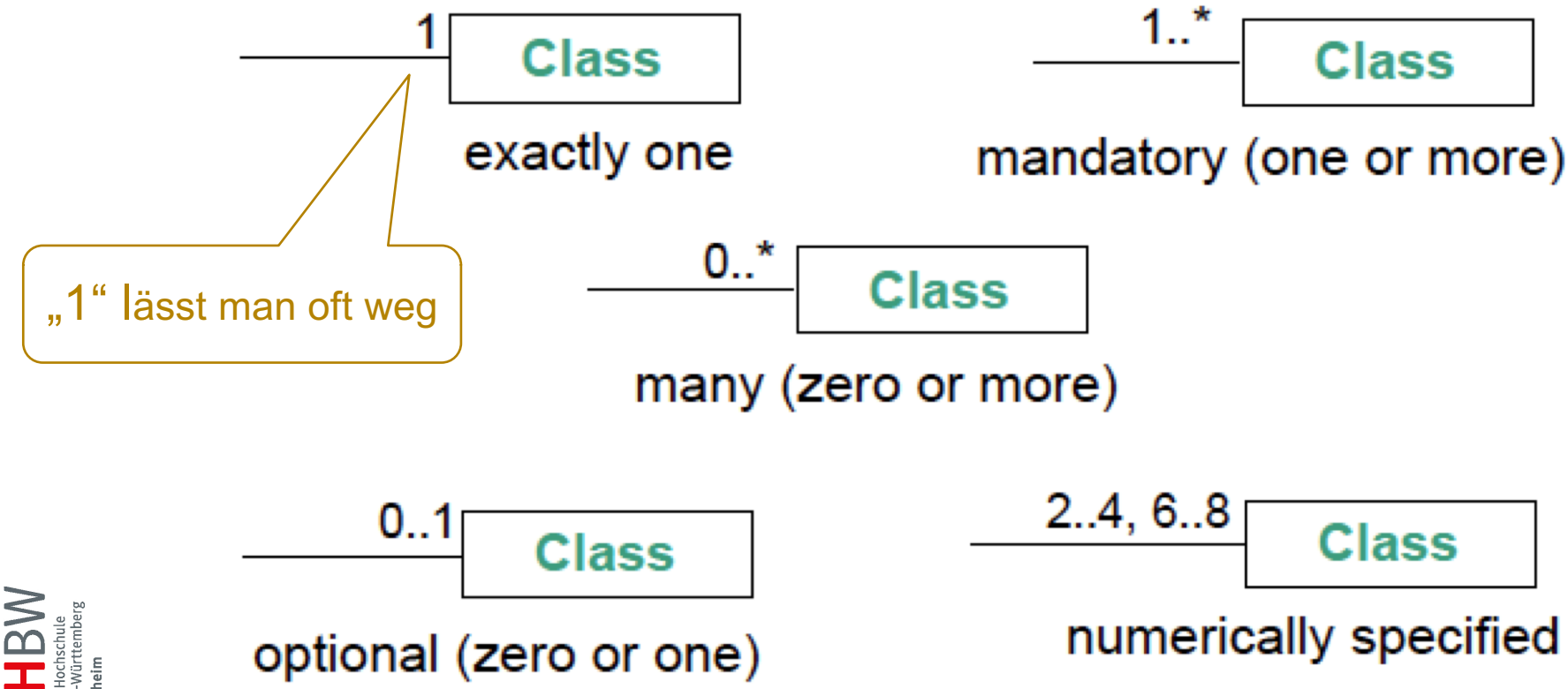
# Beziehungen zwischen Klassen in der UML



