# Informatik I - Programmierung
# Globalübung
# 2.12.2003

# Objektorientierung
# Konzepte & Notationen

**Thomas Weiler**

**Fachgruppe Informatik**
**RWTH Aachen**



## Inhalt

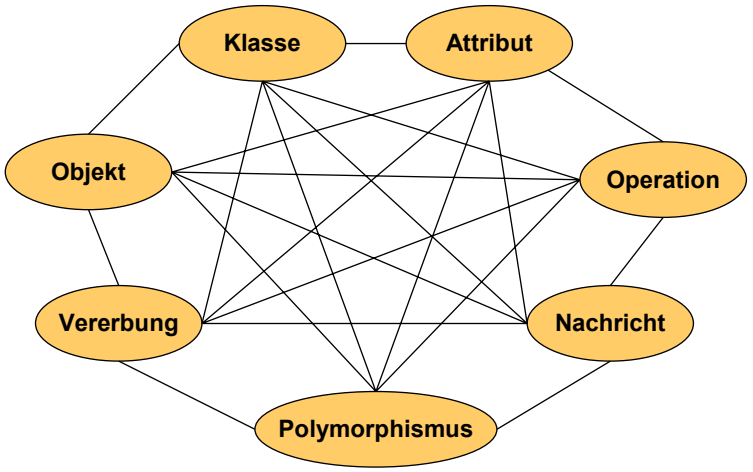
- Objektorientierung
- Notationen
- Entwurfsprinzipien
- Werkzeuge

# Objektorientierung

- **Objektorientierung ist mehr als Programmieren in einer objektorientierten Sprache.**
- **Objektorientierung ist ein durchgängiger Ansatz für die Software-Entwicklung.**
  - Analyse, Design und Codierung
  - Bei allen Tätigkeiten werden die **selben Modellierungskonzepte** verwendet
    - Identifizieren und Definieren von abgeschlossenen autonomen Einheiten, die Struktur und Verhalten besitzen.
    - Finden von Gemeinsamkeiten im Sinne der Generalisierung / Spezialisierung.
  - **Brüche** zwischen den Ergebnissen der Aktivitäten werden geringer.
  - Grenzen zwischen den Aktivitäten **verschwimmen.**

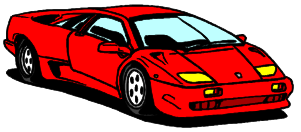



# Grundkonzepte der Objektorientierung



# Objekt - Fachlich

- **Objekte entsprechen den für die Anwendung relevanten Gegenständen**
- **Die Gegenstände sind charakterisiert durch**
  - die **Information**, die sich an ihnen ablesen lässt
  - die **Aktionen**, die an ihnen ausgelöst werden können
- **Objekte kapseln die Informationen und das interne Verhalten und bieten nach außen eine Menge von Operationen (Schnittstelle) an**





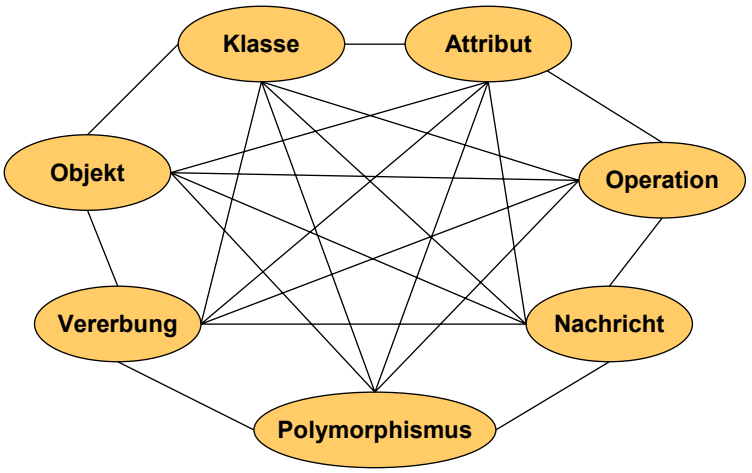
# Objekt - Softwaretechnisch

- **Ein Objekt ist eine Datenkapsel, die aus zwei Teilen besteht**
  - Der Wert der Daten repräsentiert den **Zustand** des Objekts
  - Daten können nur mithilfe von **Operationen** verändert werden (Kapselung)

