2022

UML2 Klassendiagramme

PROGRAMMIERUNG UND SOFTWAREENTWICKLUNG

1. Überblick

Ein Klassendiagramm stellt die Struktur der zu entwerfenden oder abzubildenden Struktur dar und zeigt dessen wesentlichste statische Eigenschaften sowie ihre Beziehung zueinander. Das Klassendiagramm enthält die grundlegendsten Modellierungssrpache der UML und die wichtigsten grafischen Symbole.

1.1 Modellierung von Klassen

Folgende Notationselemente werden in einem Klassendiagramm in der Praxis dargestellt:

- Klassen
- Attribute
- Operationen
- Assoziationen (mit den Sonderformen:)
    - Aggregation
    - Komposition
- Generalisierungsbeziehungen
- Abhängigkeitsbeziehungen
- Schnittstellen

1.2 Grundlagen der Klassenmodellierung

der Object Modeling Technique (OMT) [Rum93] entwickelt.

Zusätzlich schlägt die Klassenmodellierung die Brücke zur dynamischen Betrachtung des Systems, indem sie die auf den Objekten ausgeführten Operationen berücksichtigt. Jede Klasse kann wahlfrei Operationen definieren, im Verlauf der Implementierung können die einer Operation zugeordneten Methoden ein bestimmtes „Verhalten“ auslösen, das heißt, eine bestimmte Zustandsänderung des Objekts oder eine definierte Reaktion herbeiführen.

Verbindung von Statik und Dynamik

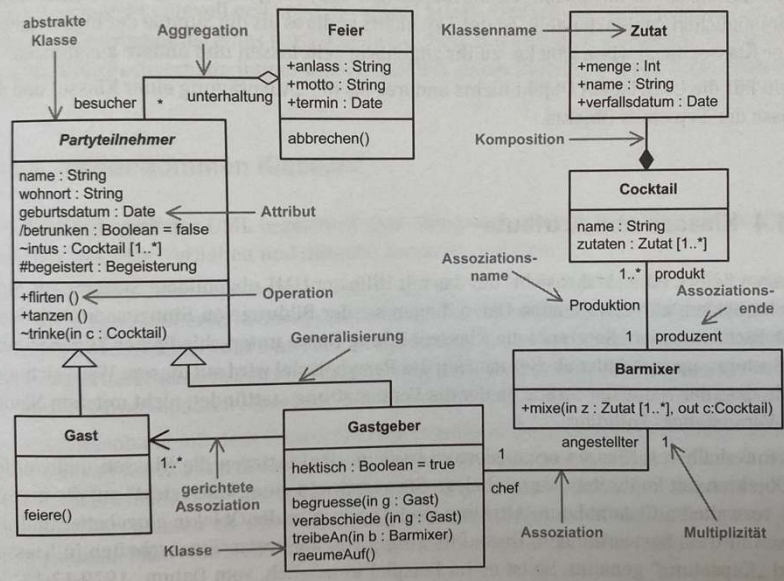
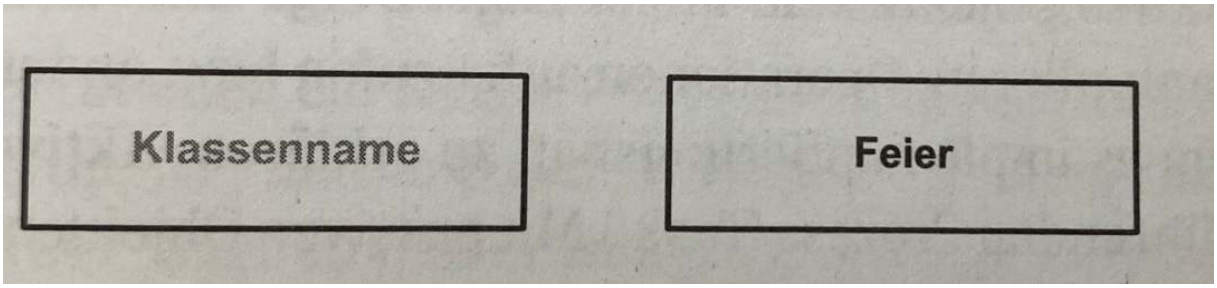


ABBILDUNG 6.1 Elemente des Klassendiagramms

1.3 Notationselemente

1.3.1 Klasse

Eine Klasse wird durch ein Rechteck mit durchgezogener Linie dargestellt, der Name der Klasse steht, in **Fett**druck und mittig, innerhalb des Rechtecks.



Klassen sind das zentrale Element in einem Klassendiagramm. Eine Klasse beschreibt eine Menge von Objekten mit gemeinsamer Semantik, gemeinsamen Eigenschaften und gemeinsamen Verhalten. Die Eigenschaften werden durch Attribute und das Verhalten durch Operationen abgebildet. Diese beschreiben wir in ihrer Bedeutung, und zwar in separaten Abschnitten.

→ Abstrakte Klassen

Abstrakte Klassen sind unvollständig und nicht instanziierbar. Sie dienen nur als Strukturierungsmerkmal, ohne die vollständige Form von Objekten vorzugeben.

