Abschnitt 15: Unified Modeling Language (UML)

Überblick

Was ist UML?

- Die *Unified Modelling Language* (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 Diagramm*)

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 im Hauptstudium.
- 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.

Road Map

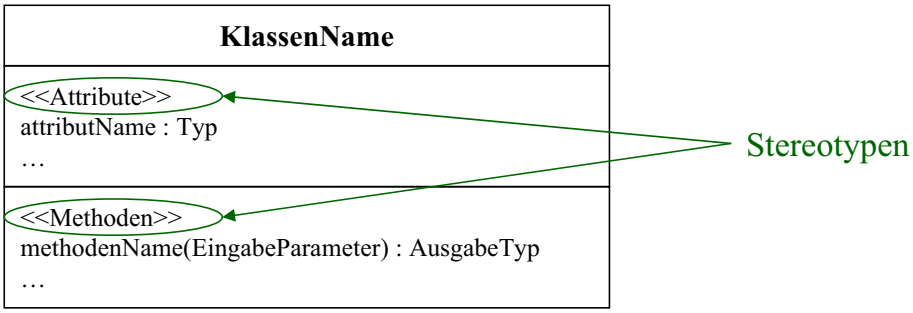
- Insbesondere werden wir im Folgenden noch einmal alle (Java-) Konzepte betrachten, die wir in diesem Teil der Vorlesung diskutiert haben.
- Wir werden uns anschauen, wie diese Konzepte in UML aufgeschrieben bzw. umgesetzt werden werden.
- Wir werden jedes Konzept noch einmal vor dem Hintergrund der 5 Grundideen der oo Modellierung diskutieren:
  - Abstraktion (Zusammenfassen ähnlicher Objekte, Vernachlässigung von Details)
  - Kapselung (Verstecken von Realisierungsdetails)
  - Wiederverwendung (von Code zur Vermeidung von Fehlern und Steigerung der Robustheit von Programmen)
  - Beziehungen (Assoziationen) zwischen den Objekten verschiedener Klassen
  - Polymorphismus

Überblick

Wiederholung

- 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.
- Welche oo Modellierungsaspekte werden durch die Konzepte der Klassen und Objekte realisiert?

Allgemeine Form einer Klassendefinition in UML:



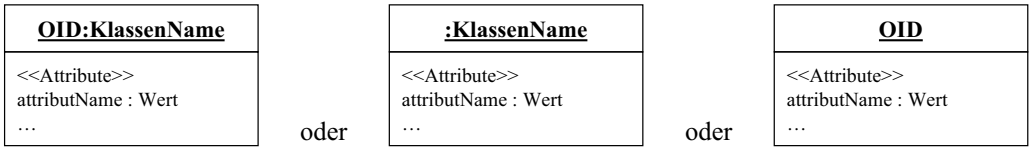
- Die Sichtbarkeit von Methoden und Attributen muss spezifiziert sein (beachte Kapselung/Abstraktion).
- Die Symbole zur Spezifikation der Sichtbarkeit von Klassen und deren Elementen sind Java nachempfunden:

UML-Symbol	**entspricht in Java**
+ bzw. public	`public`
# bzw. protected	`protected`
~ bzw. package	(*default*)
- bzw. private	`private`

- **Achtung:** Es gibt in UML *keine* Default-Spezifikation für Elemente einer Klasse, d.h. deren Sichtbarkeit muss immer explizit angegeben sein!