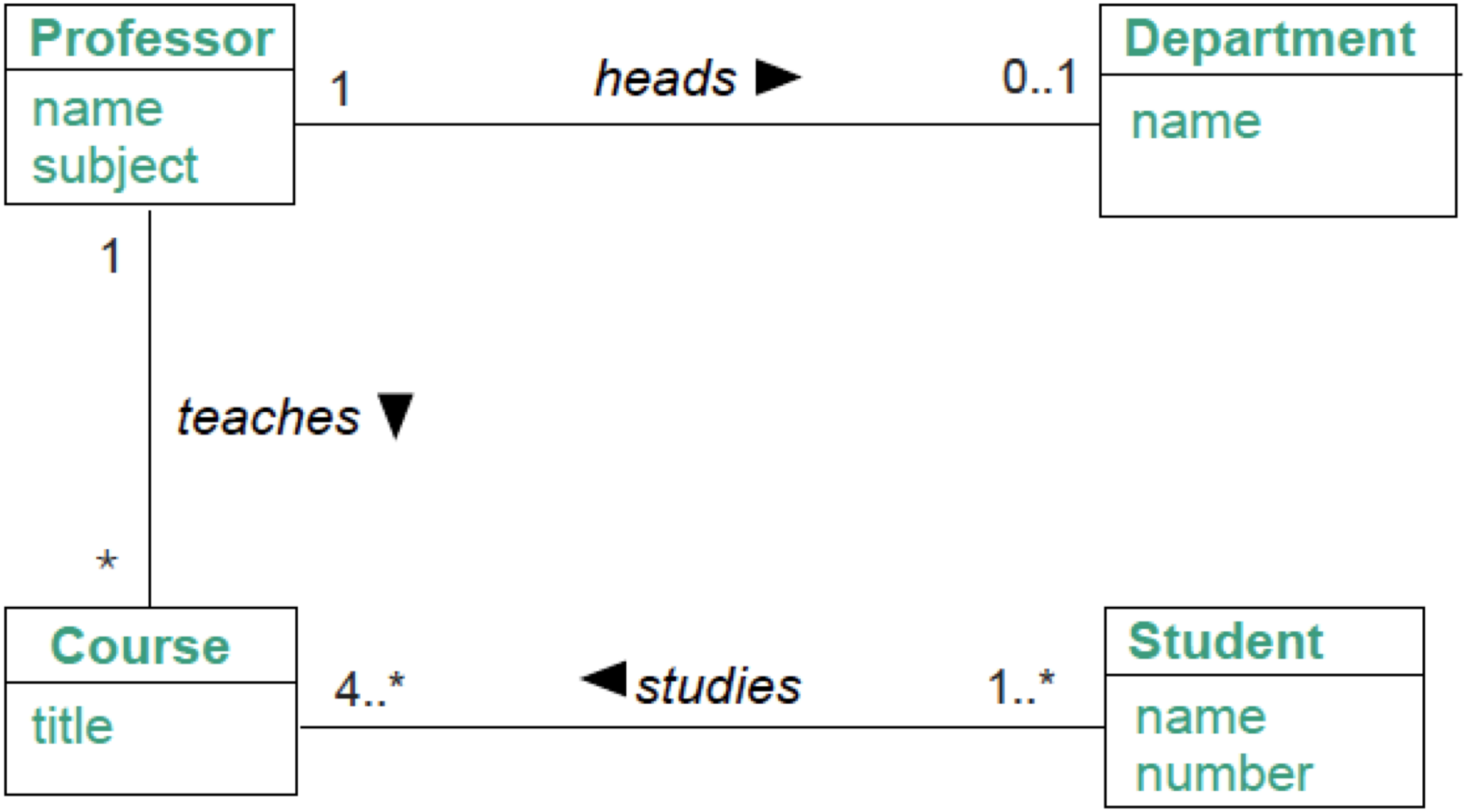

Rollen: *Rollenname*

# Kardinalität, (Häufigkeit, *multiplicity*)

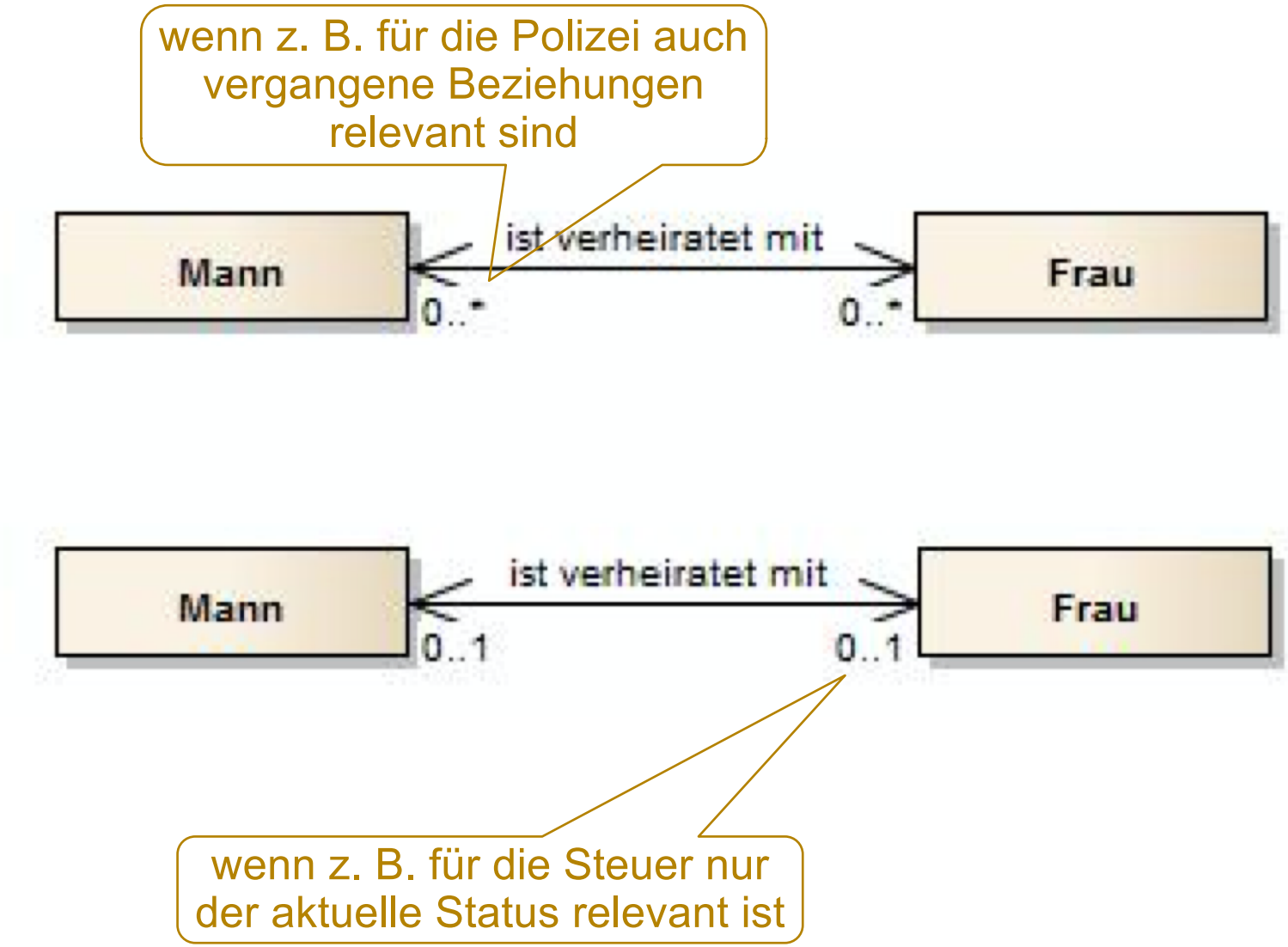
Die Kardinalität beschränkt die Anzahl der Instanzen einer Klasse, die zu einer Instanz einer anderen Klasse in Bezug stehen



