Kapitel 5

Objektorientiertes Programmieren

Übersicht

Übersicht (2)

Abschnitt 5.1

Objektorientierte Programmieren

Objektorientiertes Programmieren

- Simula 67 – erste voll objektorientierte Sprache
- Smalltalk 80 – graphische Benutzeroberflächen/Programmieren
- C++ (1983), Java (1996) – C-basierte Implementierungen

Simula-67

- Algol-60 basierte Sprache
- Funktionalität vergleichbar mit Java
- Entwickelt für Simulationen

Simulationsprobleme

- Spiele: SimCity, The Sims, ...
- Warteschlange im Supermarkt – sollte man wechseln?
- Marktsimulation – was ist der Effekt von Steuererhöhungen?

Simulationsbeispiel

- Fragestellung: ist es schneller zu wechseln?
- Ansatz: Simulation + Messungen
- Objekte: Käufer, Einkaufswagen, Kassierer, Schlange

Simulationsbeispiel (2)

Konzepte:

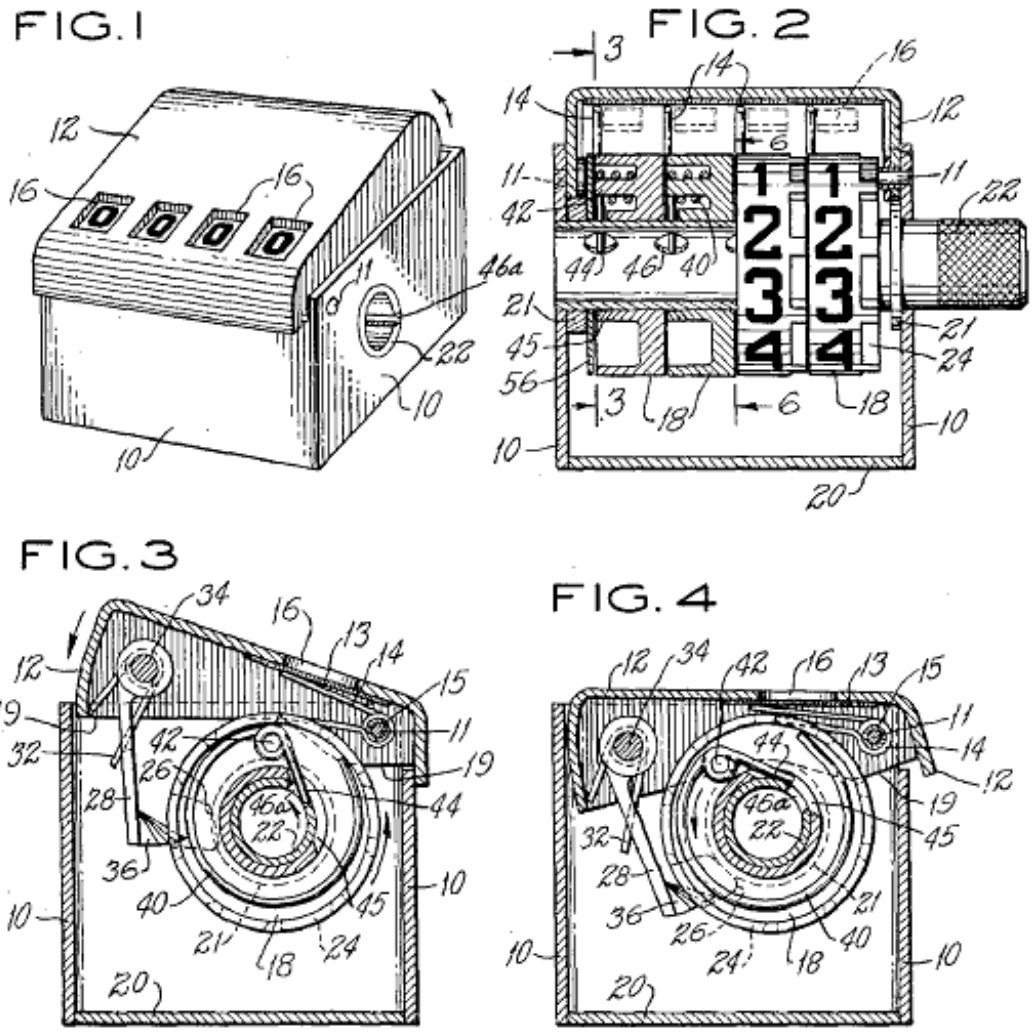
- Objekte, Klassen, Instanzen (objects, classes, instances)
- Instanzvariablen (instance variables)
- Nachrichten und Methoden (messages, methods)
- Kapselung (encapsulation)

Einfaches Java Beispiel

Einfaches Java Beispiel (2)

```
class Counter {
    int value;
    void reset() {
        value = 0;
    }
    void increment() {
        value = value + 1;
    }
    int examine() {
        return value;
    }
};
```

Einfaches Java Beispiel (3)



Einfaches Java Beispiel (4)

```
public class Counter {
    private int value;
    public void reset() {
        value = 0;
    }
    public void increment() {
        value = value + 1;
    }
    public int examine() {
        return value;
    }
};
```

Einfaches Java Beispiel (5)

Beachte:

- Klassendefinition: `class{...}`
- Instanzvariablem: Variablen innerhalb der Klasse
- Methoden: Prozeduren innerhalb Klasse
- Kapselung: `public` / `private`
- Scoping: Methoden können auf Instanzvariablen zugreifen

Einfaches Java Beispiel (6)

```
Counter counter = new Counter();
counter.reset();
counter.increment();
counter.increment();
counter.increment();
counter.increment();
print(""+counter.examine());
```

Einfaches Java Beispiel (7)

Beachte:

- Erzeugung von Instanzen mit `new`
- Aufruf von Methoden mit `object.method(arguments)`

Einfaches Java Beispiel (8)

Der Vollständigkeit halber: `static`

```
public class A {
    static int x;     // Klassenvariable
    int y;            // Instanzvariable
    static int f() {  // statische Methode
        return x;     // (y waere falsch)
    }
}
A.f();
A a = new A();
a.f();
```

Einfaches Java Beispiel (9)

Prozedural vs. Objekt-orientiert

```
class Counter {
    int value;
}

void increment(Counter c) {
    c.value += 1;
}

Counter c = new Counter();
increment(c);
```

```
class Counter {
    int value;
    void increment() {
        value += 1;
    }
}

Counter c = new Counter();
c.increment();
```

Im objektorientierten Stil wird das Objekt als implizites Argument an Methoden übergeben.