| Daten |
| --- |
| Operationen |

- **Die Aktivität der Objekte ist die Ausführung ihrer Operationen**
  - Eine Operation wird ausgeführt, wenn ein Objekt eine entsprechende **Nachricht** erhält.
  - Der Objektzustand kann sich **verändern**.
- **Jedem Objekt ist ein Typ zugeordnet**
  - Mögliche Operationen sind durch den **Objekttyp** bestimmt.
  - In den meisten statisch typisierten objektorientierten Programmier-sprachen (wie JAVA) ist der Typ durch die Klassenzugehörigkeit festgelegt.

# Grundkonzepte der Objektorientierung






# Klassen

- **Fachliche Modellierung**
  - Gemeinsame **Attribute** und gemeinsames **Verhalten** bestimmen **Ähnlichkeiten** von unterschiedlichen Gegenständen.
  - Diese Gemeinsamkeit drücken wir abstrahierend
    - in einem **Begriff** und dem
    - dahinterstehenden **Konzept** aus.
- **Begriffe**
  - können in **Beziehung** gesetzt werden (**Begriffsnetz**)
  - lassen sich **generalisieren** oder **spezialisieren** (**Begriffshierarchien)**.
  - Sie bilden als Teil der jeweiligen **Fachsprache** die Grundlage zur Zusammenarbeit zwischen den beteiligten Gruppen und fördern das Verständnis eines Anwendungsbereichs.

# Klassen - die Statik des Systems

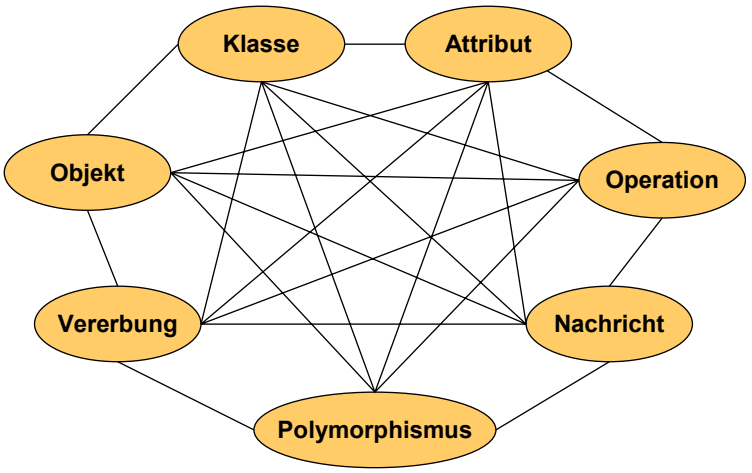
- **Technisch werden Begriffe als Klassen implementiert.**
- **Eine Klasse definiert für ihre Objekte**
  - ihren **Namen**
  - die **Speicherstruktur**
    - Daten, Attribute, Exemplarvariablen
  - die **Operationen** (Methoden, Routinen),
    - von denen ein Teil exportiert wird (**public**)
    - andere werden nur intern benötigt (**private**)
  - die **Vererbungsbeziehung** zu ihren **Oberklassen**
- **Von einer Klasse können beliebig viele Objekte erzeugt werden (Schablone)**



# Klasse - Objekt

- **Eine Klasse legt meist fest,**
  - wie die Exemplare der Klasse erzeugt und initialisiert werden.
- **Klassen bilden**
  - demnach die **Erzeugungs- und Verhaltensmodelle** für Objekte.
- **Objekte einer Klasse**
  - kennen dieselben Operationen
  - unterscheiden sich in den Werten ihrer Daten
- **Eine Klasse entspricht einem abstrakten Datentyp (ADT)**
- **Ein Objekt entspricht einer Datenkapsel, d.h. die Datenimplementierung ist verborgen.**
- **Von einer Klasse sind außerhalb sichtbar:**
  - Klassenname
  - Exportierte Operationen