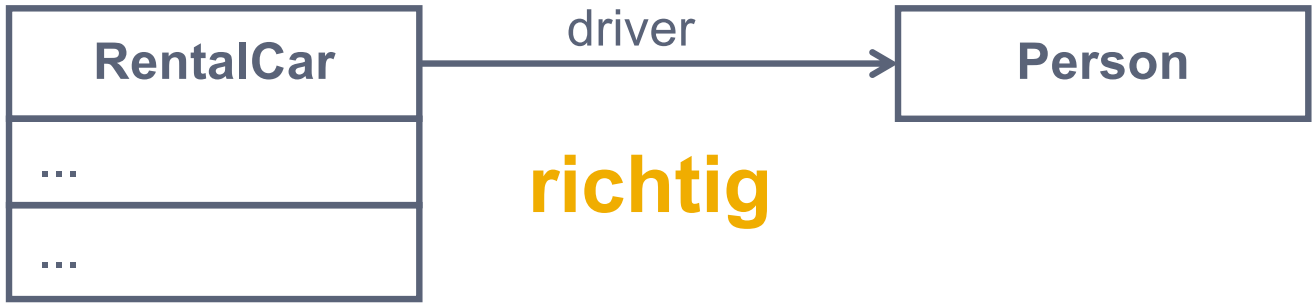
# Beispiel mit Kardinalitäten



# Kardinalität: Schnappschuss vs. Geschichte

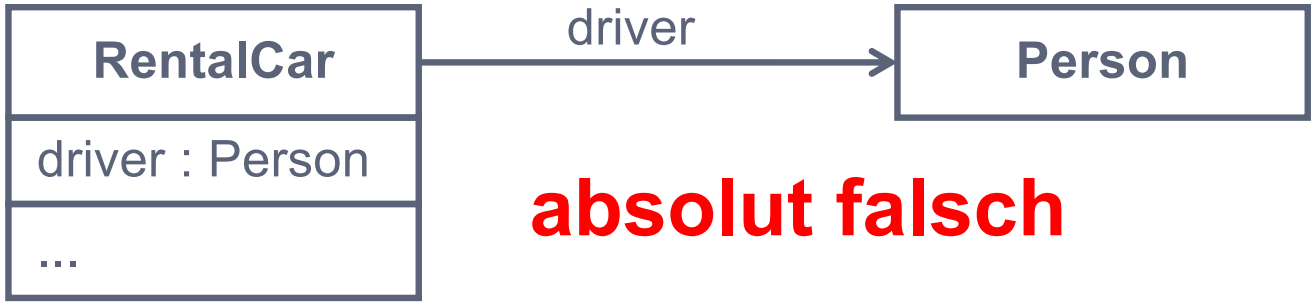


# *Zwei* gleichbedeutende Diagramme




**richtig**

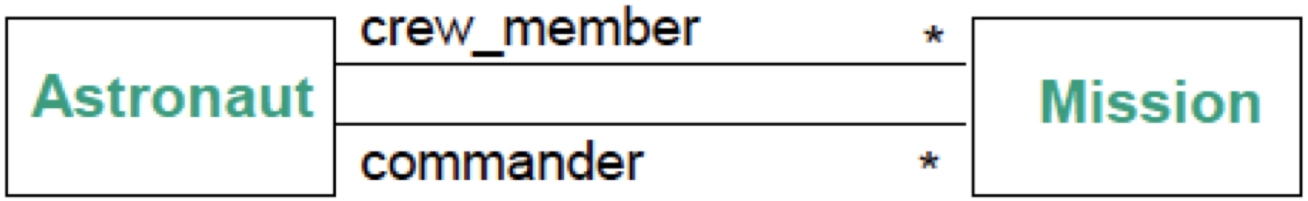

**ungeschickt**



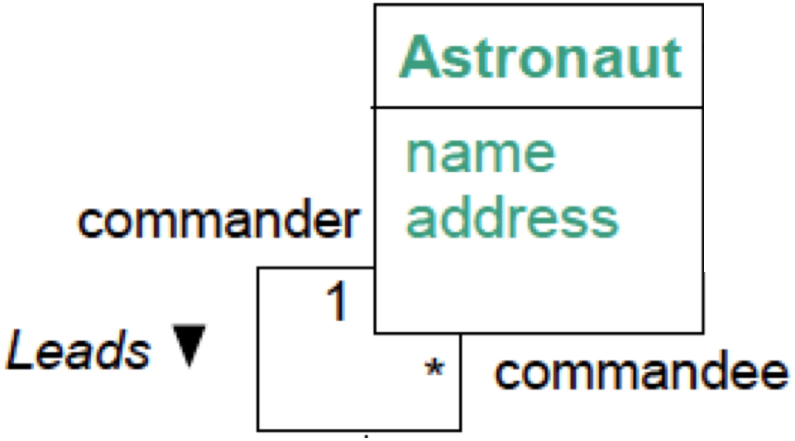

**absolut falsch**

# Sonderfälle

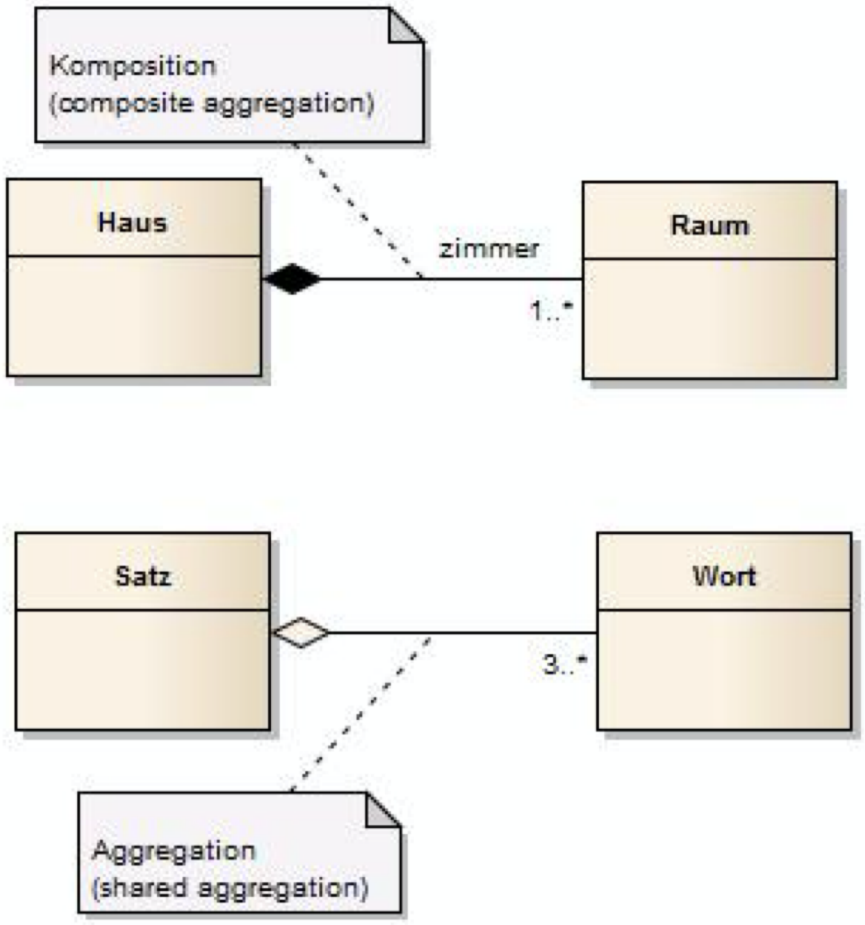
- Es kann mehr als eine Beziehung zwischen zwei Klassen existieren




- Eine Klasse kann mit sich selbst assoziiert sein



# Aggregation vs. Komposition




- Besondere Art der Assoziation: ein „Ganzes“ und seine „Teile“
- Auf der Seite des „Ganzen“ steht eine Raute (diamond)
- Ausgefüllte Raute
  - Die „Teile“ können *nicht* zu mehr als einem „Ganzen“ gehören
  - *Meistens* ist es nicht sinnvoll, dass „Teile“ alleine stehen
- Leere Raute
  - Die „Teile“ *können* zu mehr als einem „Ganzen“ gehören
  - Die „Teile“ können alleine stehen

DHBW Duale Hochschule Baden-Württemberg Mannheim

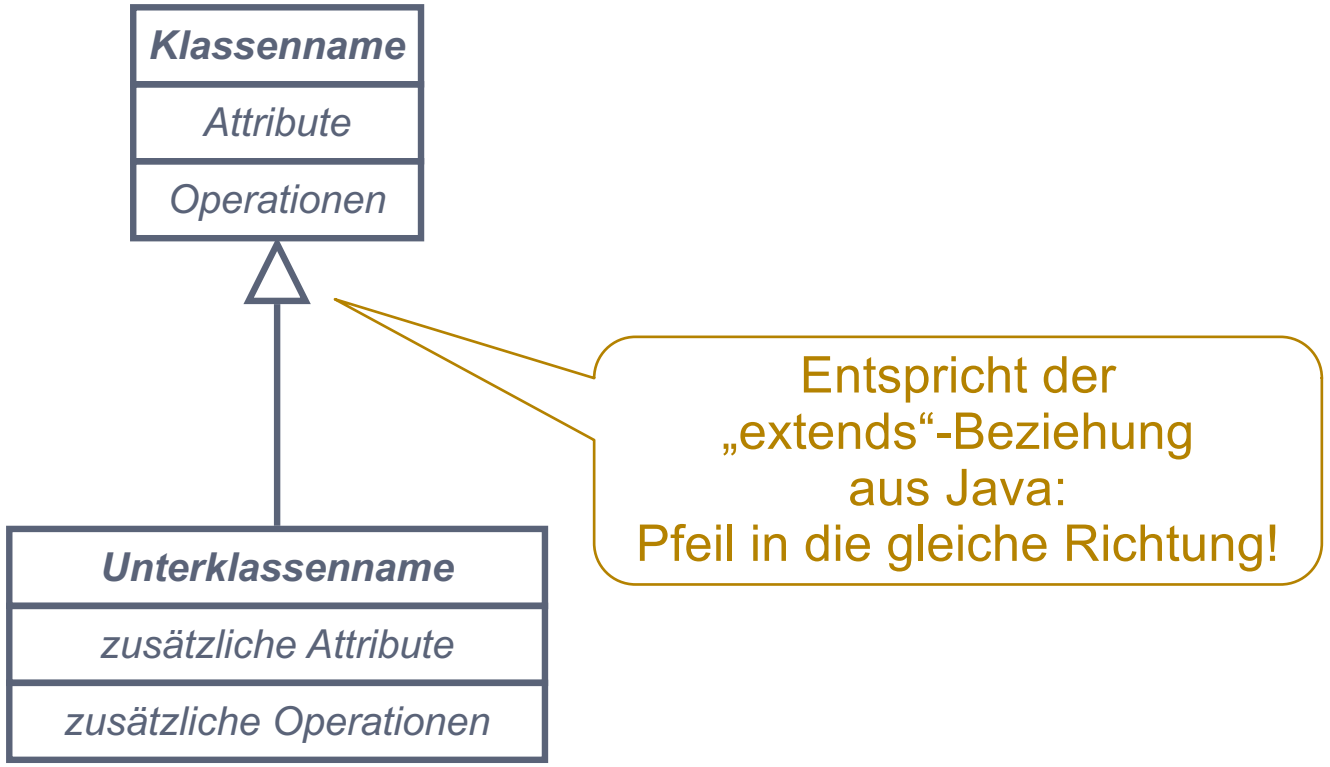
# Generalisierung / Spezialisierung

- Prinzip:
  Klassen werden anhand ihrer Gemeinsamkeiten und Unterschiede organisiert
  - Das entspricht der „Vererbung“ in objektorientierten Sprachen
  - Die Beziehung wird manchmal „is-a“-Beziehung genannt
  - Die Beziehung ist transitiv