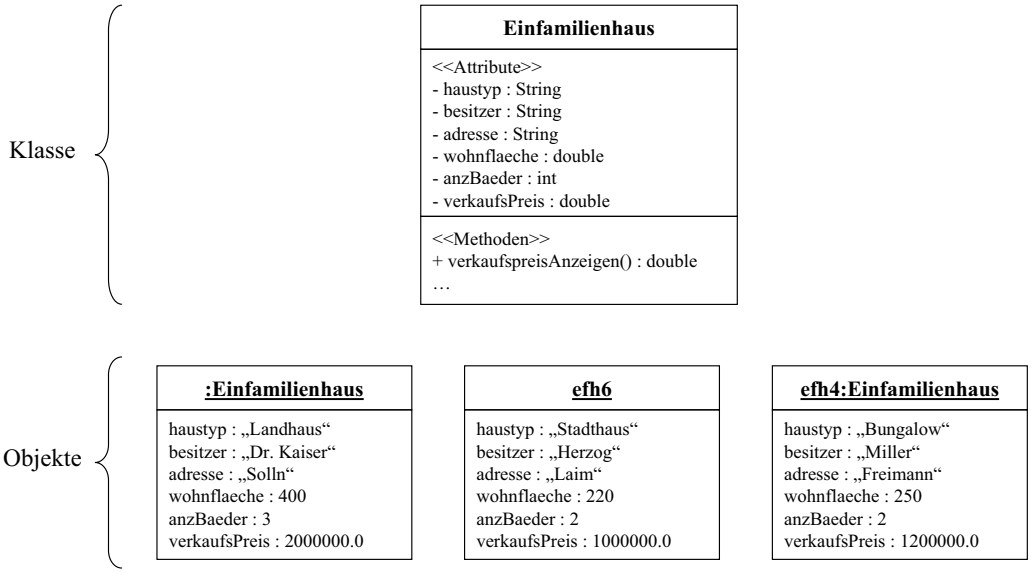
Objekte

Konkrete Objekte einer Klasse werden in UML wie folgt dargestellt:



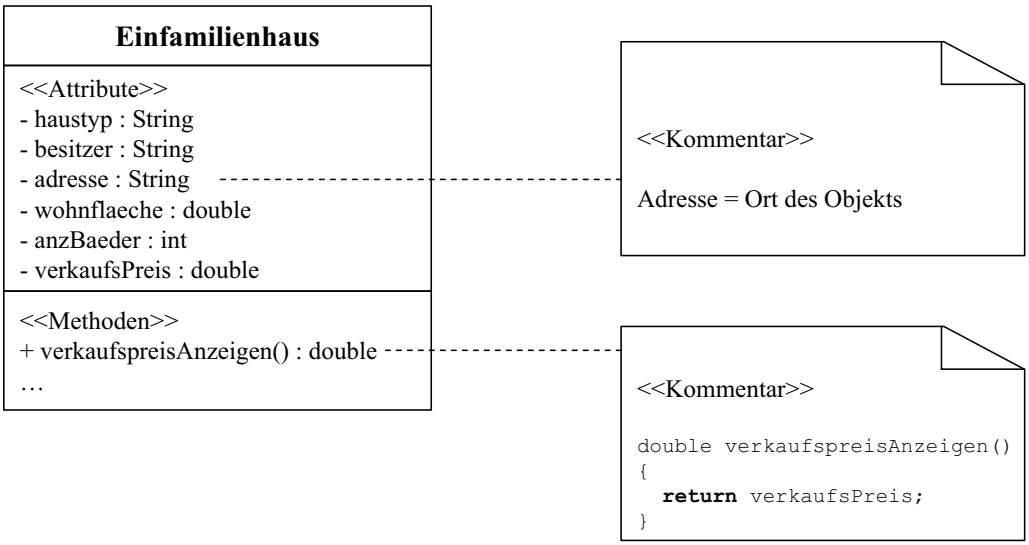
Beispiel: Klassen und Objekte

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



Kommentare

Kommentare werden in UML wie folgt dargestellt:

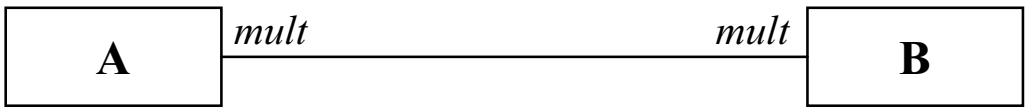


Assoziationen

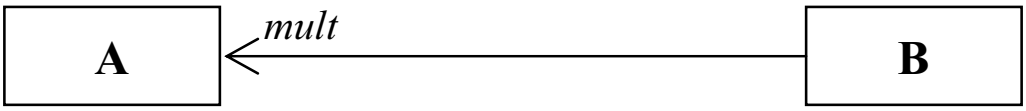
- Wir hatten gesehen, dass Objekte verschiedener Klassen nicht isoliert voneinander existieren, sondern dass es die folgenden drei Arten von Beziehungen geben kann:
  - Verwendungs- und Aufrufbeziehungen,
  - Aggregation und Komposition (“part-of”-Beziehungen),
  - Generalisierung und Spezialisierung (“is-a”-Beziehungen).
- Wir sehen uns zunächst kurz Verwendungsbeziehungen, Aggregation und Komposition an. Generalisierung behandeln wir später in einem eigenen Unterkapitel.