# Grundkonzepte der Objektorientierung



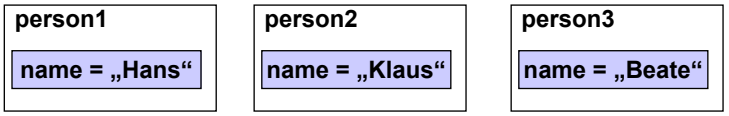



# Attribute & Operationen

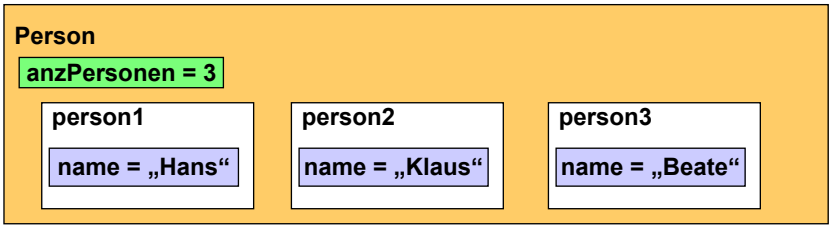
- **Attribute** beschreiben die **Daten** eines Objekts
- **Werte der Attribute** beschreiben den **Zustand** eines Objekts
- **Operationen** arbeiten auf den Attributen
- Operation können den **Zustand** eines Objekts **ändern**
- Jede für ein Objekt deklarierte Operation legt
  - den **Namen** der Operation,
  - die **Argumentobjekte** und die
  - **Rückgabeobjekte** fest.
- Alle zusammen bilden die **Signatur** der Operation.
- Die Menge aller Signaturen der öffentlichen Operationen eines Objekts bildet die **Schnittstelle des Objekts**

# Klassen- vs. Objektattribut

- **Objektattribut: Jedes Objekt speichert einen Wert für das entsprechende Attribut**




- **Klassenattribut (statisches Attribut): Die Klasse speichert einen Wert für das entsprechende Attribut**






# Klassen- vs. Objektmethode

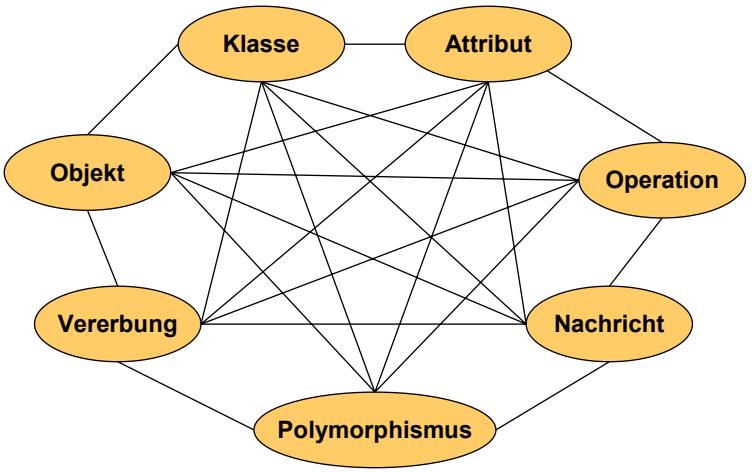
- **Objektmethode**
  - Existiert für jedes Objekt
  - Kann die Objektattribute des einzelnen Objekts verändern
- **Klassenmethoden (statische Methode)**
  - Existiert nur einmal für die Klasse
  - Arbeitet im Allgemeinen auf den Klassenattributen

```
public class Person {
  private static int anzPers = 0;

  public Person() {
    anzPers++;
  }

  public static int getAnzPersonen() {
    return anzPers;
  }
}
```