- Notation in der UML:
  Ein Pfeil mit hohlem Kopf, der auf die allgemeinere Klasse zeigt

DHBW Duale Hochschule Baden-Württemberg Mannheim

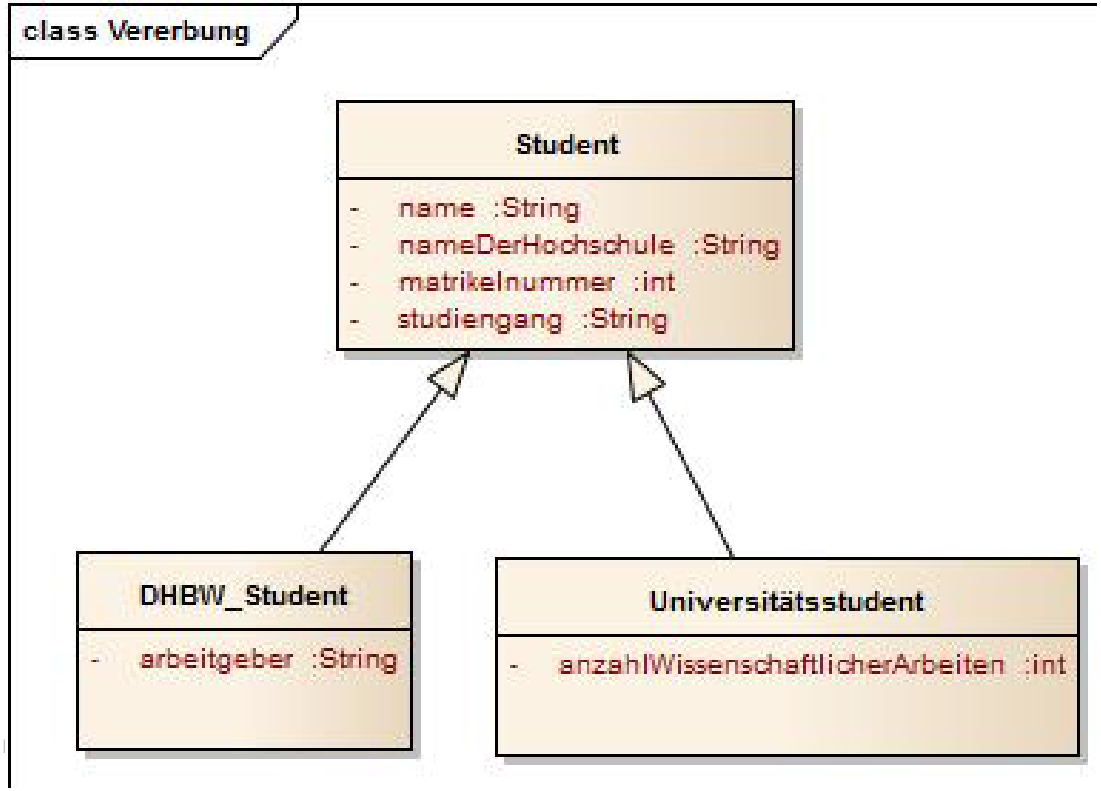
# Vererbung in der UML



# Unterklassen

Unterklassen...

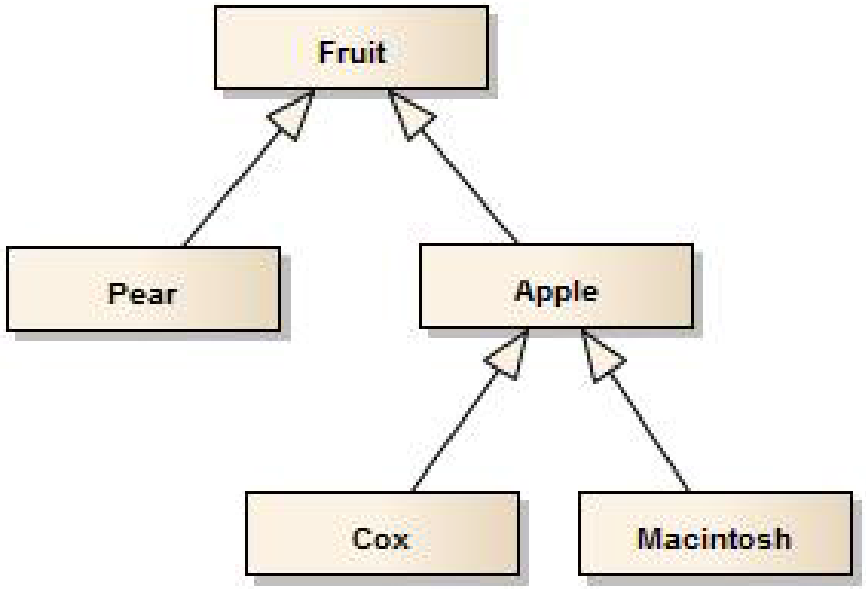
- erben die Attribute und die Operationen ihrer Oberklasse(n)
- werden so gewählt, dass ein Objekt einer Unterklasse überall dort einsetzbar ist, wo auch ein Objekt ihrer Oberklasse eingesetzt werden kann
- können die Implementierung einer geerbten Operation überschreiben



# Transitivität

Die Generalisierungsbeziehung ist transitiv,

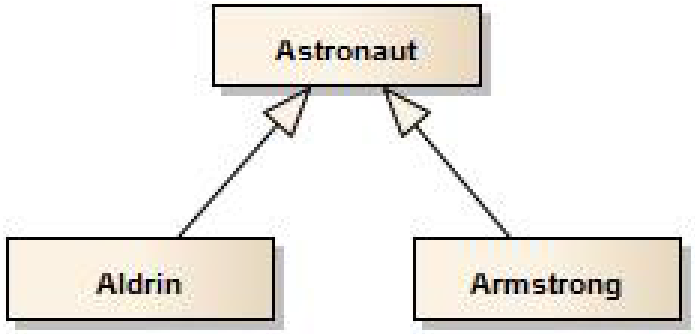
- d.h. ein Objekt einer Klasse ist immer auch ein Objekt *aller* ihrer Oberklassen,
- d.h. es besitzt *alle* Attribute *aller* Oberklassen und *alle* Operationen *aller* Oberklassen sind darauf anwendbar



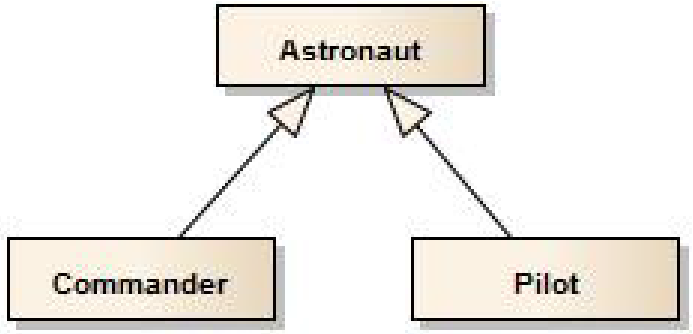
# Gern gemachter Fehler

Gerne werden Unterklassen eingeführt, die eigentlich Objekte sind

- Unterklassen sollten nur dann eingeführt werden, wenn sie eigene Attribute oder Operationen besitzen
- Warnsignal:
  eine Klasse, von der es nur ein einziges Objekt gibt