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:

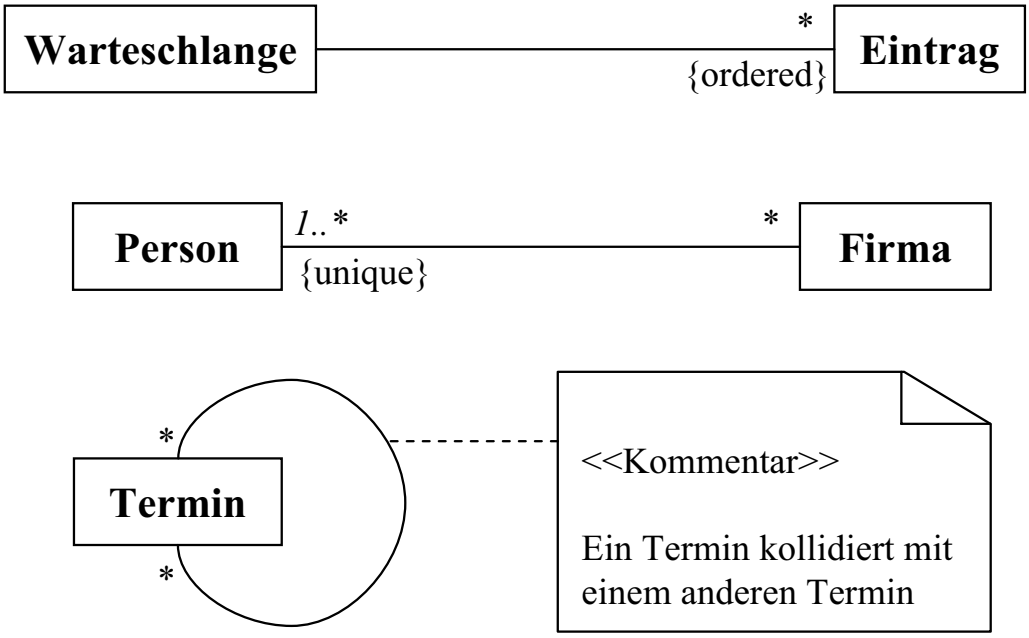


Multiplizität einer Assoziation

Für *mult* kann (u.a.) stehen:

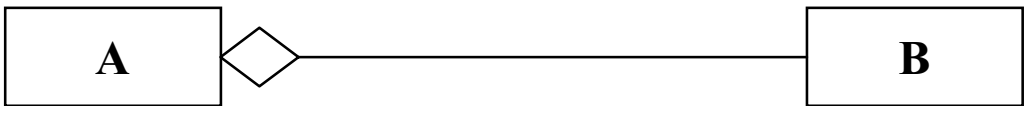
- * beliebig viele,
- *n..m* mindestens *n*, maximal *m*,
- Zusatz {unique}: die verwendeten Objekte sind alle paarweise verschieden,
- Zusatz {ordered}: die verwendeten Objekte sind geordnet (impliziert {unique}).

Beispiel: Assoziationen



Aggregation

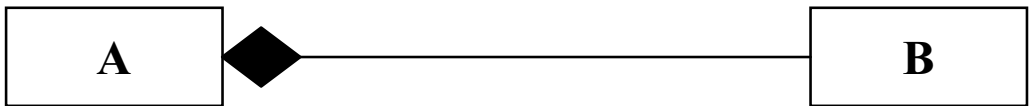
- Die Situation, dass Objekte der Klasse *A* aus Objekten der Klasse *B* zusammengesetzt sind, die Zusammensetzung aber *nicht* essentiell für die Existenz eines Objekts der Klasse *A* ist (*Aggregation*) stellt man in UML wie folgt dar:



- Auch hier kann man Multiplizitäten angeben.

