

Eine kurze Einführung in

# Objektorientierte Analyse und Design mit UML

Michael Brenner

# Analyse? Design?

Ich dachte hier wird programmiert...

- Vom Problem des Kunden zur Software
  - Informell → semi-formal → Programm
- Semi-formale Ebene: UML & Patterns
  - zwischen „zu vage“ und „zu spezifisch“
  - erleichtert Kommunikation
    - mit Kunden
    - mit Domänen-Experten
    - innerhalb des Teams

# Die Unified Modeling Language

- Booch, Rumbaugh, Jacobson (ca. 1995)
- Idee: *einheitliche* (graphische) Notation/Sprache zur *Modellierung* von Software-Systemen
- Nicht: Beschreibung des *Entwicklungsprozesses*
- Versch. Diagrammtypen für versch. Aspekte
  - können und sollen sich ergänzen
  - Analogie Architektur: Grundriss, Außenansicht, Werkpläne für Handwerker, 3D-Modell

# Die verschiedenen Diagramme

- Klassendiagramm
- Use Case
- Interaktionsdiagramme:
  - Sequenzdiagramm
  - Kollaborationsdiagramme
- Package-, Zustands-, Aktivitäts-,  
physikalisches Diagramm

# Design Tools

- Problembeschreibung → Modellierung
  - UML-Editor
- Modell → Programm
  - Automatische Code-Generierung
- Kommerziell: Rational Rose, Together, etc.
- In diesem Kurs: ArgoUML  
(<http://argouml.tigris.org>)

# Klassendiagramme

- **Perspektiven:**
  - **Analyse:** konzeptuelle Perspektive
    - Focus: Zusammenhänge zw. Entitäten der Domäne
  - **Entwurf:** Spezifikation
    - Focus: Schnittstellen zwischen Klassen
  - **Implementation**
    - Focus: konkrete Realisierung der Klassen
- **„Alles hat seine Zeit...“**

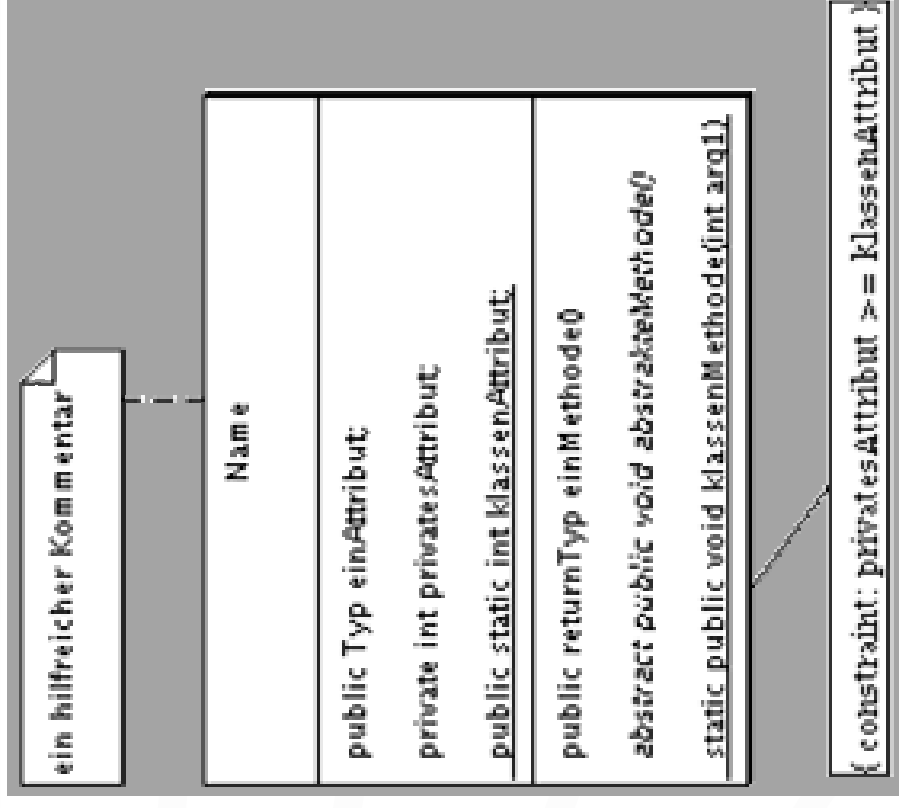
# Die einzelne Klasse

- Klassenname | Attribute | Methoden
- Sichtbarkeit: - (privat), + (öffentlich), # (protected)
- Klassenmethoden/– attribute unterstrichen
- Abstrakte Methoden/ Klassen kursiv oder {abstract}

Name
einAttribut : Typ
-privatesAttribut : int
<u>KlassenAttr : int</u>
eineMethode() : returnType
abstrakteMethode() : void
<u>KlassenMethode() : int</u> ; void