falsch

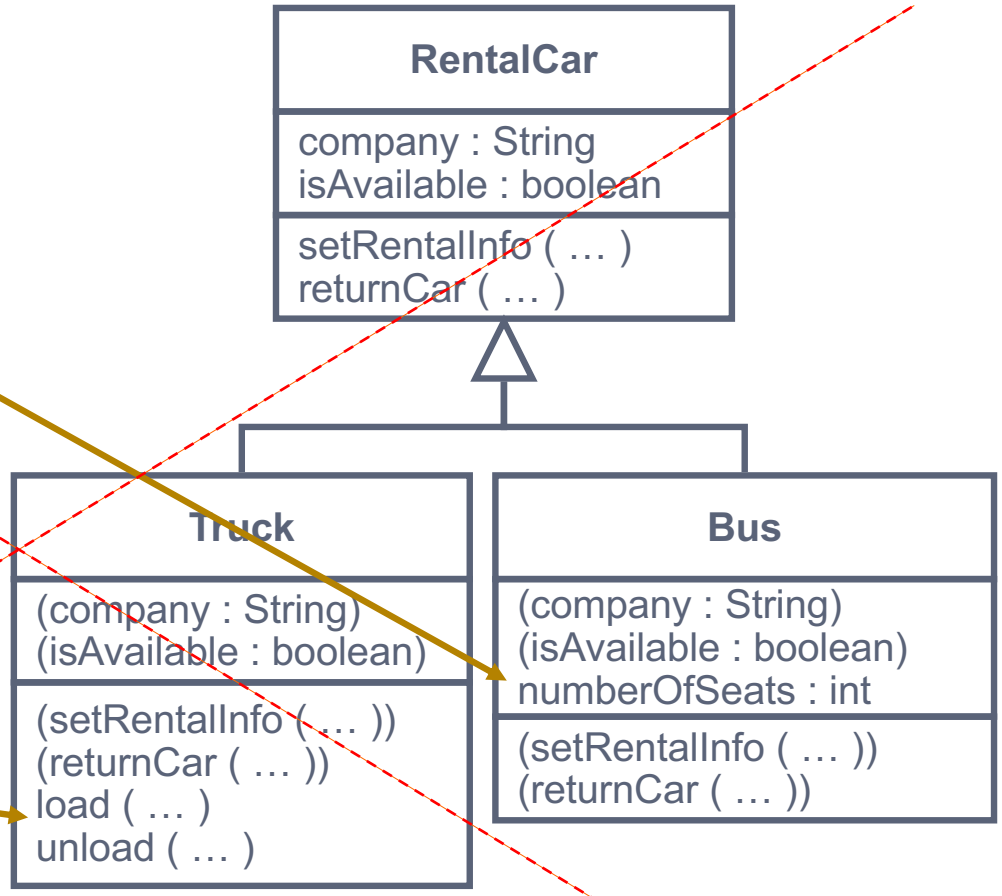



richtig

# Beispiel für Vererbung in der UML

- Ein Bus
  - ist ein (spezieller) Mietwagen,
  - der eine (große) Anzahl von Sitzplätzen bietet

- Ein Lkw
  - ist ein (spezieller) Mietwagen,
  - der beladen und entladen werden kann