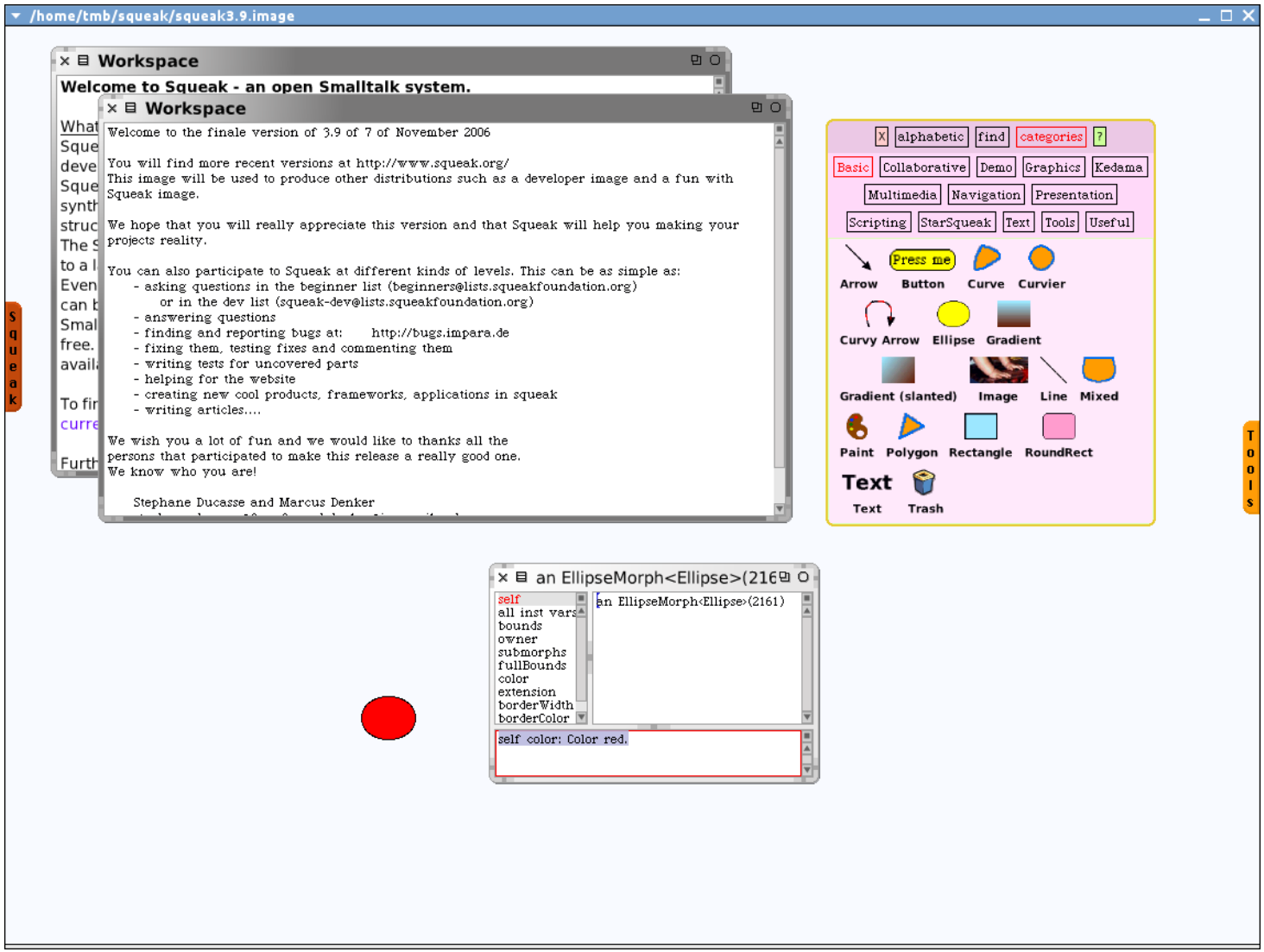
Einfaches Java Beispiel (10)

```
class Counter {
    int value;
    void increment() {
        value += 1;
    }
}
```

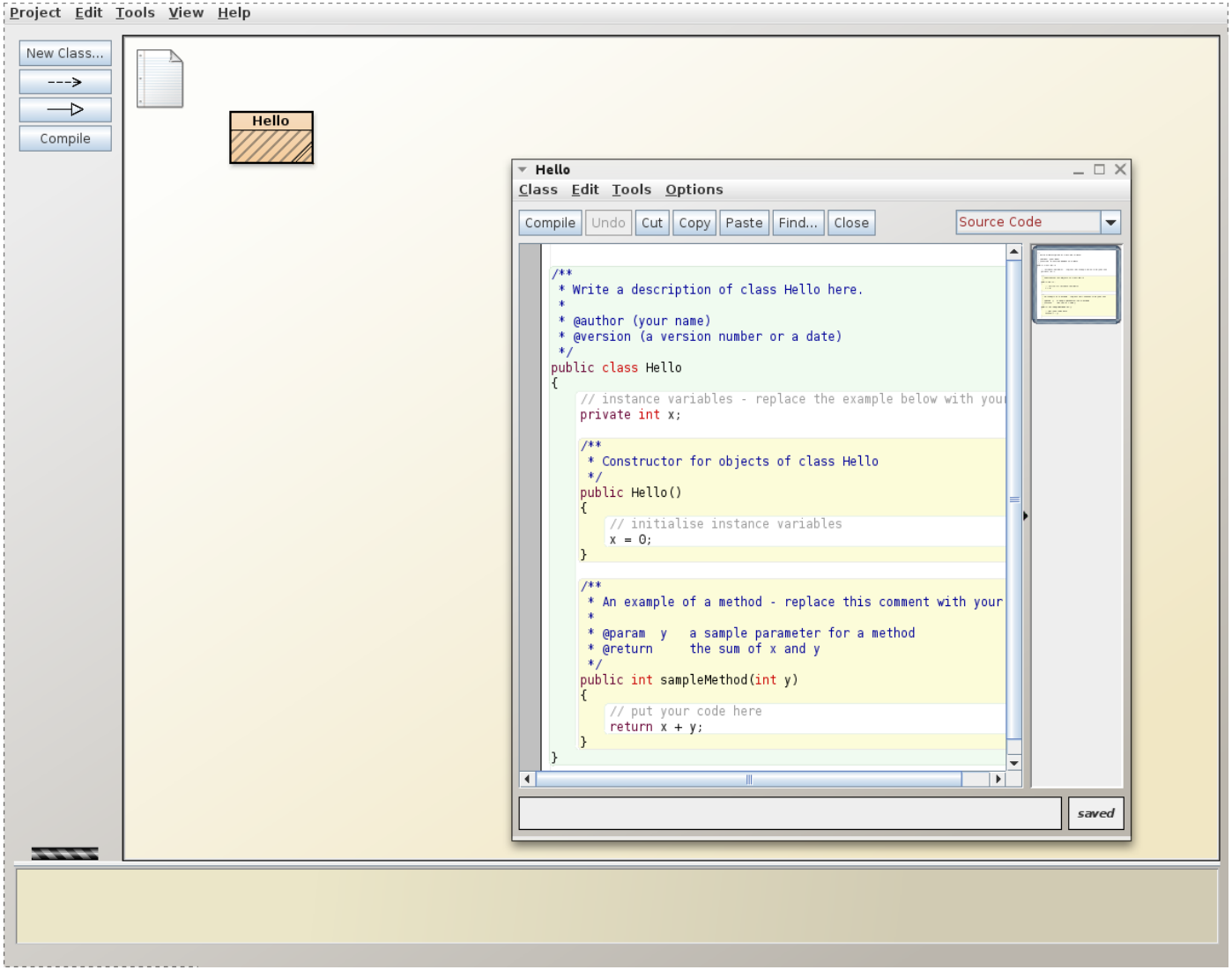
```
class Counter {
    int value;
    void increment() {
        this.value += 1;
    }
}
```

Das implizite Argument kann explizit mit `this` aufgerufen werden.

Smalltalk


/home/tmb/squeak/squeak3.9.image
Workspace
Welcome to Squeak - an open Smalltalk system.
Workspace
Welcome to the finale version of 3.9 of 7 of November 2006
You will find more recent versions at http://www.squeak.org/
This image will be used to produce other distributions such as a developer image and a fun with Squeak image.
We hope that you will really appreciate this version and that Squeak will help you making your projects reality.
You can also participate to Squeak at different kinds of levels. This can be as simple as:
- asking questions in the beginner list (beginners@lists.squeakfoundation.org)
or in the dev list (squeak-dev@lists.squeakfoundation.org)
- answering questions
- finding and reporting bugs at: http://bugs.impara.de
- fixing them, testing fixes and commenting them
- writing tests for uncovered parts
- helping for the website
- creating new cool products, frameworks, applications in squeak
- writing articles....
We wish you a lot of fun and we would like to thanks all the persons that participated to make this release a really good one.
We know who you are!
Stephane Ducasse and Marcus Denker
X alphabetic find categories ?
Basic Collaborative Demo Graphics Kedama
Multimedia Navigation Presentation
Scripting StarSqueak Text Tools Useful
Press me
Arrow Button Curve Curvier
Curvy Arrow Ellipse Gradient
Gradient (slanted) Image Line Mixed
Paint Polygon Rectangle RoundRect
Text
Text Trash
an EllipseMorph<Ellipse>(216
self
all inst vars
bounds
owner
submorphs
fullBounds
color
extension
borderWidth
borderColor
an EllipseMorph<Ellipse>(2161)
self color: Color red.
Squeak
TOOLS