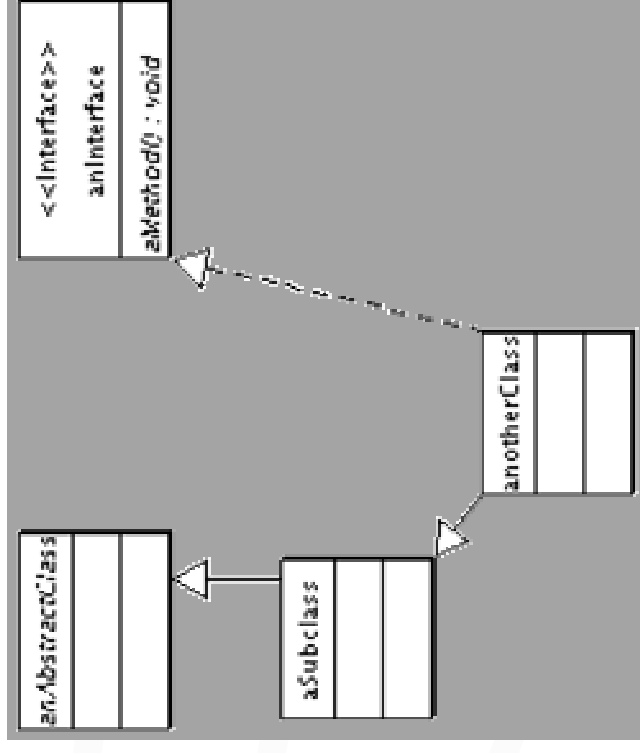
# (An)notationen

- Kommentarfelder
- Constraints in  
{ geschweiften  
Klammern }
- Spezifikations- &  
Implementations-  
Sicht in Java-  
Notation



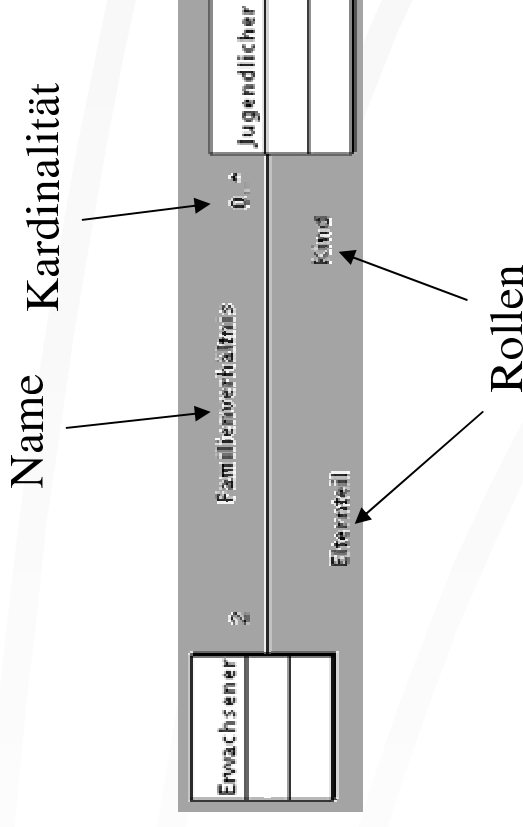
# Klassenhierarchien

- Abstrakte Klassen:  
*kursiv* oder {abstract}
- Interfaces: «interface»
- Generalisierung  
(~ Vererbung): Linie
- Realisierung von  
Interfaces: gestrichelt