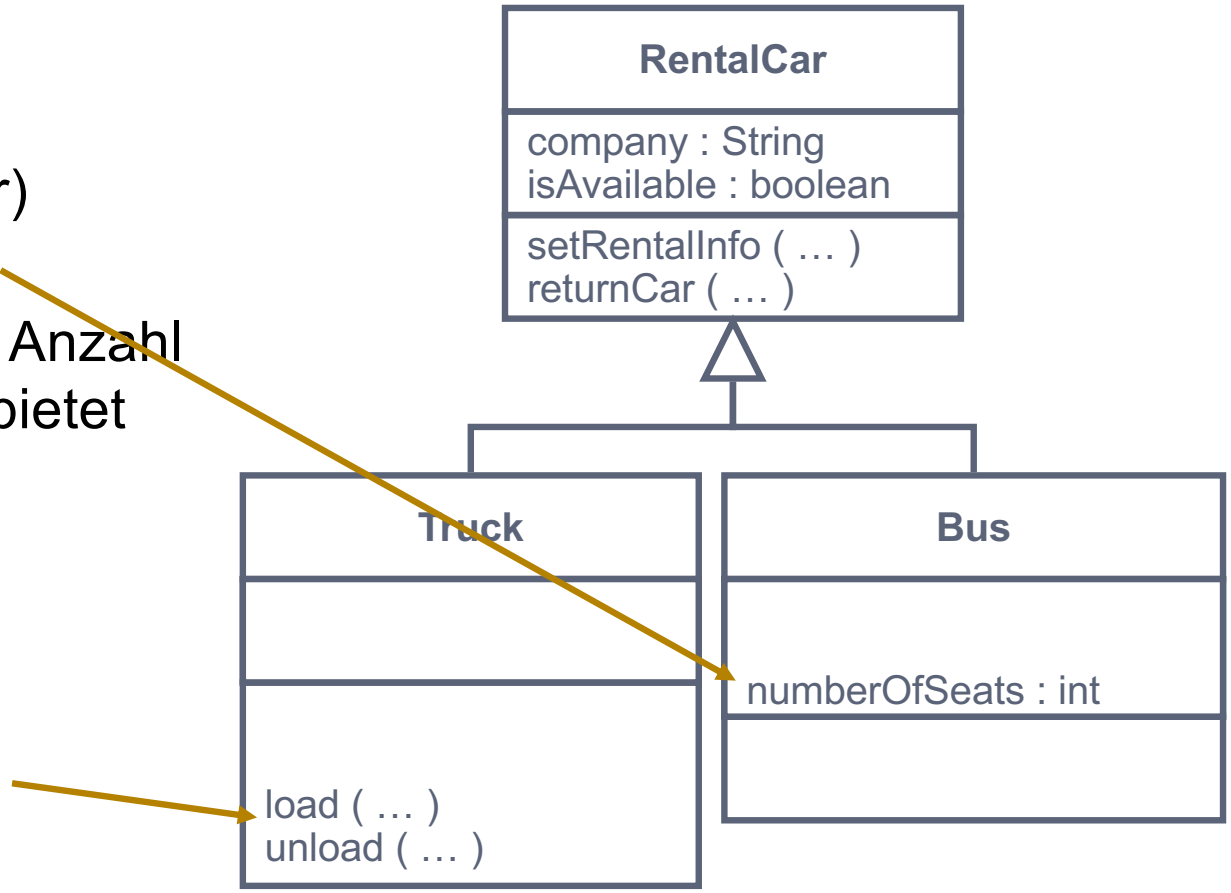

Ererbte Attribute und Operationen sind hier in Klammern gesetzt

Ererbte Attribute und Operationen werden in der UML *nicht* aufgeführt

# Beispiel für Vererbung in der UML

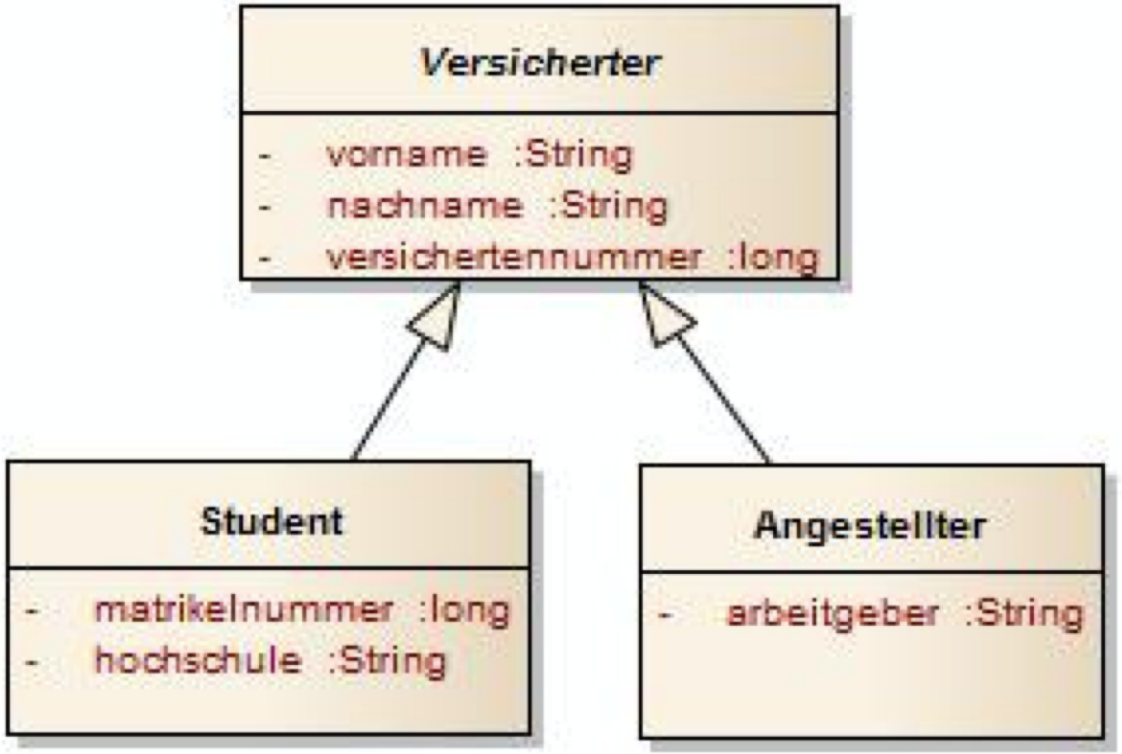
- Ein Bus
  - ist ein (spezieller) Mietwagen,
  - der eine (große) Anzahl von Sitzplätzen bietet

- Ein Lkw
  - ist ein (spezieller) Mietwagen,
  - der beladen und entladen werden kann




Ererbte Attribute und Operationen werden in der UML *nicht* aufgeführt

# Beispiel

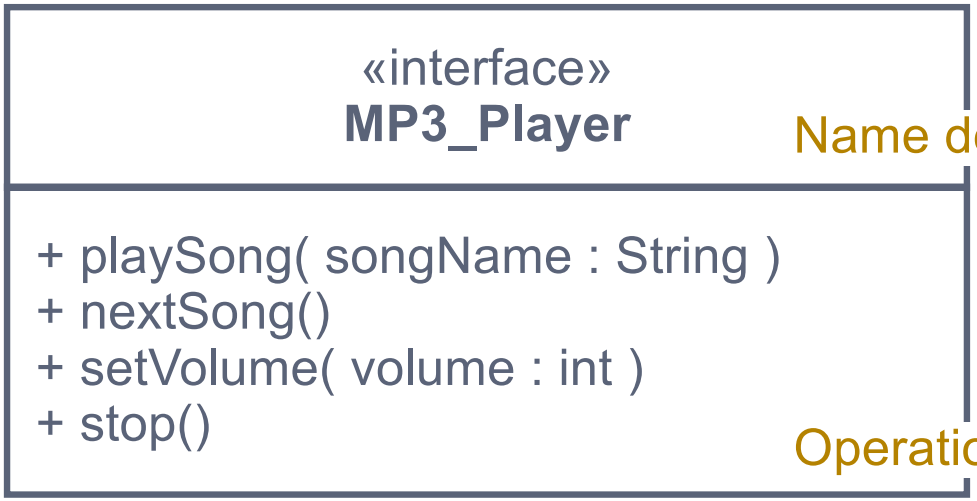

Versicherter
- vorname :String
- nachname :String
- versichertennummer :long
Student
- matrikelnummer :long
- hochschule :String
Angestellter
- arbeitgeber :String

# Was ist eine Schnittstelle?

- Ein Interface wird benutzt, um zwei oder mehr Komponenten miteinander zu verbinden
- Festlegungen, um die Kombinierbarkeit sicherzustellen
- Besser wäre: „Verbindungsstelle“
- Eine Schnittstelle *tut nichts selbst*

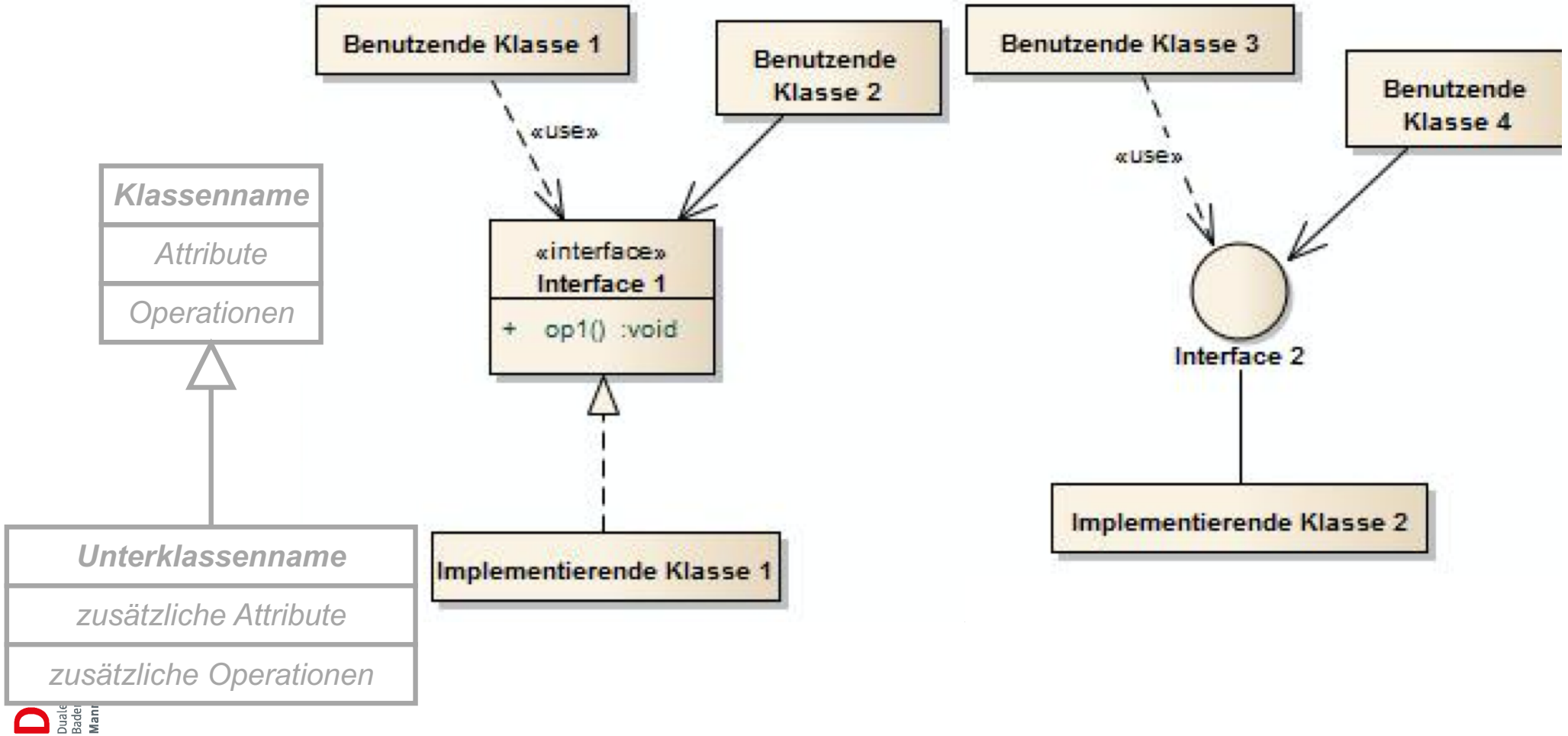
- Beispiel: Die äußere Form eines Steckers zusammen mit der Konvention über die Verdrahtung der drei Kontakte