Komposition

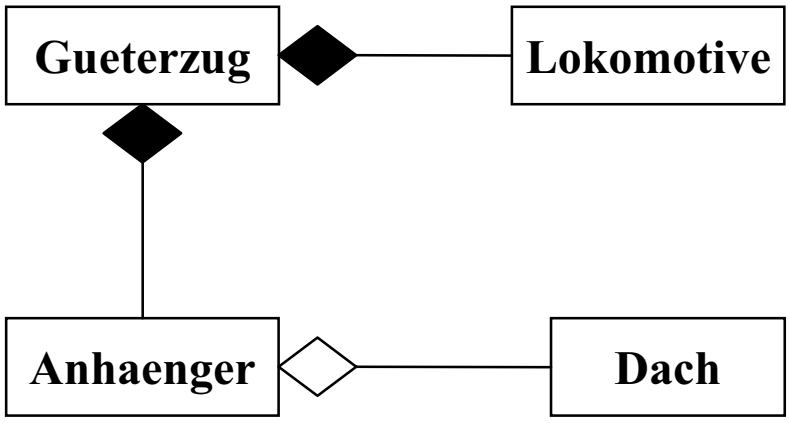
- Die Situation, dass Objekte der Klasse *A* aus Objekten der Klasse *B* zusammengesetzt sind und diese Zusammensetzung essentiell für die Existenz eines Objekts der Klasse *A* ist (*Komposition*) stellt man in UML wie folgt dar:



- Auch hier kann man Multiplizitäten angeben.

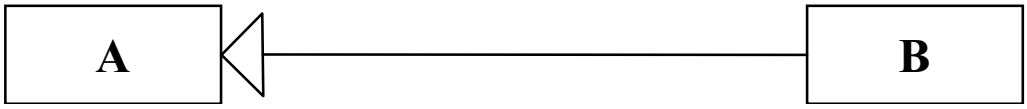
Beispiel: Aggregation und Komposition

Beispiele für Aggregation und Komposition:



Generalisierung/Spezialisierung

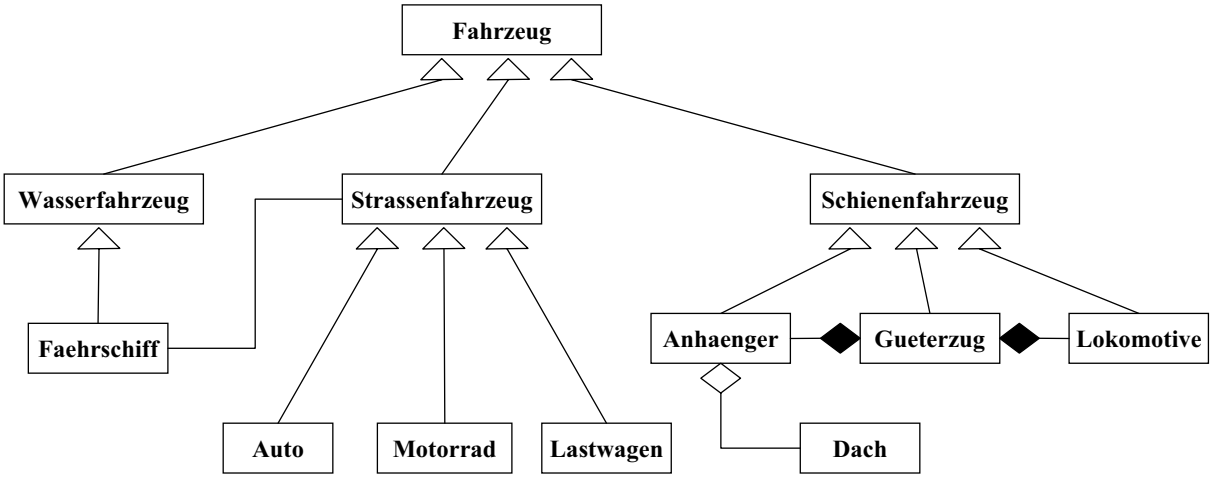
- Eine weitere wichtige Art der Beziehung zwischen Objekten verschiedener Klassen ist die Vererbungs-Relation (Generalisierung/Spezialisierung).
- In UML wird die Vererbungsbeziehung zwischen der Vaterklasse *A* und der abgeleiteten Klasse *B* dargestellt als:



- 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:

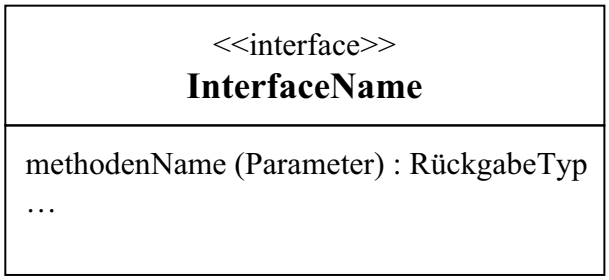


Welche grundlegenden oo Konzepte sind hier zu erkennen?