BlueJ


Project Edit Tools View Help
New Class...
Compile
Hello
Hello
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Close
Source Code
/**
* Write a description of class Hello here.
*
* @author (your name)
* @version (a version number or a date)
*/
public class Hello
{
// instance variables - replace the example below with your
private int x;
/**
* Constructor for objects of class Hello
*/
public Hello()
{
// initialise instance variables
x = 0;
}
/**
* An example of a method - replace this comment with your
*
* @param y a sample parameter for a method
* @return the sum of x and y
*/
public int sampleMethod(int y)
{
// put your code here
return x + y;
}
}
saved

Abschnitt 5.2

Objektorientierte Modellierung von SW-Systemen

Unterabschnitt 5.2.1

Objektorientiertes Paradigma

Objektorientierte Modellierung von Softwaresystemen

"The basic philosophy underlying object-oriented programming is to make the programs as far as possible reflect that part of the reality they are going to treat.

human beings from the outset are used to and trained in the perception of what is going on in the real world. The closer it is possible to use this way of thinking in programming, the easier it is to write and understand programs.

O. Lehrmann Madsen, B. Moller-Petersen, K. Nygaard: Object-oriented Programming in the BETA Programming Language, Addison-Wesley, 1993.

Objektorientiertes Paradigma:

Für Menschen besteht

- die physische/materielle Umgebung und
- die gedankliche/geistige Welt

aus

- logisch zusammengehörigen Objekten mit
- eigenständiger Identität
- einem Zustand, der sich mit der Zeit ändern kann,
- der Möglichkeit den Zustand zu verändern
- die Möglichkeit zu kommunizieren und kooperieren

Beispiele: (zum Objektbegriff)

Physische Objekte:

- Stühle, Tische, DVD-Spieler, Computer,
- Autos, Staudämme,
- Häuser, Städte, Länder, Flüsse, Atmosphäre,
- Personen, Tiere, Pflanzen, Augen, Organe,
- Bücher, Bibliotheken, Läden

Beispiele: (zum Objektbegriff) (2)

Gedankliche Objekte und Werte:

- Bücher, Vorlesungen, Prüfungen,
- Reisen, Ruderregatten, Kriege,
- Gesetze, Regeln, Pläne,
- Konten, Börsenkurse, Optionsscheine
- Sprachen, Völker, Religionen (?)
- Prozedur, Variable, Programme,
- Web-Server, Telefonnetze, GUI-Fenster,
- Funktionen, Mengen, Zahlen, Kreise.

Beispiele: (zum Objektbegriff) (3)

Begriffe ohne Objektcharakter (?):

- Wachstum, Farbe, Größe,
- Klugheit, Liebe, Schönheit, Sein,
- Demokratie, Humor,
- Einigkeit, Gerechtigkeit, Freiheit,
- Freizeit, Arbeit,
- Effizienz, Information, Sichtbarkeit.

Zum Objektbegriff:

Zustand:

- Objekte haben *Attribute* (Alter, Größe,...).
- Die *Werte der Attribute* können sich verändern; in der Programmierung heißt das, dass jedes Objekt für jedes Attribut eine Variable besitzt.
- Der *Zustand eines Objekts* zu einem Zeitpunkt ist charakterisiert durch die aktuellen Attributwerte.

Zum Objektbegriff: (2)

Lebensdauer:

- Jedes Objekt hat eine Lebensdauer; das ist die Zeit von seiner Entstehung bis zum Verschwinden bzw. von seiner Erzeugung bis zur Löschung.
- Die Lebensdauer kann in Zeit oder in Ablaufschritten gemessen werden.

Zum Objektbegriff: (3)

Aufenthaltsort:

- Objekte besitzen normalerweise einen Ort, der durch eine Adresse charakterisiert wird.
- Beispiele für Adressen: Hausadresse, Emailadresse, Web-Adresse, Telefonnummer, Geo-Koordinaten, Rechneradresse, Adresse im Hauptspeicher.

Zum Objektbegriff: (4)

Verhalten:

- Im Allg. können Objekte aktiv sein; sie können
  - → ihren Zustand verändern,
  - → ihren Aufenthaltsort verändern,
  - → anderen Objekten eine Nachricht schicken,
  - → Nachrichten von anderen Objekten empfangen
  - → neue Objekte erzeugen.
- Zu jeder Nachricht gibt es eine sogenannte Methode, die beschreibt, wie eine Nachricht bearbeitet wird.

Zum Objektbegriff: (5)

Identität / Gleichheit:
Im Allg. können Objekte aktiv sein; sie können

- Zu einem Objekt gibt es im Allg. gleichartige Objekte (z.B. zwei Bücher, zwei Häuser,...)
- Objekte kann man vergleichen.
- Objekte besitzen eine Identität, die vom Zustand unabhängig ist; d.h. sie können sich in allen Eigenschaften gleichen, ohne identisch zu sein (z.B. zwei baugleiche Autos).
- Insbesondere kann man durch Klonen Objekte erzeugen, die sich in allen Eigenschaften gleichen, aber nicht identisch sind.

Bemerkungen:

- Die *charakteristischen Eigenschaften* Zustand, Lebensdauer, Ort, Verhalten und Identität sind als Hilfestellung, nicht als scharfe Begriffsklärung zu verstehen.
- Anhand der charakteristischen Eigenschaften lässt sich untersuchen, wie ausgeprägt der Objektcharakter eines Gegenstands ist.

Beispiel:

Typisches Objekt: Repräsentation eines Autos und seines Zustands.

Attribute und ihr Zustand :

VIN:	*2390480148018DE9348'*
Farbe:	*rot*
Marke:	*Daimler*
Schiebedach:	*ja*
Alter:	*12 Jahre*
Tankfüllung:	*halbvoll*
Kilometerstand:	*234568*
Geschwindigkeit:	*130 km/h*

- All diese Attribute sind sichtbar; gibt es auch private Attribute?
- Dies ist nur ein Modell/eine Abstraktion der Wirklichkeit.
- Ein detaillierteres Modell enthält die interne Struktur.

Beispiel: (2)

Nachrichten, die verstanden werden:

- bremsen
- Gas geben
- Scheibenwischer (an/aus)
- ...
- Auch dass sind nur die “öffentlichen” Nachrichten.

Beispiel: (3)

Objekte in Softwaresystemen haben oft keine direkte realen Äquivalente:

- GUI Fenster
- Knöpfe
- Dateien
- MP3 Dateien
- “Dreiecke”, “3D Modelle”
- Verhaltensweisen

Beispiel: (4)

GUI Fenster

- Position, Größe
- Rahmenfarbe
- Zustand
- Fensterinhalte
- Verhaltensweisen, *event handlers*

(Squeak)

Bemerkung:

Die üblichen mathematischen Objekte (Funktionen, Mengen, Zahlen) besitzen

- keinen (veränderlichen) Zustand,
- keine Lebensdauer,
- keinen Aufenthaltsort,
- kein Verhalten,
- keine Identität jenseits der Gleichheit.

- Deshalb nennen wir sie ***Werte***.
- vergleiche mit ***Referenzen***
- Werte können aber auch durch Objekte repräsentiert werden

Beziehungen zwischen Objekten:

Objekte können zueinander in unterschiedlichen Beziehungen stehen:

- Objekt X kann ein Teil von Objekt Y sein.
- Objekt X kann mit einem anderen assoziiert oder verknüpft sein.

Besteht keine Beziehung zwischen zwei Objekten, können sie auch nicht direkt mit einander kommunizieren.

