Abschnitt 9: Objektorientierter Entwurf mit der Unified Modeling Language (UML)

Überblick

Objektorientierte Algorithmenentwicklung

- Wie schon bei der imperativen oder funktionalen Programmierung ist der Entwurf der eigentlichen Algorithmen (und damit die Modellierung der Problemstellung) die entscheidende Herausforderung.
- Den fertigen Entwurf in einer oo Programmiersprache zu implementieren ist dann wiederum relativ trivial.
- Im folgenden schauen wir uns eine Konzeptsprache an, die den oo Entwurf unterstützt: die *Unified Modelling Language* (UML).

Was ist UML?

- UML ist eine Konzeptsprache zur Modellierung von Software-Systemen (insbesondere, aber nicht zwingend: objektorientierter Programme).
- UML ist eine Art Pseudo-Code, der allerdings eine wohl-definierte Semantik besitzt und von vielen Programmen verarbeitet werden kann.
- UML bietet sogar die Möglichkeit, Code-Fragmente oder gesamte Implementierungen anzugeben (z.B. in Java-Notation). Es gibt Tools, die daraus automatisch Java-Code generieren, der (je nach Modellierungstiefe) auch ausführbar ist.
- UML-Code selbst ist nicht ausführbar. Dennoch wird UML von vielen Experten als Prototyp für die nächste Generation von Programmiersprachen betrachtet.

Bestandteile von UML

- Die UML-Notation folgt einer intuitiven Diagramm-Notation.
- Eigentlich umfasst UML ein ganzes System von Konzeptsprachen, d.h. es gibt verschiedene Diagramm-Typen.
- Jeder Diagrammtyp hat seinen eigenen Fokus, z.B.
  - die statische Klassen-Struktur (*Klassendiagramm*, *Class Diagram*),
  - die abstrakte Funktionalität des Programms (*Anwendungsfalldiagramm*, *Use Case Diagram*)
  - die möglichen Zustände und Zustandsübergänge, die ein Objekt einer bestimmten Klasse während seiner "Existenz" einnehmen bzw. ausführen kann (*Zustandsdiagramm*, *State Machine Diagram*)

Klassendiagramme

- Im Rahmen dieser Vorlesung werden wir nur kurz auf Klassendiagramme eingehen und lernen, wie die oo Konzepte in UML modelliert werden.
- Einen tieferen Einblick in UML erhalten Sie in den Vorlesungen zur Software-Entwicklung.
- Klassendiagramme (auch: *Objektdiagramme*) beschreiben die statische Struktur eines Programms. Die konzeptionelle Sicht steht dabei im Vordergrund, die Realisierungsdetails werden meist mit anderen Diagrammtypen beschrieben.
- Klassendiagramme beschreiben im Wesentlichen die Klassen, Objekte und deren Beziehungen zu einander.

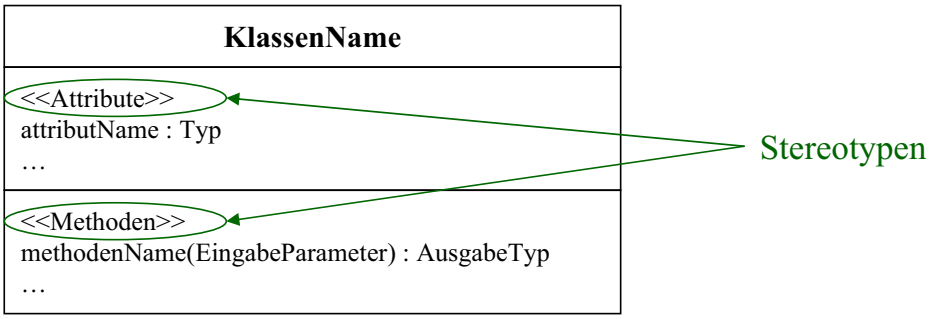
Überblick

Grundlagen

- Wie bereits erwähnt setzen die Konzepte der Klassen und Objekte die oo Grundkonzepte “Abstraktion” und “Kapselung” um.
- Klassen modellieren die gemeinsamen Eigenschaften von Objekten, insbesondere deren Attribute und Methoden.
- Attribute beschreiben den Zustand von Objekten einer Klasse und sollten für andere Benutzer (z.B. Objekte) verborgen sein.
- Methoden beschreiben das Verhalten der Objekte einer Klasse und sollten für andere Benutzer bekannt und verfügbar sein.

