TU Darmstadt — Fachbereich Informatik — Fachgebiet Softwaretechnik

Grundzüge der Informatik I — WS 2001/2002

UML-Kurzreferenz

Diese UML-Kurzreferenz orientiert sich im Zweifelsfall nicht an der gängigen UML-Literatur, sondern an der Vorlesung. Inkonsistenzen zwischen den hier vorgestellten UML-Diagrammtypen und in der Literatur zu findenden Beschreibungen sind möglich.

1 Diagramme

1.1 Klassendiagramme

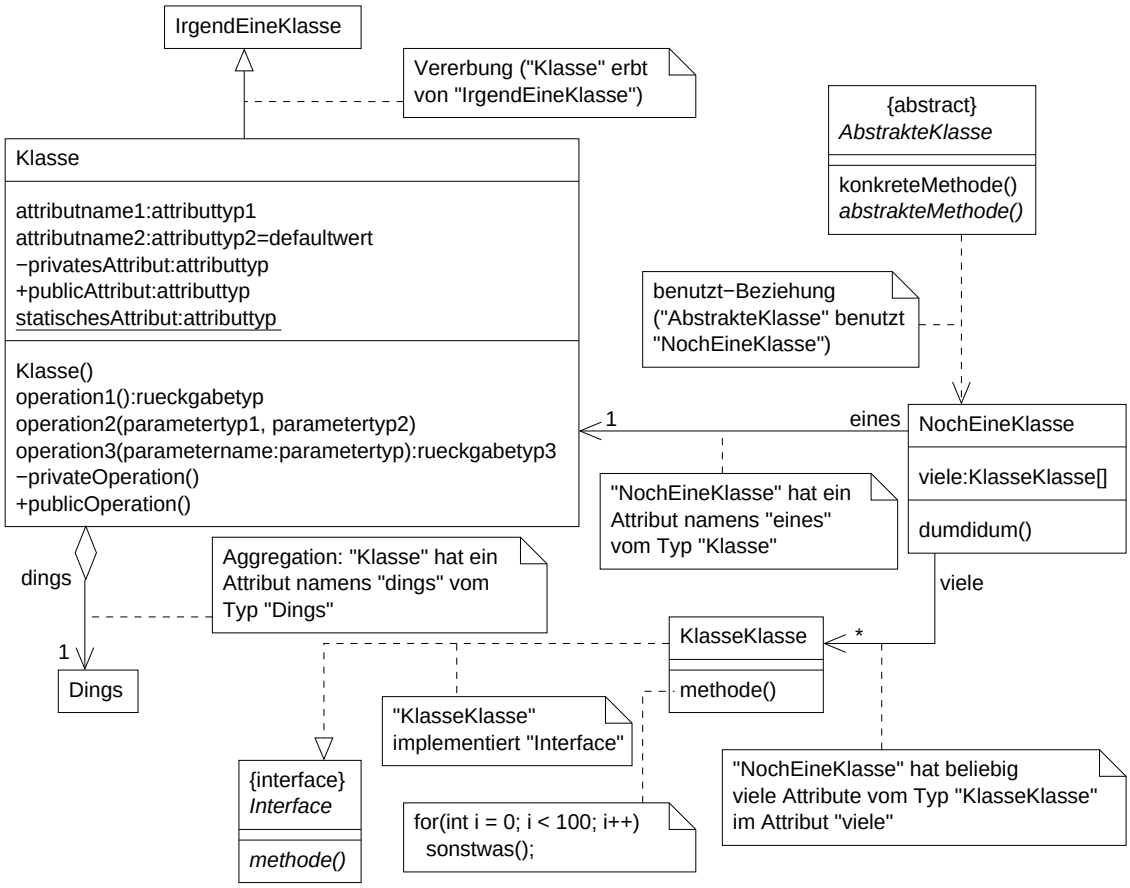


Abbildung 1: Beispiel für ein UML-Klassendiagramm

Abbildung 1 zeigt ein Beispiel für ein UML-Klassendiagramm, das die in der Vorlesung besprochenen UML-Elemente von Klassendiagrammen enthält. Die Boxen mit den „umgeknickten“ rechten oberen Ecken sind Kommentare, können aber auch Implementierungsdetails einzelner Methoden enthalten. Die von ihnen ausgehenden gestrichelten Linien führen zu den Stellen, auf die sich der Kommentar bezieht, bzw. zu der Methode, deren Implementierung im Kommentar steht.