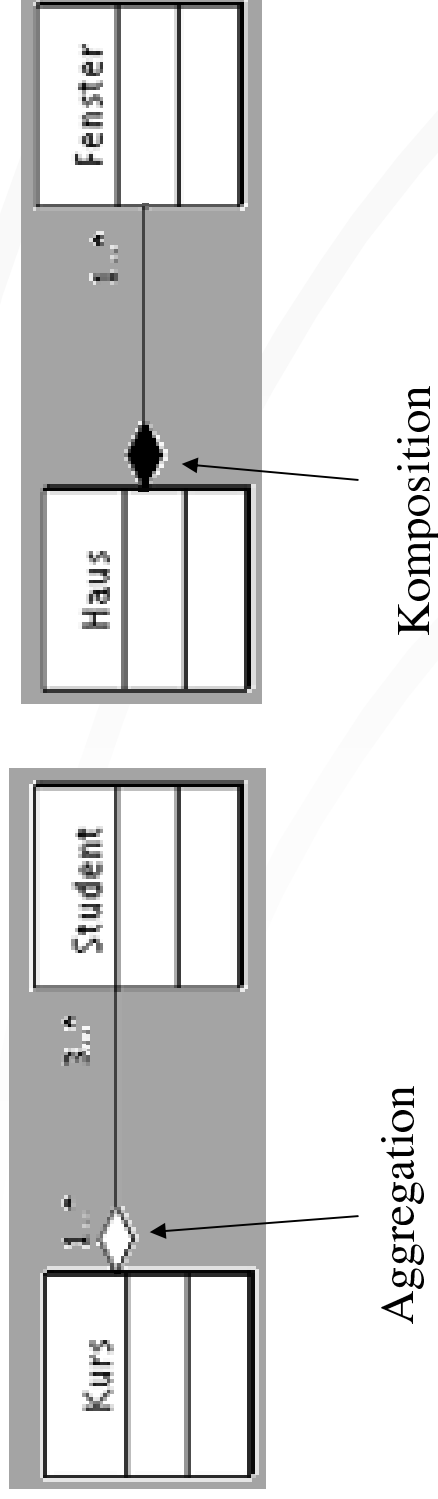
# Andere Relationen zwischen Klassen

- Vererbung ist nicht alles!
- Assoziationen:
  - Name
  - Rollen
  - Kardinalität
  - Navigierbarkeit



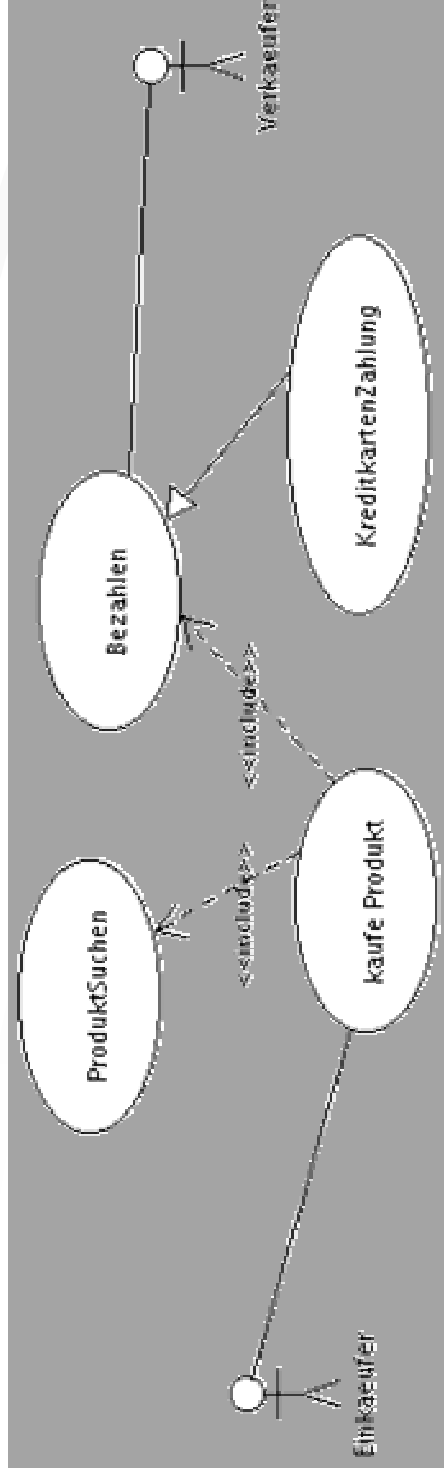
# Erweiterte Assoziationen

- Aggregation: Teil-Ganzes-Relation
  - Unklare Semantik → besser nicht verwenden!
- Komposition: Nur *ein* Ganzes, cascading delete



# Use Cases

- beschreiben typische Interaktionen bei der Benutzung eines Programms
- Szenarien und Aktoren (auch nichtmenschliche)
- Generalisierung und Teil-Ganzes-Relation



# Use Case: Beschreibung

- **Grundzenario: Sequenz nummerierter Schritte**
  1. Kunde durchsucht Katalog
  2. Kunde sucht gewünschtes Produkt im Laden
  3. Kunde verhandelt den Preis
- **Alternativen (v.a. bei Scheitern eines Schritts)**
  - Wenn in Schritt 2 Kunde gewünschtes Produkt nicht findet, dann: Hilfe durch Verkäufer
- **Formaler: Vor- und Nachbedingung für jedes Szenario aufschreiben**

# Interaktionsdiagramme

- modellieren Nachrichten- und Kontrollfluss
- Sequenzdiagramm betont zeitliche Reihenfolge
- Kollaborationsdiagramm zeigt Interaktion zwischen verschiedenen Klassen

# Zusammenfassung UML

- UML ist ein Werkzeug für objektorientierte Domänenanalyse und Design
- viele sich ergänzende Diagrammtypen
- abstrakte Beschreibungssprache zur Kommunikation zwischen Entwicklern, Anwendern, Experten
- Design Patterns:
  - Allgemeine Beschreibung von Entwurfsproblemen und –lösungen
  - semiformale Darstellung: UML, Beispiele, Beschreibung