Nachrichten und Methoden

Objekte *bieten **Dienste** an* und *nehmen Dienste* anderer Objekte *in Anspruch*.

Um einen Dienst in Anspruch zu nehmen, schickt ein Objekt einem anderen eine ***Nachricht***, die den Dienst bezeichnet und Parameter übergibt (Auftragserteilung).

Erhält ein Objekt eine Nachricht, führt es eine ***Methode*** aus, die der Handlungsvorschrift zur Ausführung des Dienstes entspricht.

In der Objektorientierung werden also Auftragserteilung und –ausführung getrennt.

Beispiel: (Dienste & Aufträge)

Modellierung eines Buchkaufs:

- Buchhändlerin B bietet den Dienst an, per Email Bücher zu bestellen.
- Herr K. (Senderobjekt) gibt Frau B. (Empfängerobjekt) den Auftrag, den "Mann ohne Eigenschaften" zu besorgen.
- Herr K. weiß nicht, wie Frau B. den Auftrag ausführt.
- Frau B. besitzt eine Methode, wie mit dem Auftrag zu verfahren ist. Nach Ausführung schickt sie Herrn K. das Buch zu.

Begriffsklärung: (objektbasiertes System)

In der Objektorientierung werden Systeme als Menge kooperierender Objekte modelliert,

- die untereinander in Beziehung stehen und
- über Nachrichten miteinander kooperieren.

Derartige Systeme heißen ***objektbasiert***.

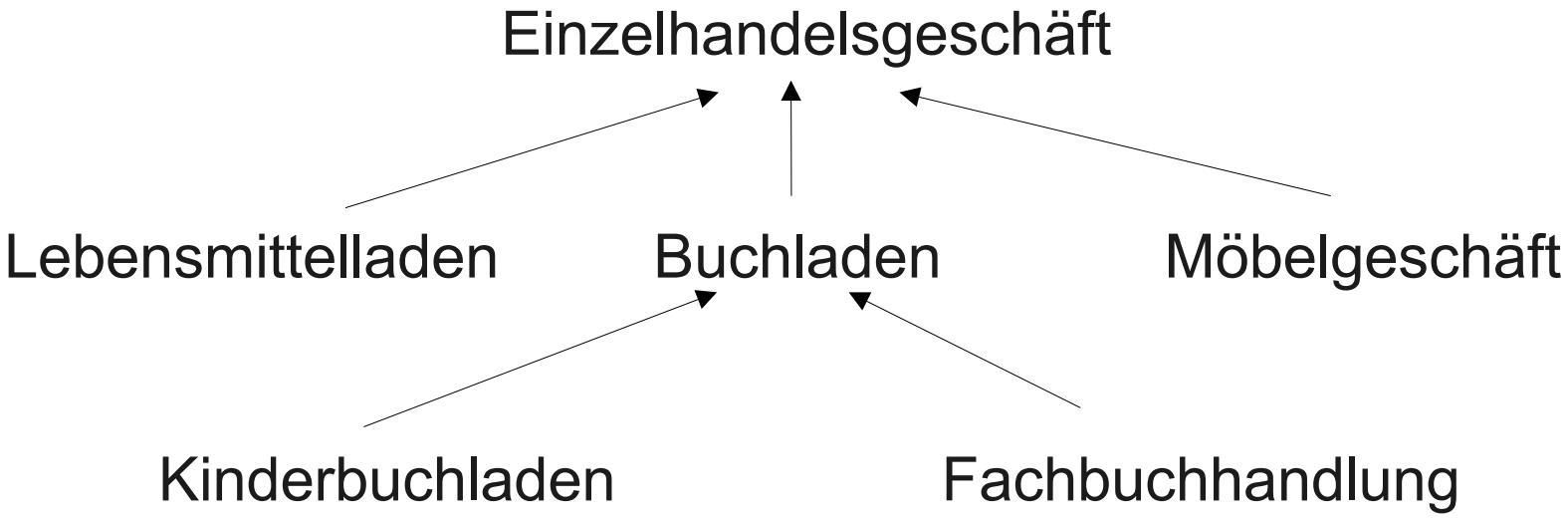
Klassifikation und Vererbung:

Objekte lassen sich nach ihren Eigenschaften klassifizieren:

- Alle Objekte mit ähnlichen Eigenschaften werden zu einer Klasse zusammen gefasst.
- Die Klassen werden hierarchisch geordnet.

Beispiel: (Klassifikation)

Als Objekte betrachten wir Einzelhandelsgeschäfte. Sie lassen sich nach ihren Produkten klassifizieren:



- Die übergeordneten Klassen besitzen Eigenschaften, die allen untergeordneten Klassen gemeinsam sind.
- Beachte die Unterscheidung: Objekt ↔ Klasse

Beobachtung:

- Es gibt Klassen, zu denen nur die Objekte der Unterklassen gehören (z.B. gibt es kein Geschäft, das nur ein Einzelhandelsgeschäft ist). Diese Klassen nennt man ***abstrakt***.
- Es gibt Klassen, zu denen eigene Objekte und die Objekte der untergeordneten Klassen gehören.

Vorteile der Klassifikation:

- Übergeordnete Klassen besitzen Eigenschaften, die allen untergeordneten Klassen gemeinsam sind. Auf Basis dieser Eigenschaften lassen sich Methoden formulieren, die für alle Objekte der untergeordneten Klassen funktionieren.
- Eigenschaften können von übergeordneten zu untergeordneten Klassen vererbt werden.

Begriffsklärung: (objektorientiertes System)

Objektbasierte Systeme, bei denen

- die Objekte Klassen zugeordnet und
- die Klassen gemäß einer Klassifikation hierarchisch geordnet sind und Vererbung erlauben,

heißen ***objektorientiert***.

Unterabschnitt 5.2.2

Zur objektorientierten Modellierung

Zur objektorientierten Modellierung

(Nur Einführung; OO-Modellierung ist Gegenstand von SE 2; siehe auch Goos, Band 2, 10.2)

Die Analyse einer Aufgabenstellung führt zu einem *Modell* des Ausschnitts der Welt, der zur Lösung geeignet ist.

Das Modell beschreibt die wichtigen Eigenschaften des zu entwickelnden Systems. Es orientiert sich zunächst an der Anwendung und nicht der Realisierung.

Zur objektorientierten Modellierung (2)

Bei objektorientierter Modellierung ist zu klären:

1. Welche Objekte werden benötigt?
2. Welche Beziehungen gibt es zwischen den Objekten?
3. Welche Eigenschaften besitzen die Objekte?
4. Wie lassen sich die Objekte klassifizieren?
5. Wie werden die Objekte angewendet?
6. Was ist das Verhalten der Objekte?