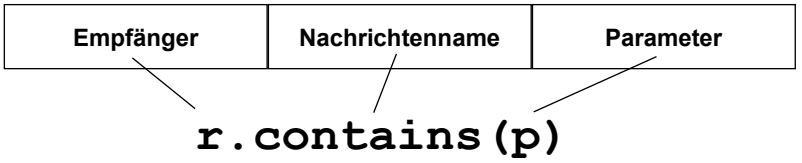
# Grundkonzepte der Objektorientierung






# Nachrichten (Botschaften)

- **Objekte kommunizieren miteinander, indem sie Nachrichten senden und empfangen**
- **Objekte reagieren auf Nachrichten indem sie eine Operation (oder Methode) ausführen.**
- **Der Empfänger einer Nachricht (immer ein Objekt) ist verantwortlich für**
  - **Entschlüsselung** der Nachricht (verstehe ich die Nachricht?)
  - **Wirkung** (was tue ich?)



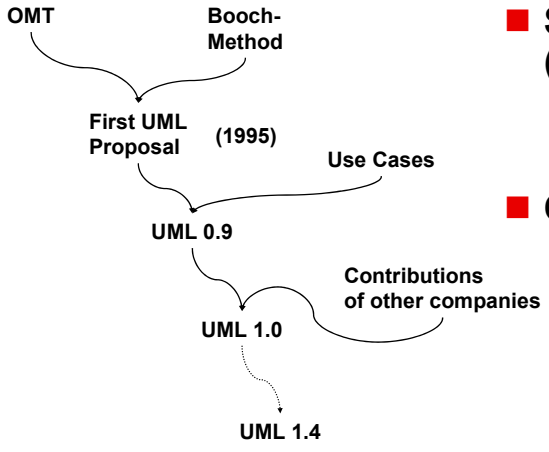
# Notationen

- **Programmiersprachen-unabhängige Beschreibung der Objekttypen (Klassen) und deren Beziehungen**
- **Weit verbreitete grafische Modellierungssprache: Unified Modeling Language UML**
- **Grundlage:**
  - OMT (J. Rumbaugh)
  - Objectory (Use Case basierter Ansatz von I. Jacobson)
  - Methode von G. Booch
- **UML ist nur eine Notation**

➔ **Es wird keine objektorientierte Vorgehensweise dadurch festgelegt**



# Geschichte der UML-Entwicklung




- **Structured Methods (ca. 1980)**
  - Structured Analysis and Design
  - Real-Time Structured Design
- **OO-Methods**
  - Shlaer/Mellor (1988)
  - Coad/Yourdon (1991)
  - Booch-Method (1991)
  - OMT: Rumbaugh (1991)
  - Objectory: Jacobson (1992)

# Grundlagen UML

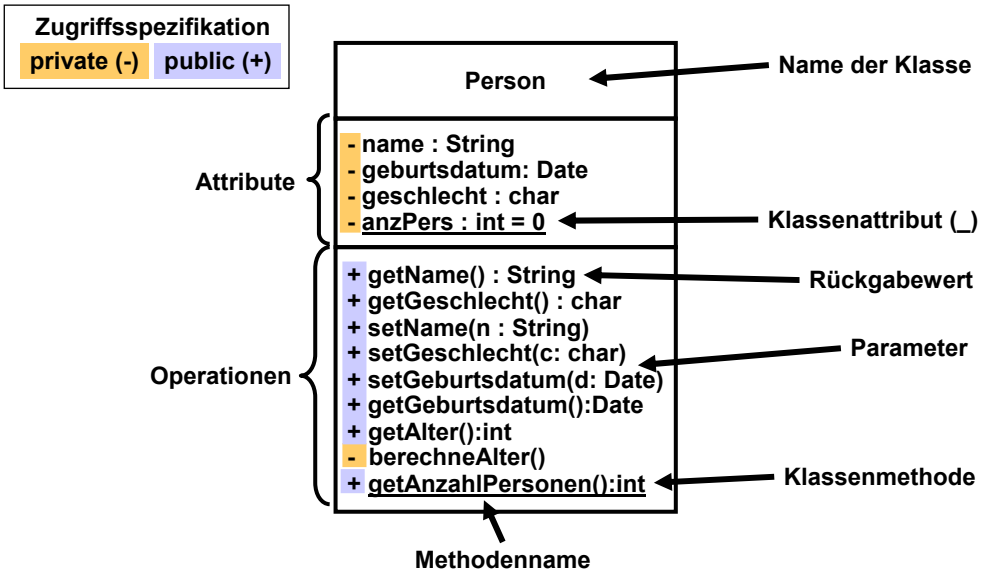
- **UML erlaubt die Modellierung eines Systems aus verschiedenen Sichten**
  - **Statische / strukturelle Sicht**
    - System besteht aus Objekten
    - Objekte und ihre Beziehungen müssen identifiziert werden.
  - **Dynamische Sicht**
    - Beschreibt das Verhalten der Objekte entlang der Zeitachse
    - Beschreibt die Kommunikation zwischen den Objekten.
- **Diagrammarten**
  - Statische Modellierung
    - Class Diagram, Use Case Diagram, Component Diagram, Deployment Diagram
  - Dynamische Modellierung
    - Statechart Diagram, Activity Diagram, Sequence Diagram, Collaboration Diagram