Klassen

Allgemeine Form einer Klassendefinition in UML:

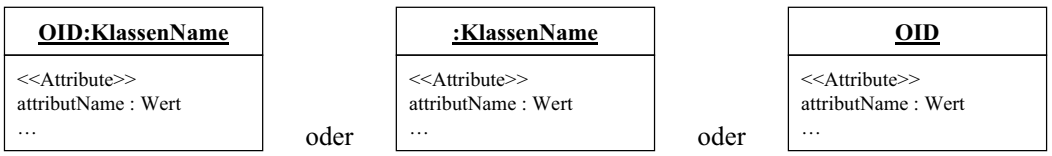


Sichtbarkeit

- Die Sichtbarkeit von Methoden und Attributen muss spezifiziert sein (beachte Kapselung/Abstraktion).
- Die Symbole zur Spezifikation der Sichtbarkeit von Klassen und deren Elementen sind unter anderem:
  - + bzw. public:
    das entsprechende Element (Attribut/Methode) ist von außen (z.B. Objekten anderer Klassen) sichtbar.
  - - bzw. private:
    das entsprechende Element (Attribut/Methode) ist von außen (z.B. Objekten anderer Klassen) NICHT sichtbar.
- Darüberhinaus gibt es weitere Möglichkeiten, die Sichtbarkeit von Elementen einer Klasse zu spezifizieren. Diese lernen wir später kennen.
- **Achtung:** Es gibt in UML (anders als z.B. in Java) *keine* Default-Spezifikation für Elemente einer Klasse, d.h. deren Sichtbarkeit muss immer explizit angegeben sein!

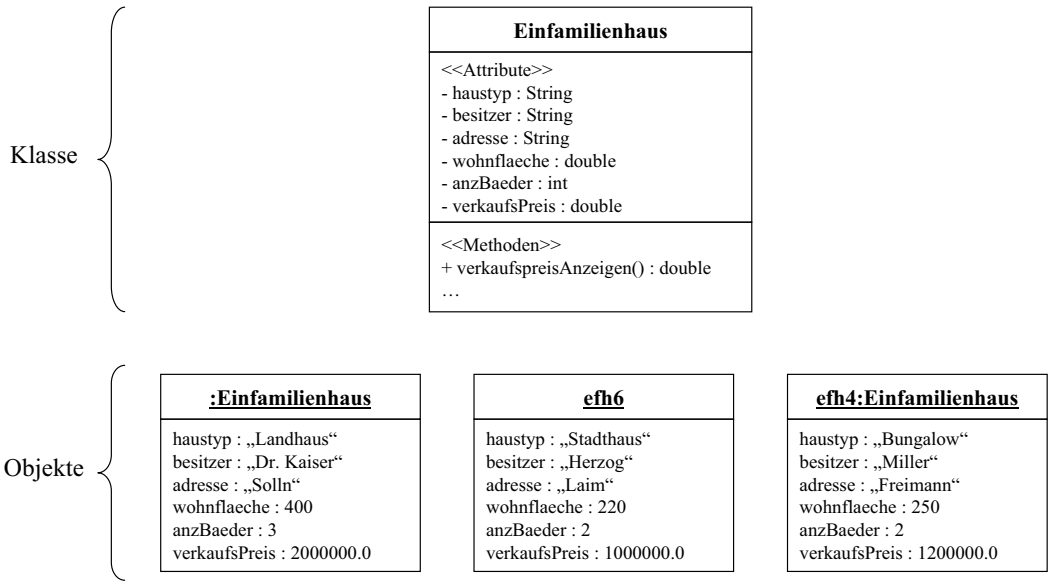
Objekte

Konkrete Objekte einer Klasse werden in UML wie folgt dargestellt:



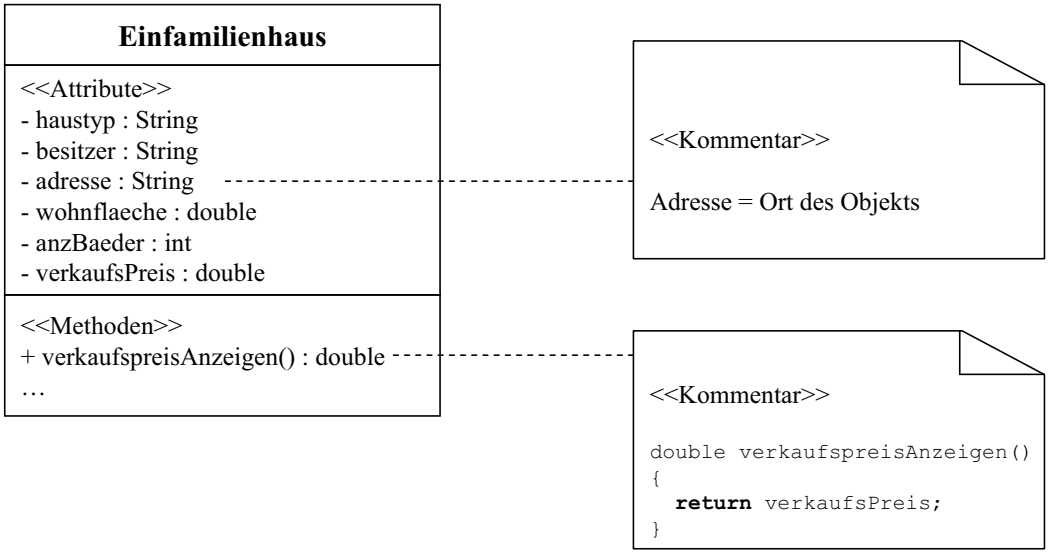
Beispiel: Klassen und Objekte

Ein Beispiel für die Klasse “Einfamilienhaus” mit drei konkreten Objekten.



Kommentare

Kommentare werden in UML wie folgt dargestellt:



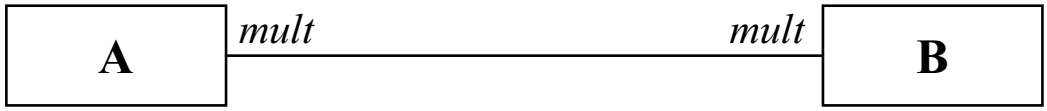
Überblick

Assoziationen

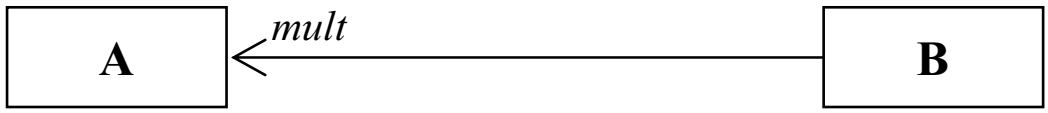
- Objekte verschiedener Klassen existieren nicht isoliert voneinander; es kann die folgenden drei Arten von Beziehungen geben:
  - Verwendungs- und Aufrufbeziehungen,
  - Aggregation und Komposition (“part-of”-Beziehungen),
  - Generalisierung und Spezialisierung (“is-a”-Beziehungen).

Allgemeine Assoziation

- Verwendungsbeziehungen sind die allgemeinste Form von Assoziationen zwischen Objekten verschiedener Klassen.
- Die Situation, dass Objekte der Klasse *A* Objekte der Klasse *B* verwenden und umgekehrt, stellt man in UML wie folgt dar:



- *mult* bezeichnet die *Multiplizität* der Assoziation.
- Die Assoziation kann auch gerichtet werden, z.B. wenn nur Objekte der Klasse *B* Objekte der Klasse *A* verwenden:

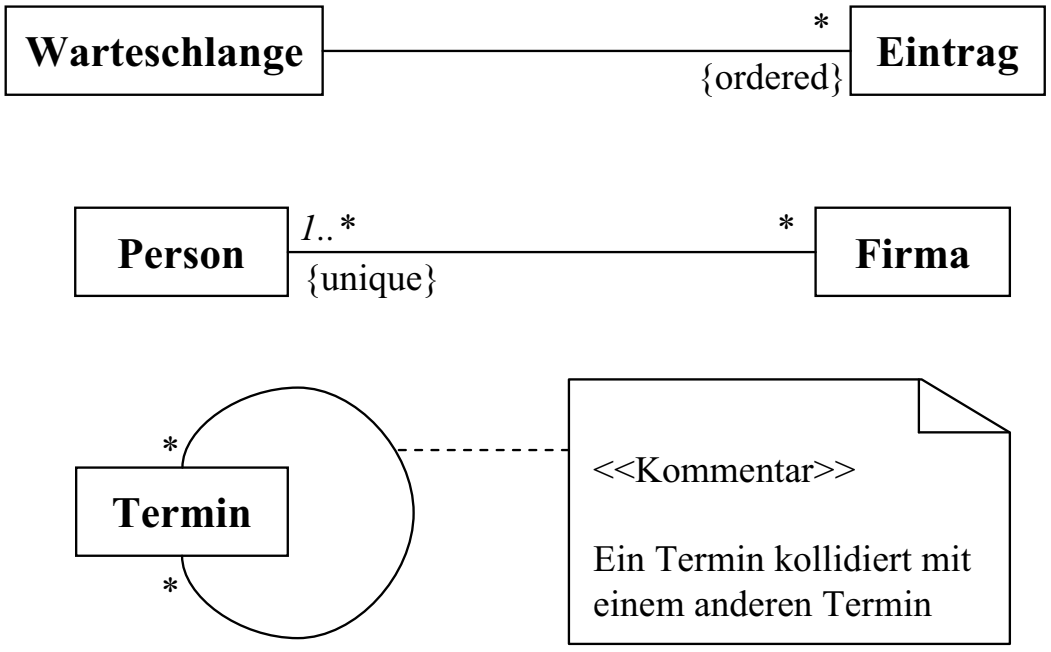


Multiplizität einer Assoziation

Für *mult* kann (u.a.) stehen:

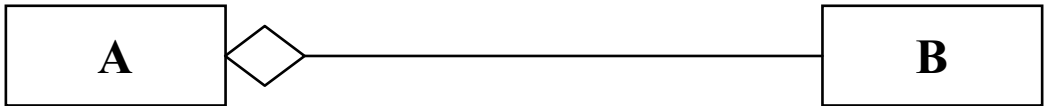
- * beliebig viele,
- *n..m* mindestens *n*, maximal *m*,
- Zusatz {unique}: die verwendeten Objekte sind alle paarweise verschieden,
- Zusatz {ordered}: die verwendeten Objekte sind geordnet (impliziert {unique}).

Beispiel: Assoziationen



Aggregation

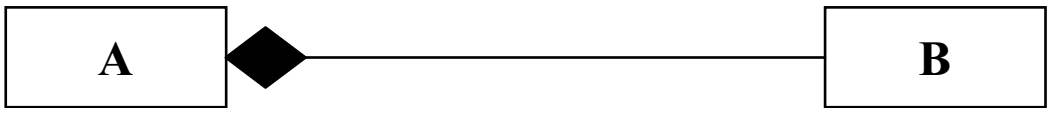
- Die Situation, dass Objekte der Klasse *A* aus Objekten der Klasse *B* zusammengesetzt sind, die Zusammensetzung aber *nicht* essentiell für die Existenz eines Objekts der Klasse *A* ist (*Aggregation*) stellt man in UML wie folgt dar:



- Auch hier kann man Multiplizitäten angeben.