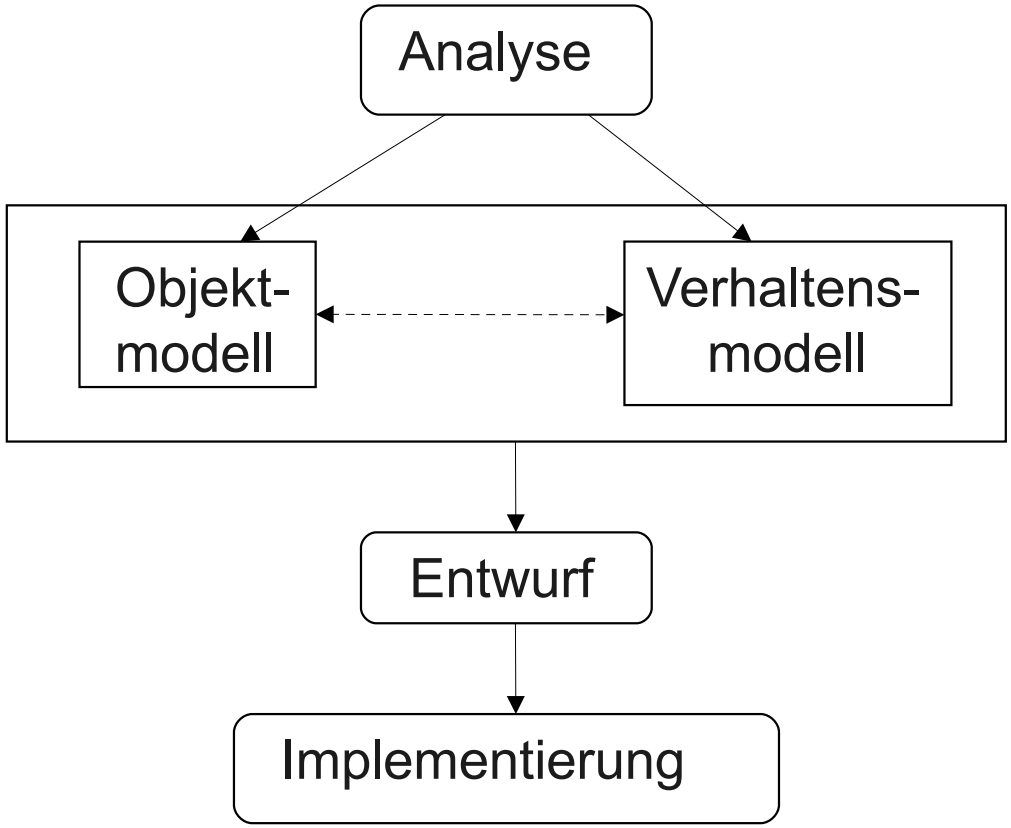
Das *Objektmodell* liefert die Antworten zu 1-4.

Anwendungsfälle beantworten Frage 5.

Das *Verhaltensmodell* klärt Frage 6.

Zur objektorientierten Modellierung (3)

Objektorientierte Analyse in Umfeld der objektorientierten Softwareentwicklung:



Begriffsklärung: (Objekt-, Verhaltensmodell)

Das ***Objektmodell*** beschreibt:

- die relevanten Klassen von Objekten,
- deren Attribute,
- welche Dienste/Nachrichten/Operationen bereitgestellt werden,
- die Beziehungen zwischen den Objekten;
- ggf. mittels einzelner ausgezeichneter Objekte.

Das ***Verhaltensmodell*** beschreibt:

- die Wirkungsweise der Methoden,
- die möglichen Zustände von Objekten,
- das Ablaufverhalten und die Interaktion zwischen den Objekten.

Objektorientierte Analyse: Ein Beispiel

Aufgabe:

Entwickle ein Planungs- und Informationssystem für die Veranstaltungen am Fachbereich Informatik.

Grober Leistungsumfang:

- Angebot der Veranstaltungen festlegen
- Raum- und Zeitplanung
- Vorlesungseinschreibung
- Prüfungsabwicklung

Beschreibungstechnik:

Die Analyseergebnisse beschreiben wir mit UML-Diagrammen. Hier erläutern wir deren Bedeutung informell. Eine präzise Einführung bietet SE 2.

1. Schritt: Auffinden der Schlüsselabstraktionen

Die ***Schlüsselabstraktionen*** sind die zentralen Begriffe des Aufgabenbereichs. Sie bilden den Ausgangspunkt zur Entwicklung des Objektmodells.

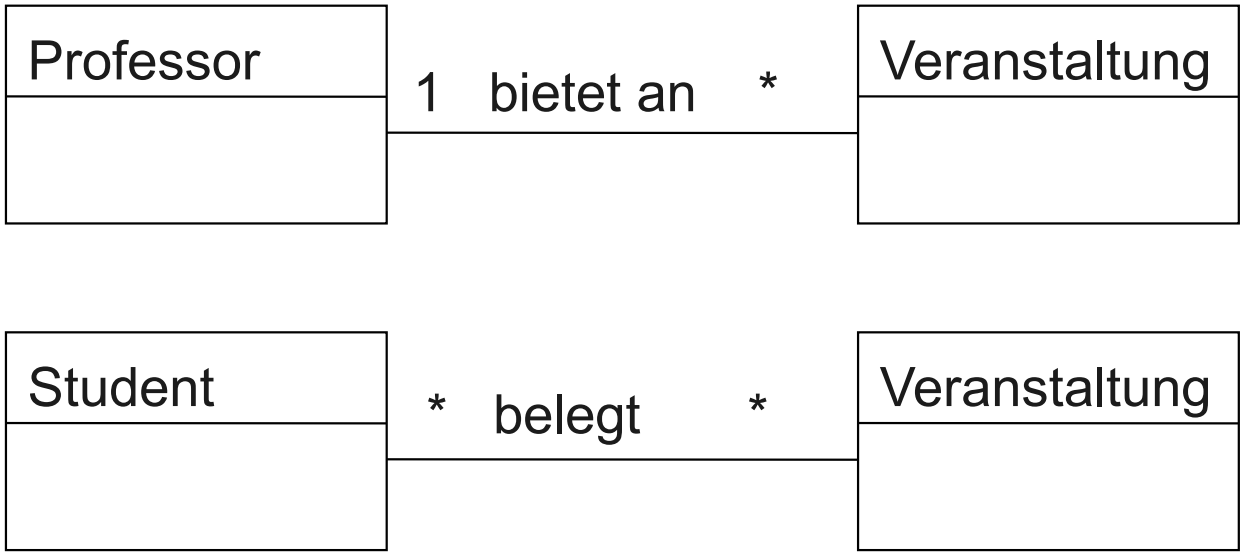
Schlüsselabstraktionen zur Aufgabe:

- Student, Professor
- Veranstaltung (in einem Semester)
- Veranstaltungsverzeichnis für ein Semester
- Vorlesung, Seminar
- Raum
- Klausur
- Zeitslot

Welche Abstraktionen haben den Charakter von Objekten / Werten?

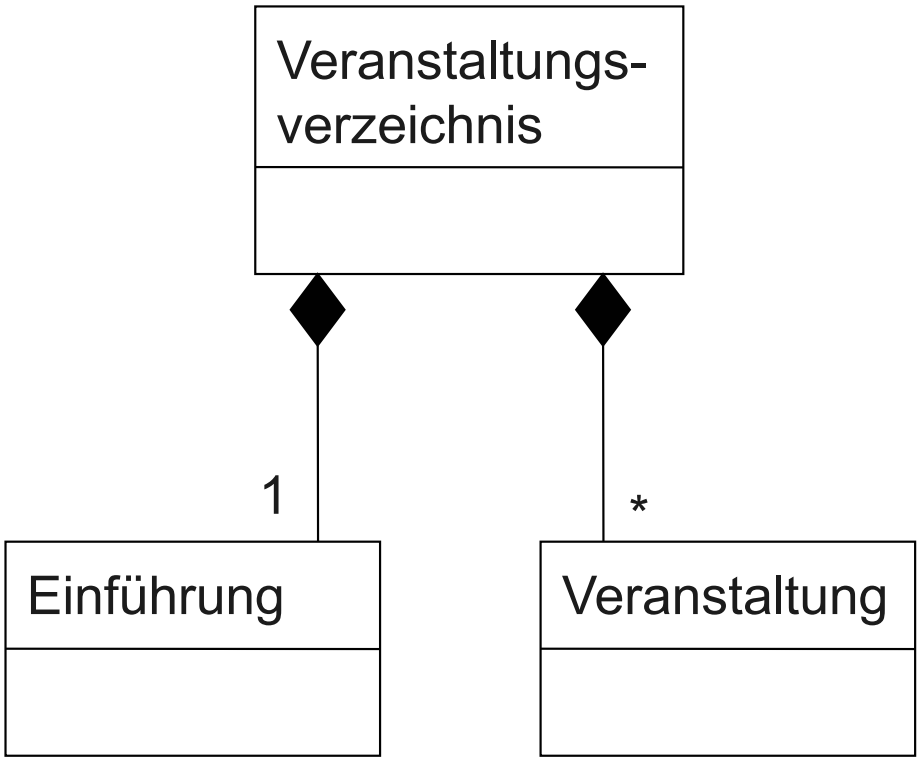
2. Schritt: Ermitteln der Beziehungen

Beziehungen zwischen den Objekten von jeweils zwei Klassen:



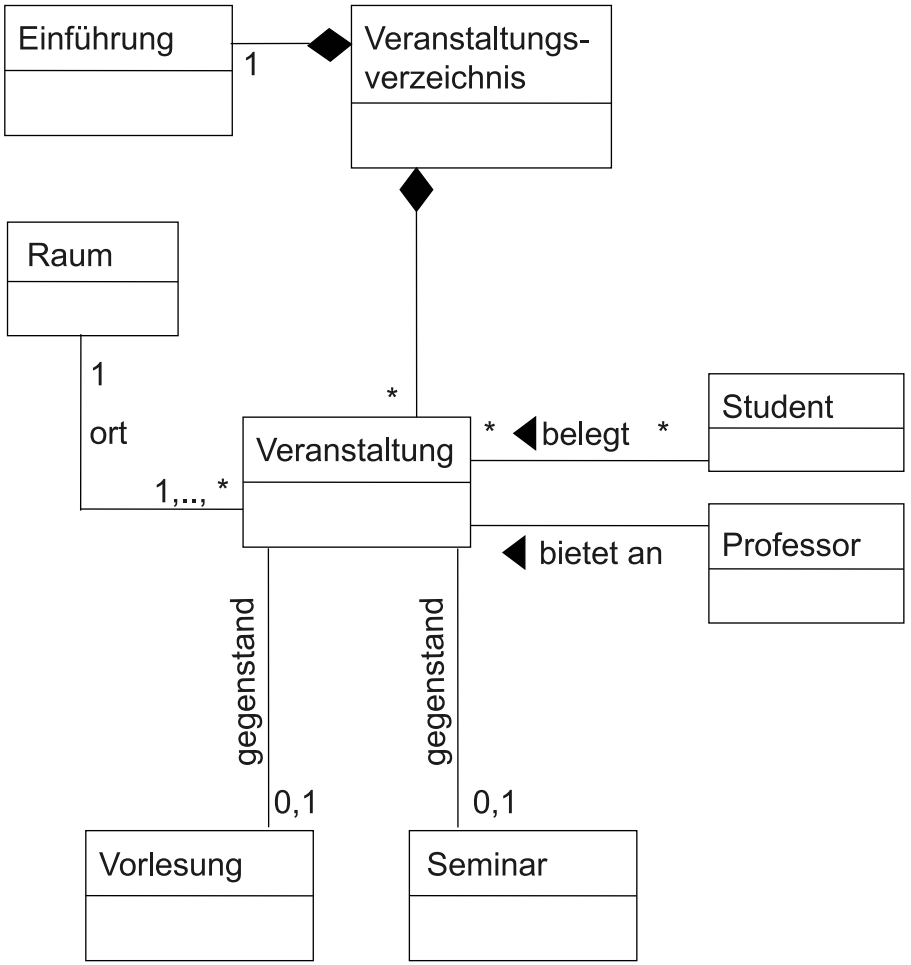
2. Schritt: Ermitteln der Beziehungen (2)

Beziehungen zwischen den Objekten mehrerer Klassen:



2. Schritt: Ermitteln der Beziehungen (3)

Zusammengesetztes Klassendiagramm:



3. Schritt: Ermitteln der Attribute

Beispiele:

Raum
nummer: String sitzplätze: int

Veranstaltung
name: String termine: ??

Student
namw: String semester: int

Professor
name: String gruppe: String

Bemerkung:

Viele Beziehungen lassen sich als Attribute ausdrücken und umgekehrt.

Beispiel:

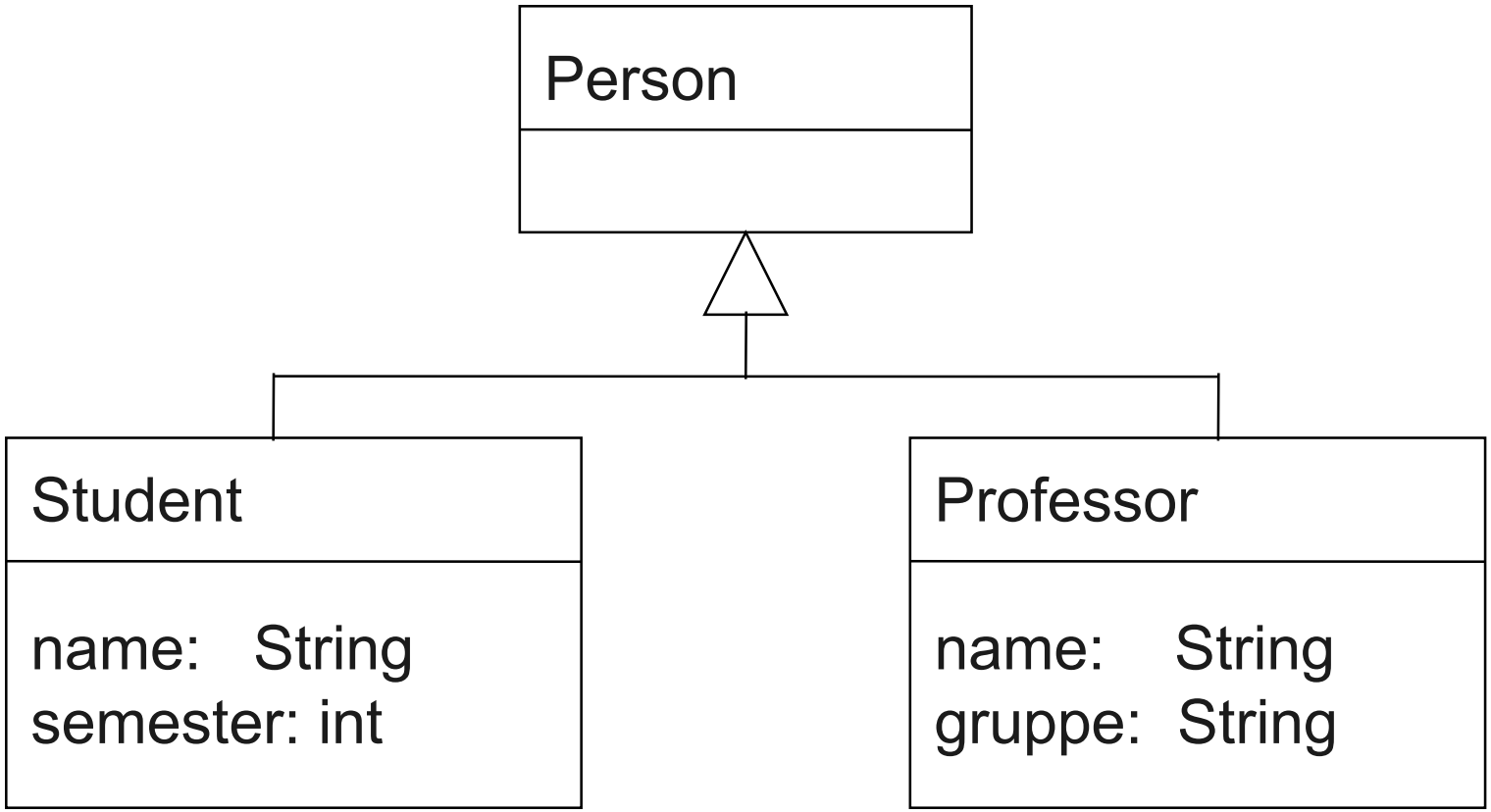
Termine kann man

- als Attribute mit Zeitslots als Typ oder
- als Beziehung zwischen Veranstaltungen und Zeitslots

modellieren.