Klassen können in Modulen zusammengefasst werden. Auch UML bietet eine Darstellungsmöglichkeit für solche Gruppierungen. Abbildung 2 zeigt ein Beispiel.

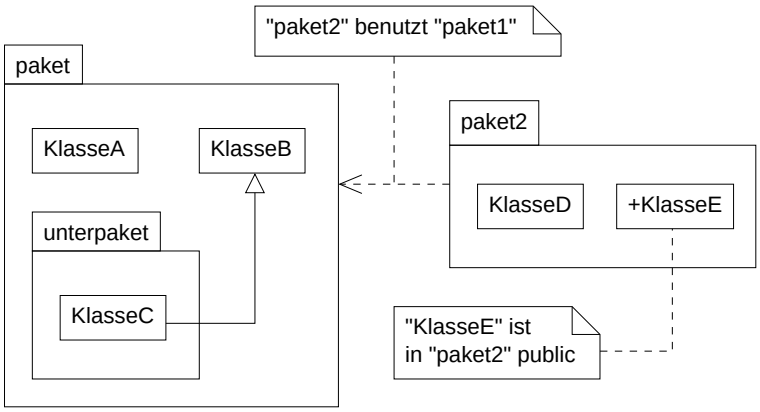


Abbildung 2: Beispiel für ein UML-Klassendiagramm mit Paketen

1.2 Objektdiagramme

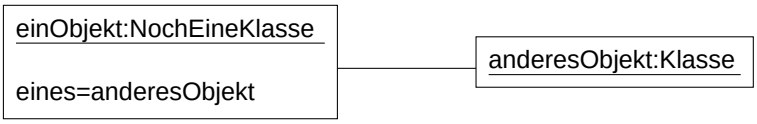


Abbildung 3: Beispiel für ein UML-Objektdiagramm

Objektdiagramme bieten weitaus weniger Möglichkeiten als Klassendiagramme. In Abbildung 3 ist ein einfaches Beispiel für eine Instanz von `NochEineKlasse` namens `einObjekt`, deren Attribut `eines` eine Referenz auf eine Instanz `anderesObjekt` der Klasse `Klasse` enthält. Die Klassen sind Abbildung 1 entnommen.

1.3 Ergänzende Bemerkungen

1.3.1 Vollständigkeit

Im Allgemeinen ist es nicht notwendig, dass ein UML-Diagramm auch wirklich *alle* Einzelheiten der modellierten Klassen enthält. Normalerweise beschränkt man sich bei der grafischen Darstellung auf das Wesentliche, um die Grafik nicht zu „überladen“.

Bestimmte Angaben können dabei auf mehrere Arten gemacht werden:

- **Methodenparameter** können mit oder ohne Namen angegeben werden. Der Typ sollte allerdings auf jeden Fall genannt werden.
- **Attribute** sind auf zwei Weisen darstellbar. Entweder gibt man sie mit Namen und Typ in der Schreibweise `name:typ` innerhalb der Klasse an oder man zeichnet eine Assoziation oder Aggregation (s. u.) und schreibt den Namen des Attributs an diese Assoziation.

1.3.2 Kardinalitäten

Kardinalitäten, also Anzahlen, sind relativ komplex modellierbar. In Abbildung 1 finden sich zwei Beispiele: eine schlichte `1`, die für eine *one-to-one*, und ein Stern (`*`), der für eine *one-to-many*-Beziehung steht.

Damit sind die Möglichkeiten für die Angabe von Kardinalitäten noch lange nicht ausgereizt. Man kann für *one-to-n*-Beziehungen exakte Grenzen angeben, z. B. `2..3`, wenn zwei oder drei Elemente auf einer Seite erlaubt sind. Hier sind beim Modellieren durchaus viele Freiheiten vorhanden.

1.3.3 Assoziation und Aggregation

Die Assoziation wird durch einen einfachen Pfeil, die Aggregation durch einen Pfeil mit Raute am Anfang modelliert. Der Unterschied zwischen diesen beiden Beziehungen besteht darin, dass eine *Assoziation* eine einfache *strukturelle* Beziehung ist – also durchaus auch die Beziehung einer Klasse zu ihren Attributen –, während eine *Aggregation* die Beziehung eines Ganzen zu seinen Teilen darstellt.