# UML Klassendiagramm

- **Ein Klassendiagramm beschreibt die Statik eines Systems in Form von**
  - Klassen und deren Beziehungen
  - Schnittstellen
  - Paketen und deren Beziehungen.
- **Eine Spezialform des Klassendiagramms ist das Objektdiagramm.**

# Klassendiagramm - Klasse

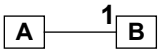




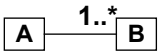

# Klassendiagramm - Beziehungen

- **Allgemeine Assoziation**
  - beschreibt die gemeinsame Struktur einer Menge von (in der Regel) statischen **Beziehungen** zwischen Objekten
  - Spezialfälle
    - Aggregation
    - Komposition
- **Ausprägungen**
  - **Binäre** Assoziation
  - **Bezeichnung** der Assoziation mit "**Leserichtung**"
  - **Rollenangaben** der beteiligten Objekte
  - Angabe der **Kardinalitäten** (Ausprägungshäufigkeit)

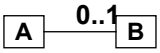
# Kardinalitäten



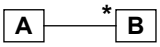

**Genau ein** B-Objekt steht in Beziehung zum A-Objekt




Jedes A-Objekt steht in Beziehung zu **wenigstens** einem B-Objekt

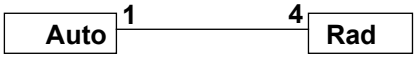



Jedes A-Objekt steht in Beziehung zu **keinem** oder zu **genau einem** B-Objekt




**Keine Einschränkung** bzgl. der Anzahl der B-Objekte

**Beispiel:**



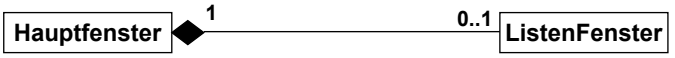

# Aggregation - Komposition

## Aggregation

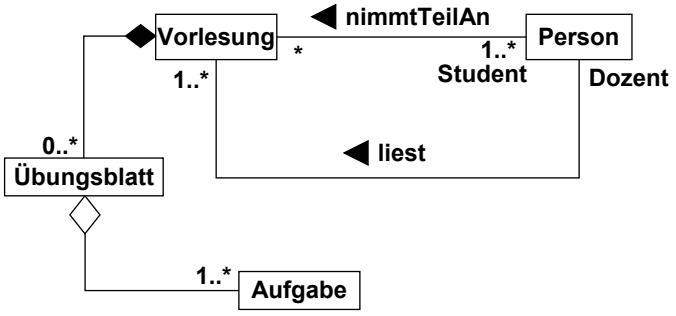
- **Teile-Ganzes-**Beziehung
- Beziehung zwischen gleichwertigen Partnern
  - Beteiligte Objekte sind **unabhängig** voneinander.