Überblick

Interfaces

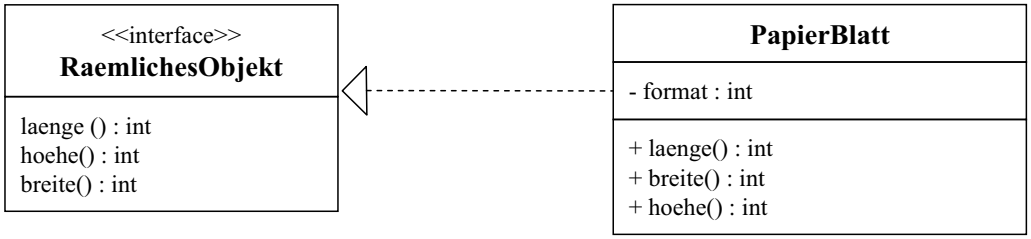
- Ein Interface spezifiziert eine gewisse Funktionalität, ist selbst aber nicht instanziierbar.
- In UML werden Interfaces wie Klassen dargestellt, allerdings wird vor dem Interfacenamen der Stereotyp “«interface»” notiert:



- Welche oo Modellierungsideen werden vom Konzept der Interfaces realisiert?

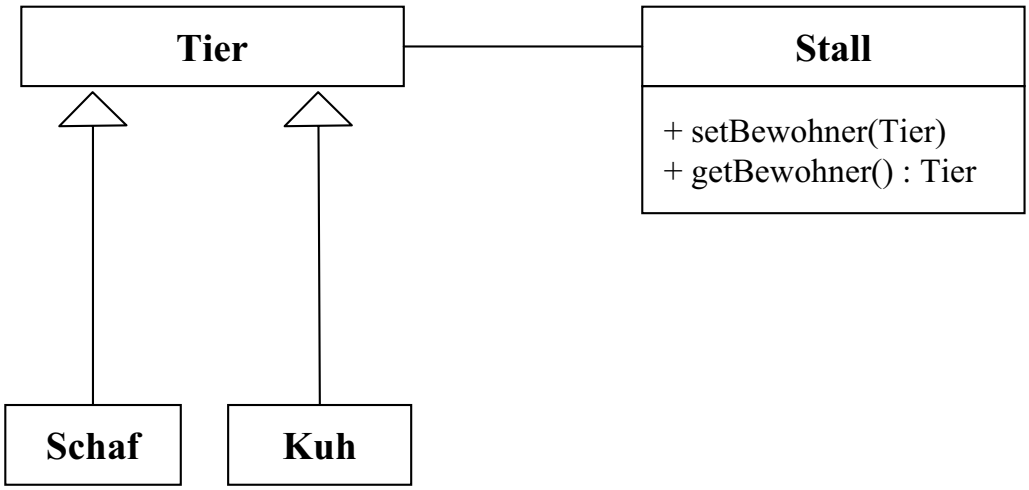
Realisierungsbeziehung

- Implementiert eine Klasse ein Interface, müssen alle Methoden des Interfaces in der Klasse implementiert werden.
- Eine Realisierungsbeziehung zwischen einem Interface und einer (implementierenden) Klasse wird in UML folgendermaßen umgesetzt:



Grenzen der Vererbungs-Polymorphie

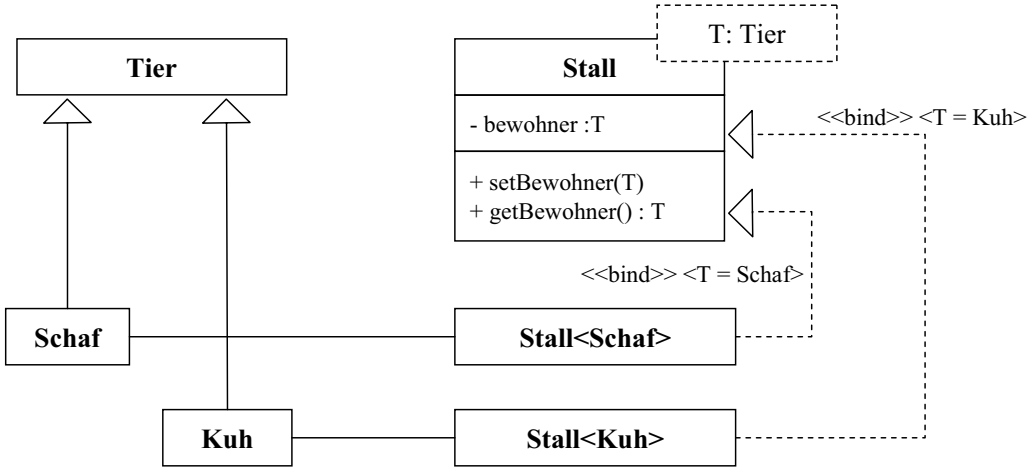
- Gegeben folgendes Beispiel:



- Welche Probleme können bei dieser Modellierung auftreten?