# Interfaces in der UML: Beispiel




- Eine Klasse, die ein Interface zur Verfügung stellt (es implementiert), muss alle Operationen zur Verfügung stellen, die von diesem Interface vorgeschrieben werden
- Interface-Namen beginnen mit einem Großbuchstaben

# Darstellung eines Interface' in der UML

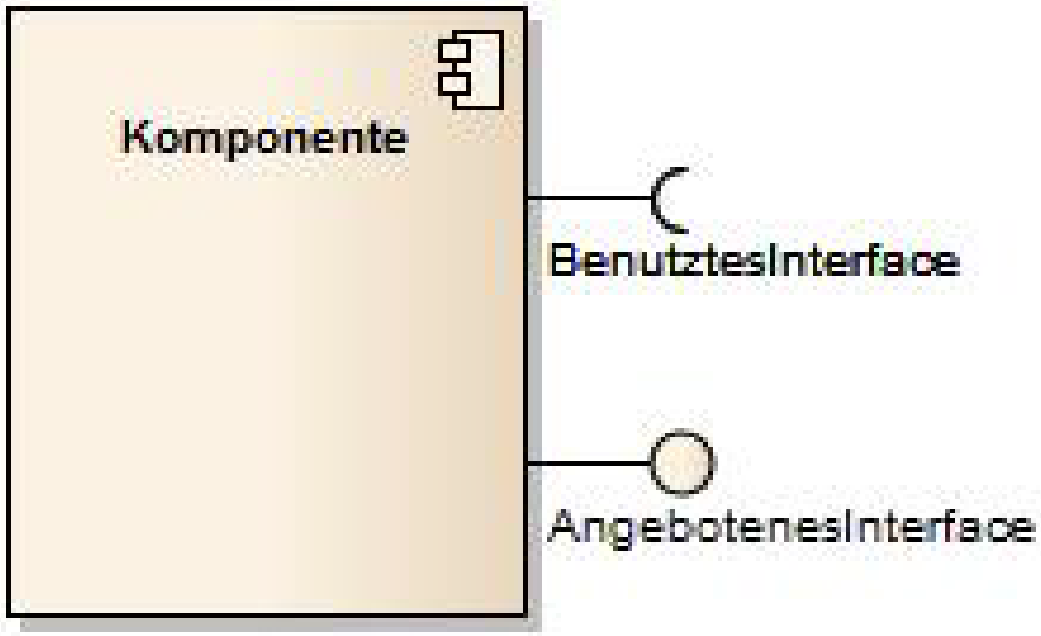


# Was ist eine Komponente?

## Fünf Kriterien

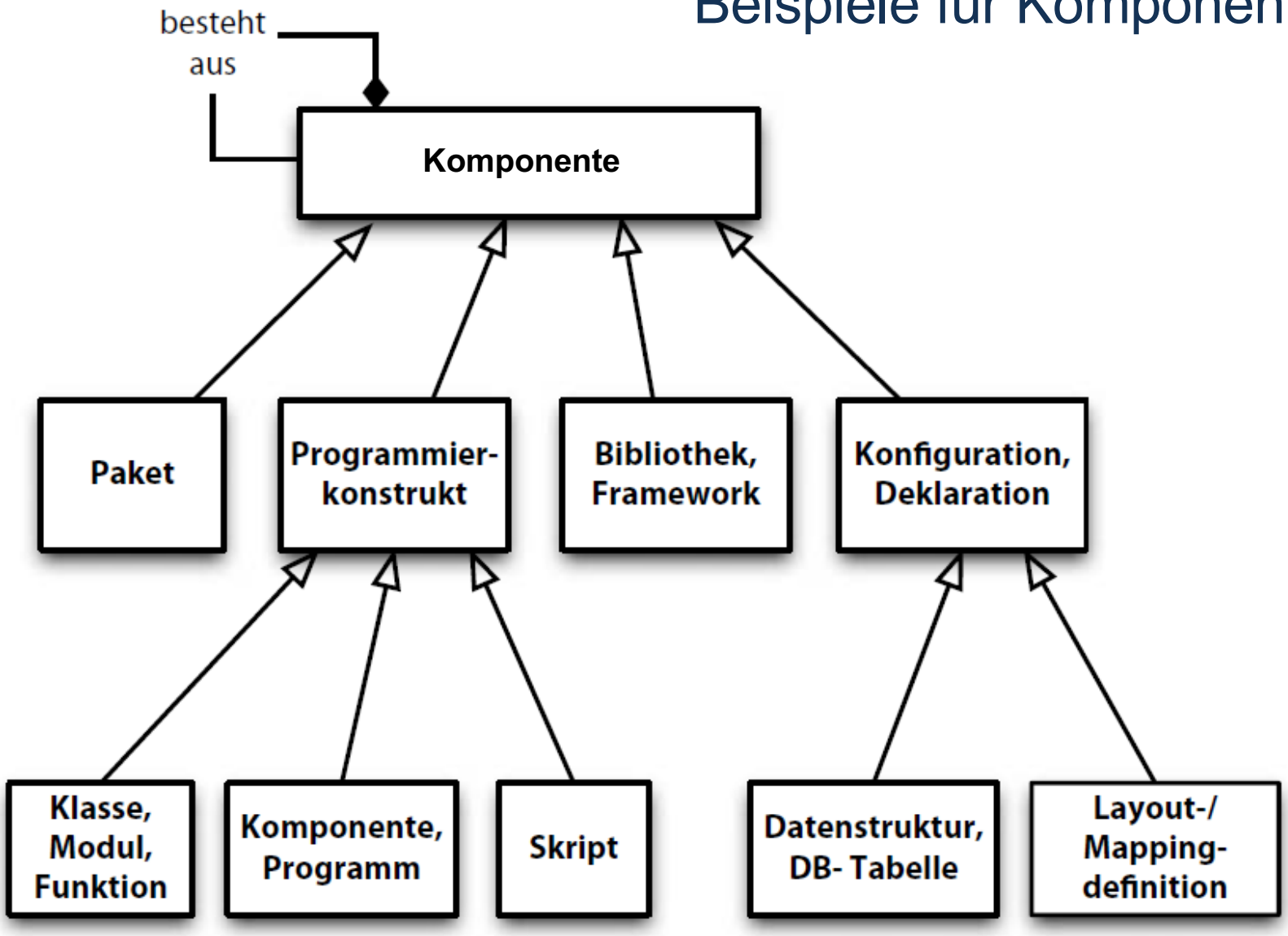
*nach Szyperski & Messerschmitt*

- Wird mehrfach genutzt
- Ist nicht kontextabhängig
- Kann mit anderen Komponenten zusammen verwendet werden
- Ist gekapselt
- Ist eine unabhängige Einheit bzgl. Auslieferung und Versionierung