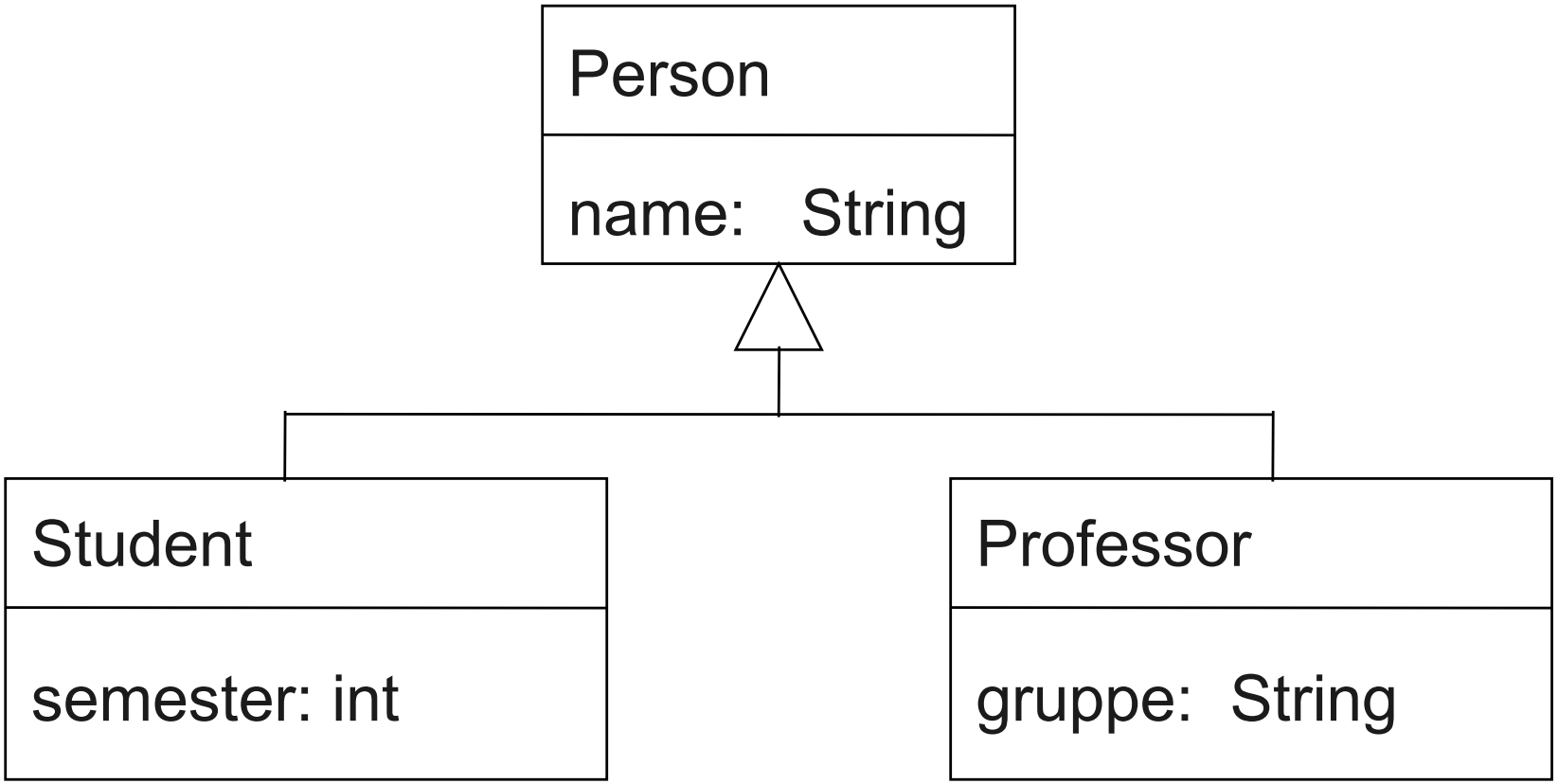
4. Schritt: Ermitteln der Klassifikation

Beispiel:



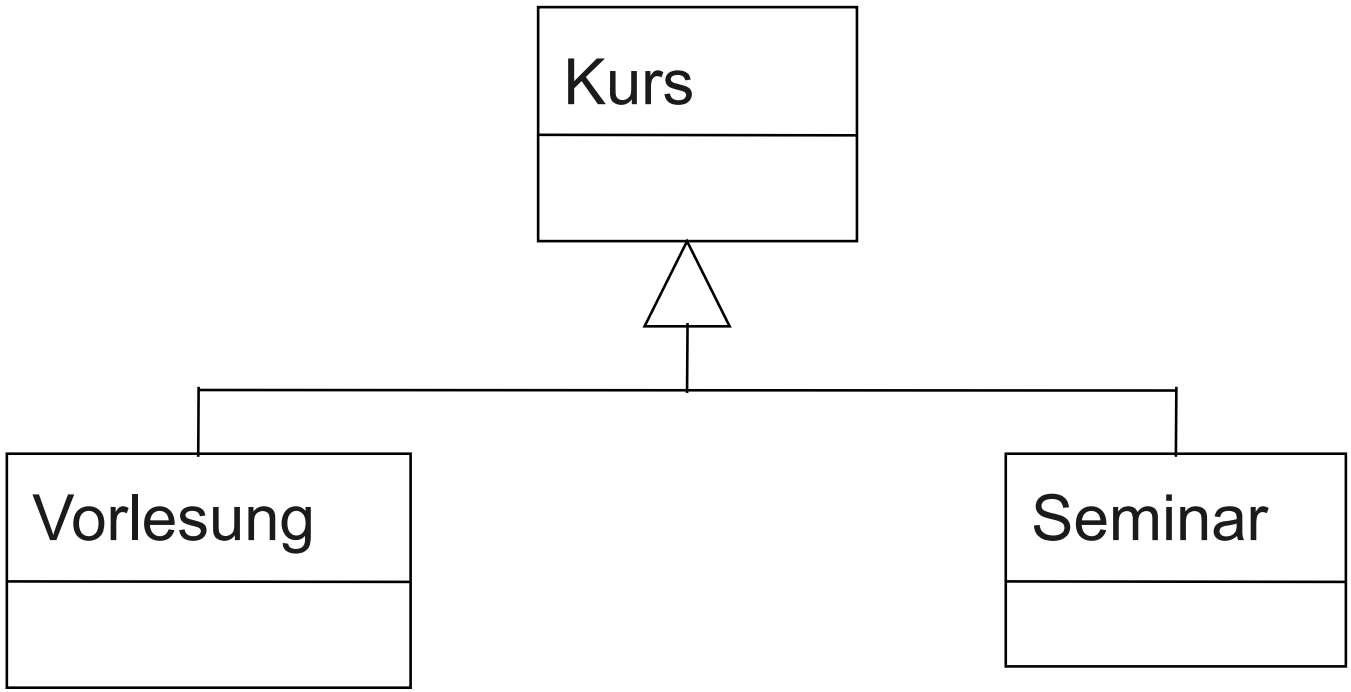
4. Schritt: Ermitteln der Klassifikation (2)

Unter Ausnutzung von Vererbung:



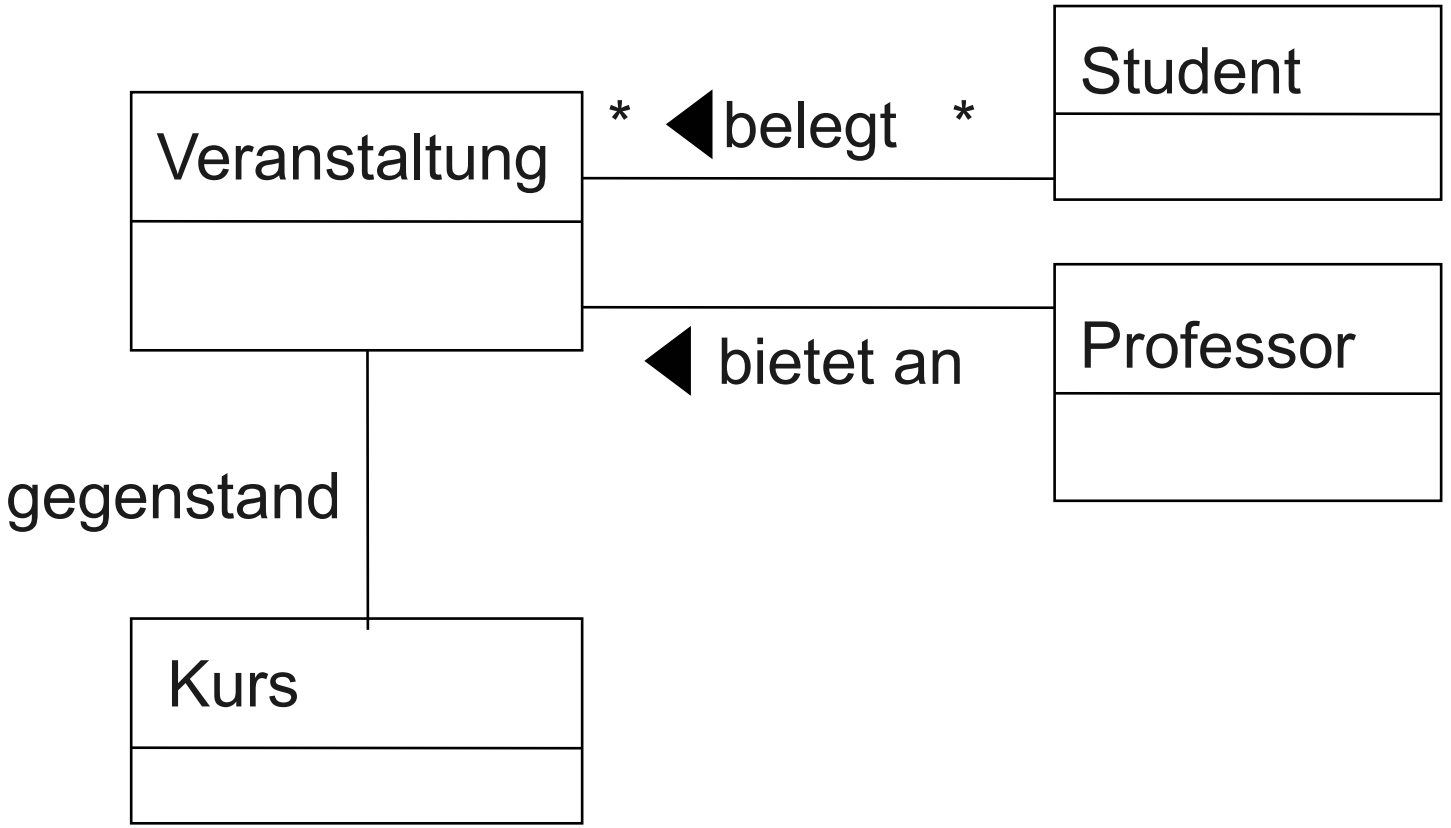
4. Schritt: Ermitteln der Klassifikation (3)

Klassifikation kann zu Vereinfachungen führen:



4. Schritt: Ermitteln der Klassifikation (4)

Dies lässt sich im Klassendiagramm für Veranstaltung verwenden:



5. Schritt: Ermitteln der Invarianten

Invarianten beschreiben Eigenschaften der Objekte oder der Beziehungen zwischen den Objekten, die im Wesentlichen zu allen Ausführungszeitpunkten gelten.

Beispiele:

- Veranstaltungsraum fasst die Anzahl der Beleger
- Zu jedem Zeitslot und Raum gibt es maximal eine Veranstaltung
- Ein Professor bietet nur Veranstaltungen an, die an unterschiedlichen Zeitslots stattfinden.
- Veranstaltungen, die von Studierenden mit bestimmtem Status zu belegen sind, dürfen sich nicht überschneiden.
- Studierende dürfen nur Veranstaltungen belegen, für die sie die Voraussetzungen haben.

6. Schritt: Ermitteln des Verhaltens

Das Verhalten von Objekten wird durch das Versenden von Nachrichten und das Bearbeiten der Nachrichten durch Methoden modelliert.

Wir betrachten hier drei unterschiedliche Arten Verhaltensaspekte zu beschreiben:

- Skizzieren der wesentlichen Anwendungsfälle
- Beschreiben des Nachrichtenaustauschs zwischen einzelnen Objekten
- Beschreibung der Nachrichten, die die Objekte verstehen (*Methodenschnittstelle*)

6. Schritt: Ermitteln des Verhaltens (2)

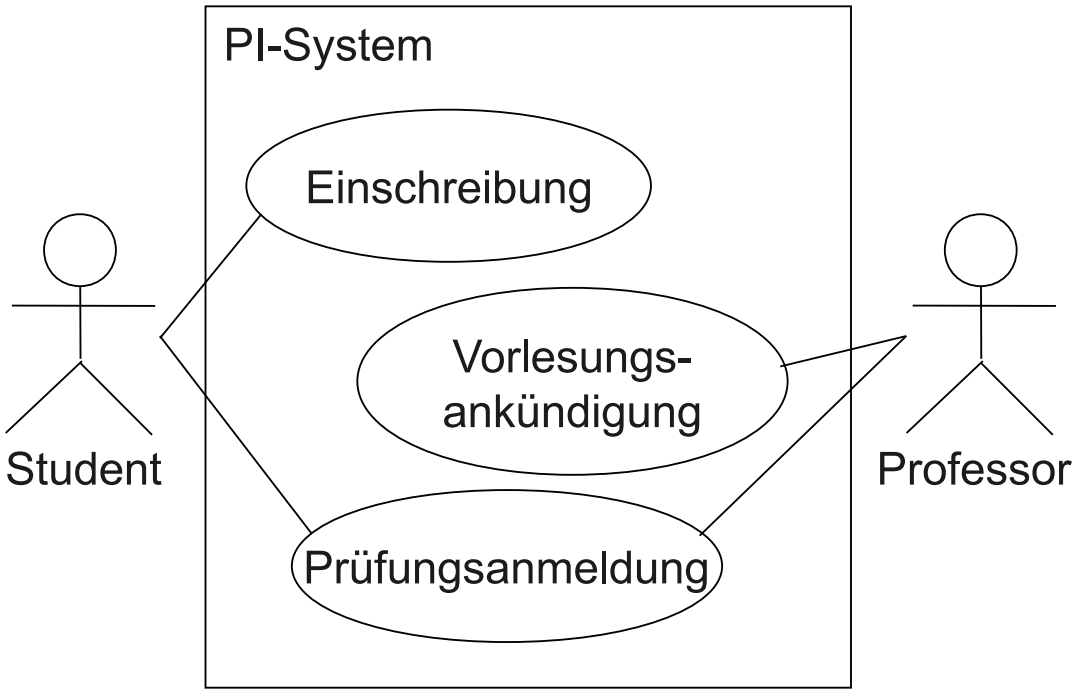
Anwendungsfälle (Use cases)

Ausgangspunkt für die Verhaltensmodellierung sind die wesentlichen Anwendungsfälle des zu entwickelnden Systems.

Ein Anwendungsfall ist ein typischer Vorgang, der mit dem zu realisierenden System durchgeführt wird.

Anwendungsfälle verdeutlichen auch die Abgrenzung des Systems von seiner Umgebung.

Beispiel:



Student und Professor sind sogenannte Akteure. Akteure können auch agierende Softwareteile der Systemumgebung sein.

Interaktion zwischen Objekten

Um die Interaktion zwischen Objekten darzustellen, beschreibt man:

- die beteiligten Objekte
- die Namen der Nachrichten
- ggf. die Parameter der Nachrichten
- den zeitlichen Ablauf der Nachrichten

(UML-Diagramme dazu siehe SE 2)