Komposition

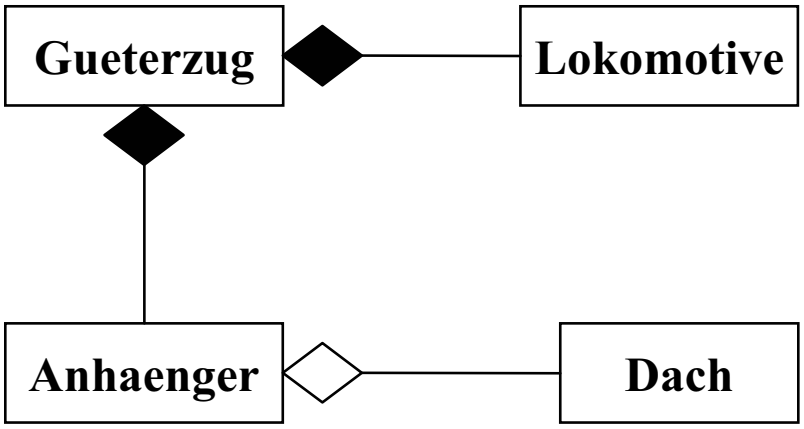
- Die Situation, dass Objekte der Klasse *A* aus Objekten der Klasse *B* zusammengesetzt sind und diese Zusammensetzung essentiell für die Existenz eines Objekts der Klasse *A* ist (*Komposition*) stellt man in UML wie folgt dar:



- Auch hier kann man Multiplizitäten angeben.

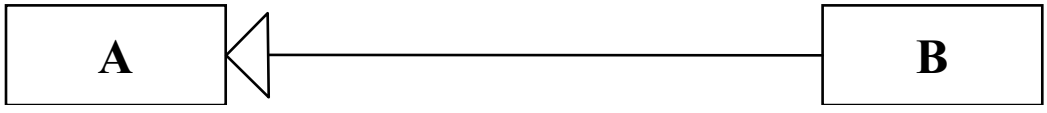
Beispiel: Aggregation und Komposition

Beispiele für Aggregation und Komposition:



Generalisierung/Spezialisierung

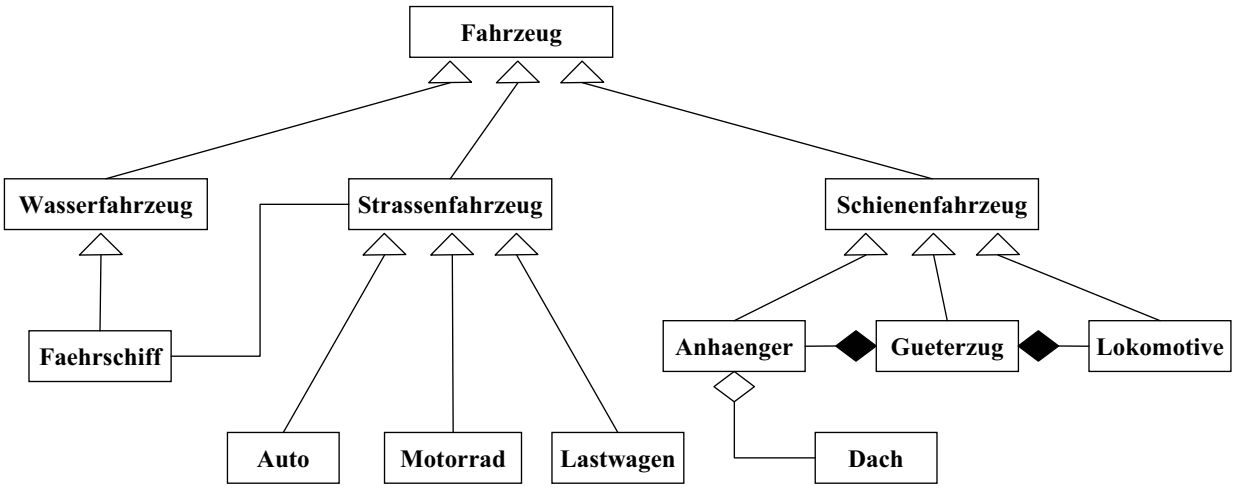
- Eine weitere wichtige Art der Beziehung zwischen Objekten verschiedener Klassen ist die Vererbungs-Relation (Generalisierung/Spezialisierung).
- In UML wird die Vererbungsbeziehung zwischen der Vaterklasse *A* und der abgeleiteten Klasse *B* dargestellt als:



- Insbesondere können nun überall dort, wo Objekte der Klasse *A* vorkommen dürfen, auch Objekte der Klasse *B* vorkommen. Welches oo Modellierungskonzept wird dabei realisiert?

Beispiel: Generalisierung/Spezialisierung

Unser Beispiel von vorher:



Welche grundlegenden oo Konzepte sind hier zu erkennen?