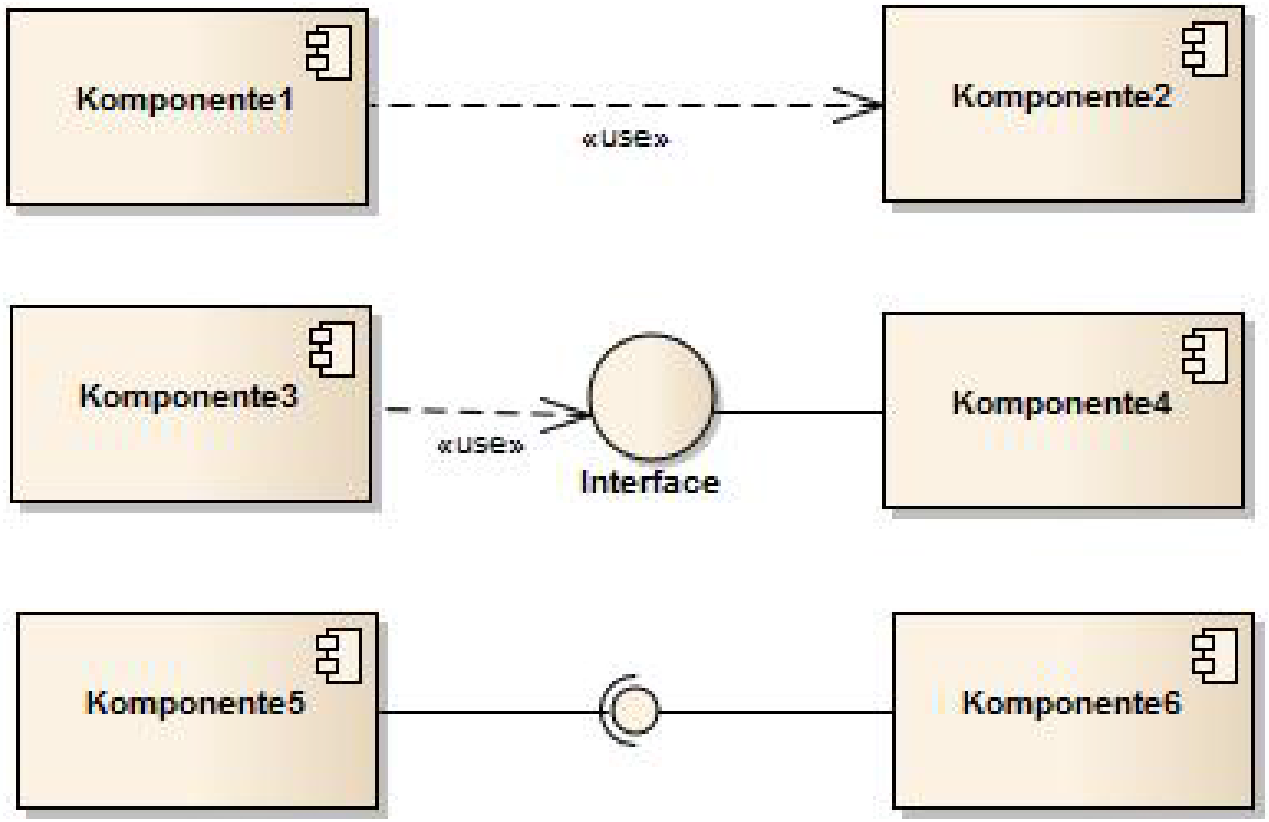

Komponenten können Hardware oder Software (oder beides) sein

# Beispiele für Komponenten

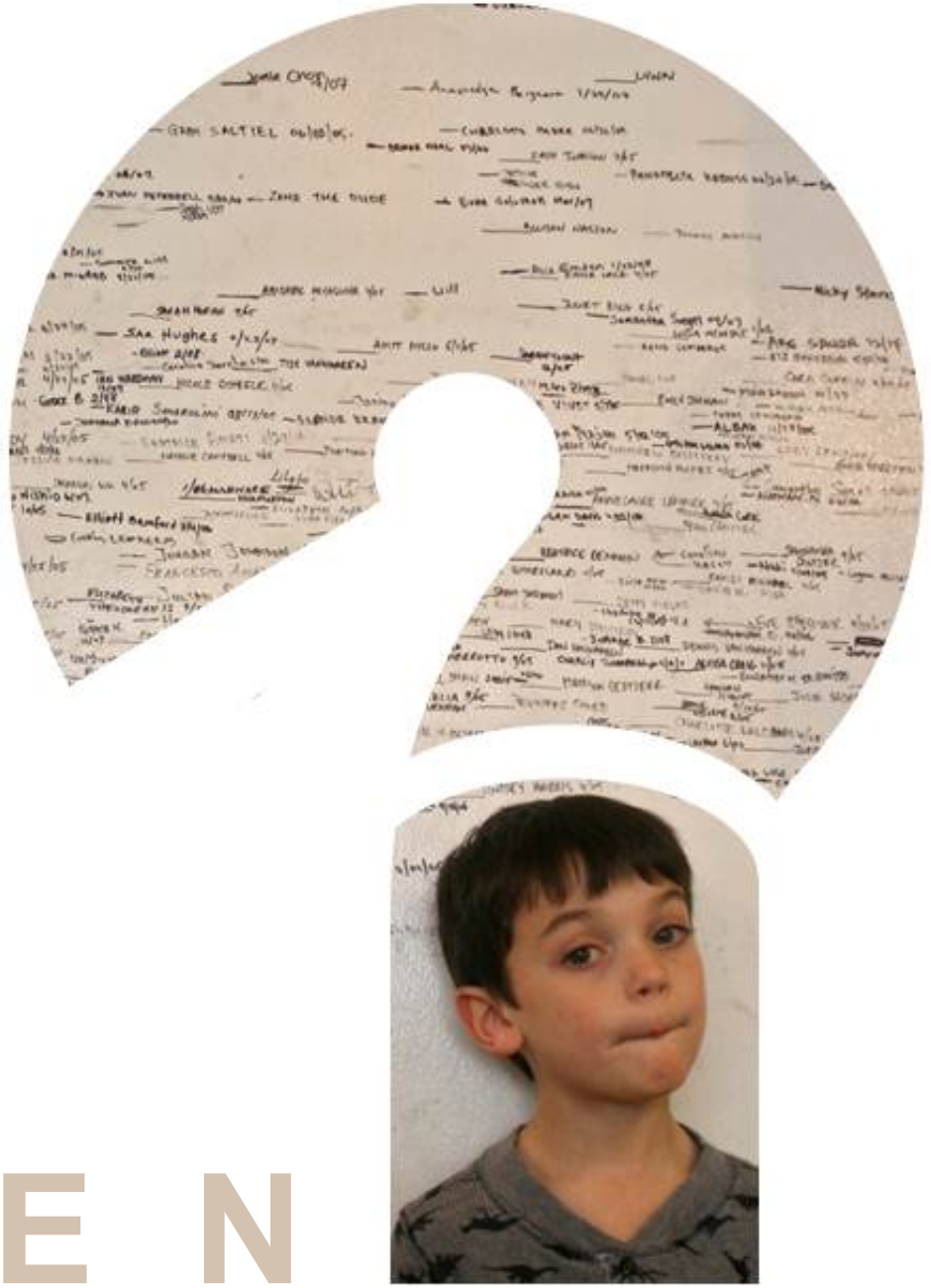



Gernot Starke, Peter Hruschka: Software-Architektur kompakt, 2009

# Komponenten benutzen einander, mit oder ohne Interfaces



# F R A G E N



# Überlegungen für das Design

- Wir überlegen uns, *welche Klassen* (zunächst *ohne Operationen und Attribute*!) wir vermutlich benötigen werden, *zum Beispiel*:
  - eine Klasse für die GUI
  - eine Klasse für die eigentliche Verwaltung
  - eventuell weitere Hilfsklassen (die wir aber noch nicht kennen)

- Wir überlegen uns, *wie* diese Klassen *zusammenarbeiten* (genauer: welche Operationen und Attribute sie dafür brauchen) und zeichnen das in ein Sequenzdiagramm ein (sowie gleichzeitig in ein Klassendiagramm)

DHBW Duale Hochschule Baden-Württemberg Mannheim

# Heuristik für das objektorientierte Design

Idee

Jede Art von Objekt in der Realität wird durch eine Klasse im Klassendiagramm repräsentiert

Vorgehen

1. Identifiziere Klassen und Attribute
   anhand der Substantive aus der Problembeschreibung
2. Identifiziere Beziehungen zwischen den Klassen
3. Identifiziere Operationen (Methoden)
   anhand der Verben aus der Problembeschreibung
   (das machen wir noch genauer!)
4. Überlege, ob Abstraktionen (Interfaces, Oberklassen) sinnvoll eingesetzt werden können / sollten

→ Das ist eine grobe Vereinfachung!!

# F R A G E N