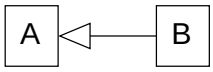
Im Allgemeinen reicht es aus, Assoziationen zu verwenden, es sei denn, man will es zu 120 % richtig machen.

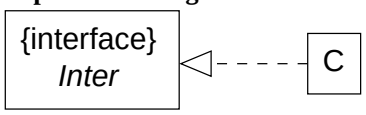
2 UML und Java

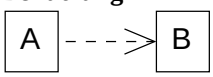
An dieser Stelle sollen einige kurze Erläuterungen zu den Beziehungen von UML und Java gegeben werden. Die Darstellung erhebt keinen Anspruch auf Vollständigkeit.

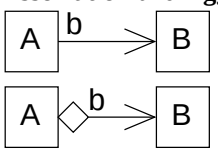
2.1 Beziehungen zwischen Klassen

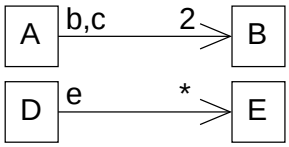
UML	Java

Vererbung
 
	`class B extends A {`
	`  ...`
	`}`

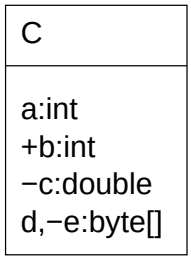
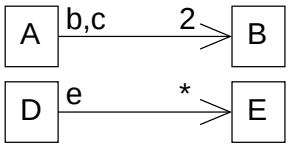
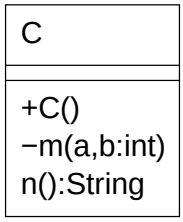
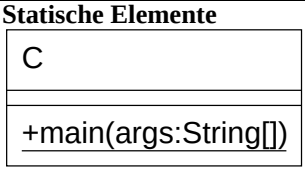
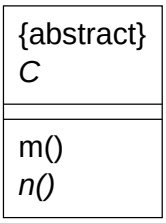
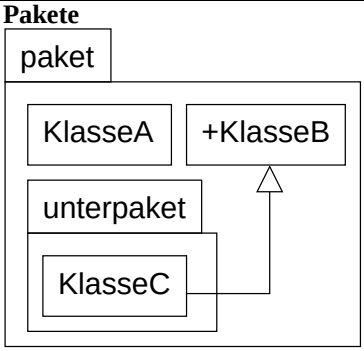
Implementierung eines Interfaces
 
	`class C implements Inter {`
	`  ...`
	`}`

Benutzung 	Diese Beziehung ist sehr vieldeutig. Es kann sein, dass `A` eine Methode von `B` aufruft, oder auch, dass `A` eine Instanz von `B` anlegt etc.

Assoziation und **Aggregation**
 
	Der Unterschied zwischen Assoziation und Aggregation ist eher philosophischer Natur. Bei der Umsetzung in Java hat in jedem Fall die Klasse `A` ein Attribut vom Typ `B` namens `b`:
	`class A {`
	`  B b;`
	`}`

Kardinalitäten
 
	`class A {`
	`  B b, c;`
	`}`
	`class D {`
	`  E e[]; // zum Beispiel`
	`}`

2.2 Klassen, Methoden und Attribute

<table>
<tr><th>UML</th><th>Java</th></tr>
<tr><td>Attribute
 
</td><td><pre>class C {
  int a;
  public int b;
  private double c;
  byte[] d;
  private byte[] e;
}</pre></td></tr>
<tr><td>Attribute einmal anders
 
</td><td>Das Beispiel ist bereits anderweitig verwendet worden.<pre>class A {
  B b, c;
}
class D {
  E e[]; // zum Beispiel
}</pre></td></tr>
<tr><td>Methoden
 
</td><td><pre>class C {
  public C() {...} // Konstruktor
  private m(int a, int b) {...}
  String n() {...}
}</pre></td></tr>
<tr><td>Statische Elemente
 
</td><td><pre>class C {
  public static void main(String args[]) {
    ...
  }
}</pre></td></tr>
<tr><td>Abstrakte Klassen
 
</td><td><pre>abstract class C {
  void m() {...}
  abstract void n() {...}
}</pre></td></tr>
<tr><td>Pakete
 
</td><td>Datei <code>A.java</code>:<pre>package paket;
class A {...}</pre>Datei <code>B.java</code>:<pre>package paket;
public class B {...}</pre>Datei <code>C.java</code>:<pre>package paket.unterpaket;
import paket.B;
class C extends B {...}</pre></td></tr>
</table>