Templates

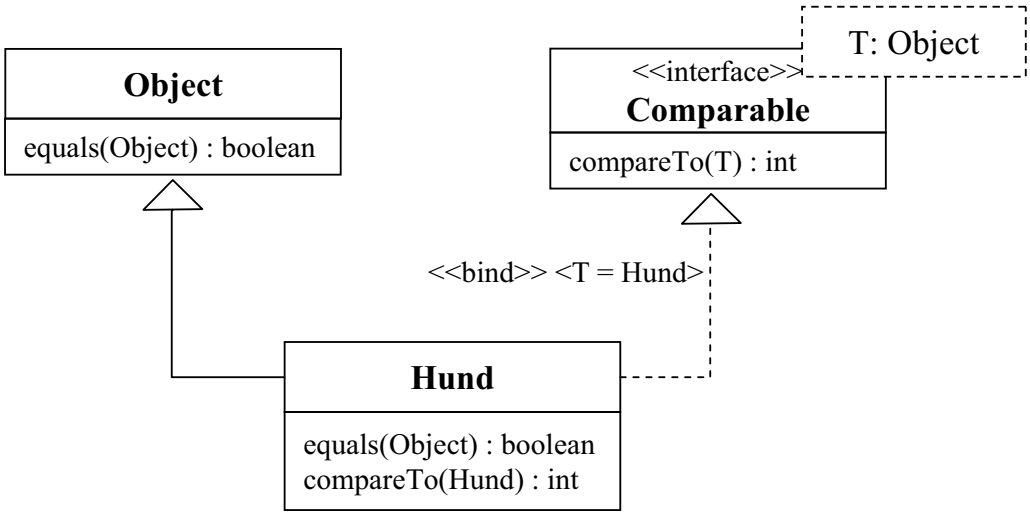
- Als Lösung des Problems haben wir typisierbare Klassen kennengelernt, in Java auch Generics genannt.
- In UML heißen typisierbare Klassen auch Templates. Unser Beispiel von der vorhergehenden Folie wird mit Templates in UML so dargestellt:



- Warum ist unser Problem nun gelöst?

Templates vs. Vererbung

Gegeben folgendes Beispiel:

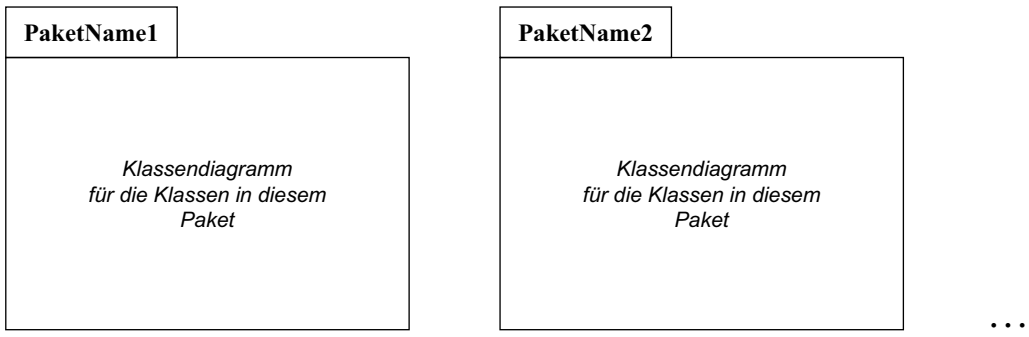


Was ist der Unterschied zwischen der (überschriebenen) Methode “equals” und der (implementierten) Methode “compareTo”?

Überblick

Packages

- Packages gruppieren Klassen, die einen gemeinsamen Aufgabenbereich haben.
- In einem UML Klassendiagramm kann dies folgendermaßen notiert werden:



- Bemerkung: Natürlich kann es Beziehungen zwischen Klassen unterschiedlicher Pakete geben.
- Welche grundlegenden oo Modellierungsaspekte werden durch das Konzept der Pakete realisiert?