## Komposition

- **Restriktive** Form der Aggregation:
  - Ein Objekt kann Teil höchstens **eines** zusammengesetzten Objekts sein.
  - Ein Bestandteil existiert höchstens solange wie das zusammengesetzte Objekt



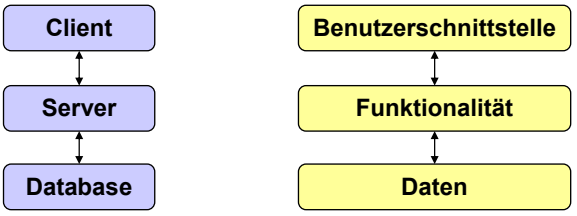
# Klassendiagramm - Beispiel






# Entwurfsprinzipien

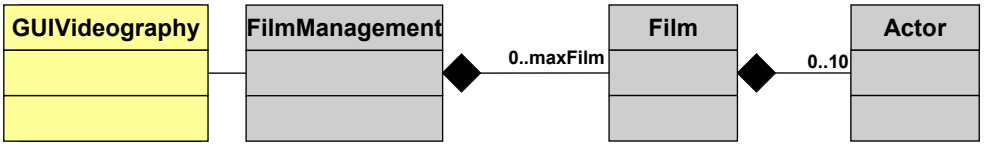
- Entwurfsprinzipien beschreiben bewährte Konzepte, die auf Makroebene zu einem besser modularisierten System führen.
- Häufig eingesetztes Entwurfsprinzip, insbesondere in Informationssystemen: 3-Schichten Architektur




- Kommunikation und Datenaustausch zwischen den Schichten erfolgt über Schnittstellen, die die Interna der Schichten verbergen

# 3-Schichten Architektur

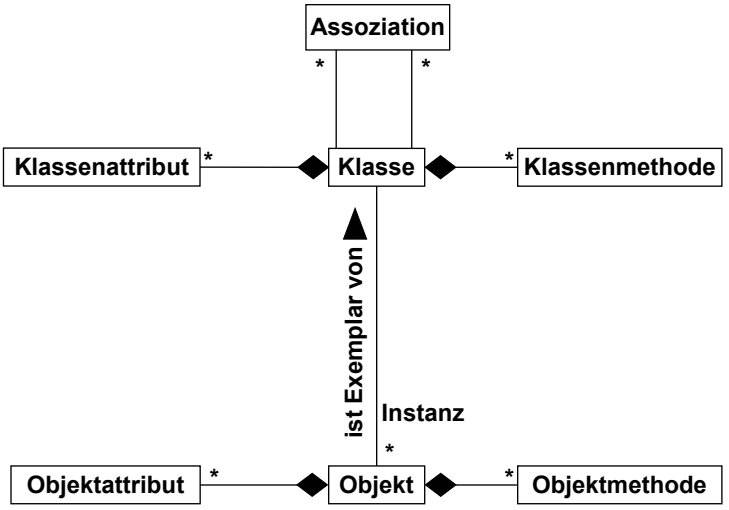
- **Vorteil: Die einzelnen Schichten können gegeneinander ausgetauscht werden, ohne dass Änderungen am übrigen System erforderlich sind**
  - Änderung / Wechsel der Benutzerschnittstelle auf der Clientschicht
  - Austausch von Algorithmen auf der Serverschicht
  - Wechsel der Datenspeicherung (proprietär vs. DBMS) auf der Datenbankschicht
- **Beispiel: Videography aus der aktuellen Übung**






# Demo

# Begriffsnetz Objektorientierung






# Werkzeuge

- **UML Werkzeuge**
  - erlauben, **UML-Diagramme** zu **erstellen**
  - helfen, **Komplexität** von Systemen (Modellen) zu **beherrschen**
  - erlauben teilweise **automatische Code-Generierung**
  - unterstützen **Refactoring**
- **Beispiele:**
  - Rational Rose (http://www.rational.com/)
  - MS Visio (http://www.microsoft.com/germany/ms/visio2003)
  - Poseidon for UML (http://www.gentleware.com)
  - Visual Paradigm for UML (http://www.visual-paradigm.com/vpuml.php)
  - ArgoUML (http://argouml.tigris.org)

# Demo