Um eine abstrakte Klasse als solche kenntlich zu machen, wird entweder der Klassenname kursiv gesetzt oder das Schlüsselwort abstract in geschweiften Klammern unterhalb des Klassennamens angegeben:

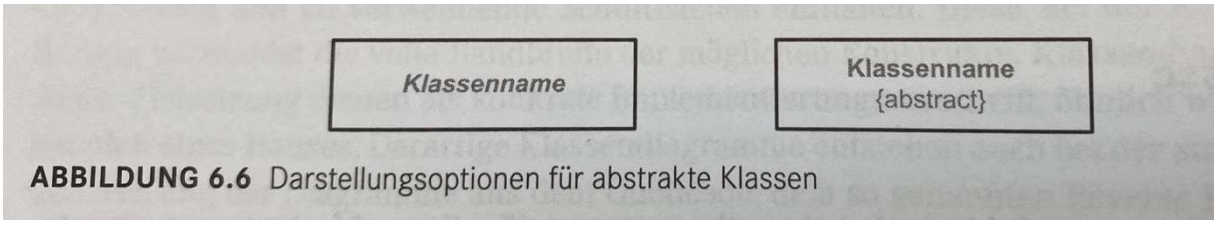


ABBILDUNG 6.6 Darstellungsoptionen für abstrakte Klassen

→ Abschnitte

Um Attribute und/oder Operationen einer Klasse darzustellen, wird das Rechteck unterhalb des Klassennamens durch horizontale Linien in je einen weiteren Abschnitt (engl. Compartment) abgeteilt, wobei typischerweise der zweite Abschnitt die Attribute und der dritte Abschnitt die Operation enthält:

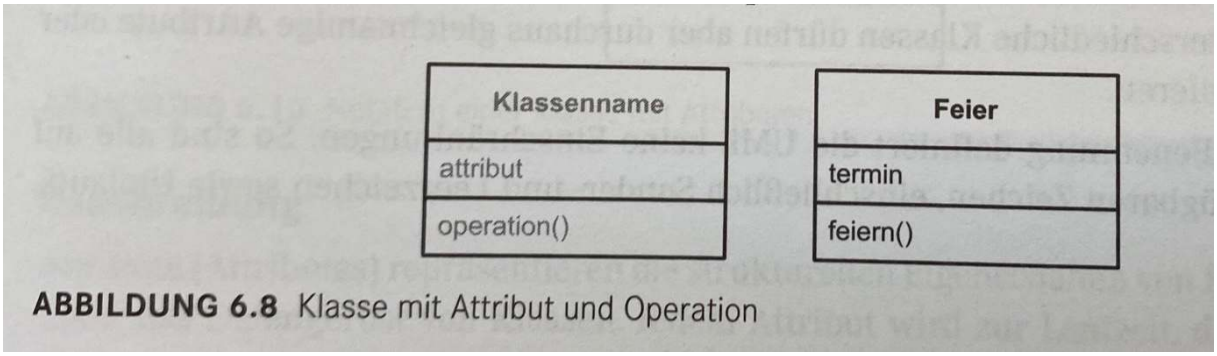


ABBILDUNG 6.8 Klasse mit Attribut und Operation

1.3.2 Attribute

Attribute werden üblicherweise im zweiten Abschnitt/Rechteckelement des Klassensymbols linksbündig untereinander angeordnet. Attributnamen werden kleingeschrieben und Klassenattribute unterstrichen.

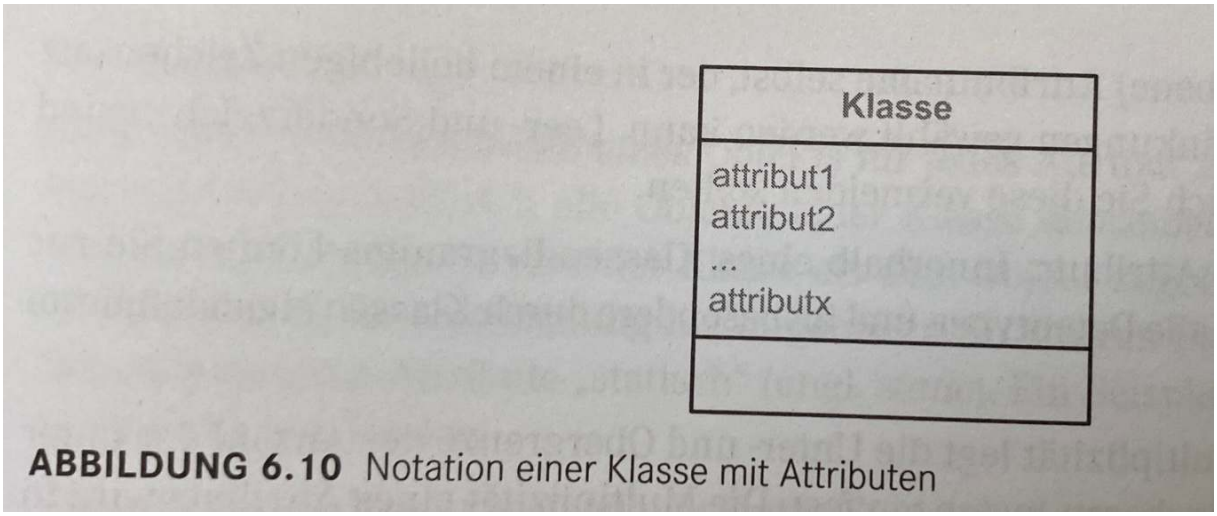


ABBILDUNG 6.10 Notation einer Klasse mit Attributen

Attribute repräsentieren die strukturellen Eigenschaften von Klassen. Sie bilden somit das Datengerüst von Klassen. Jedem Attribut wird zur Laufzeit, d.h. im Objekt der Klasse, ein entsprechender Wert zugewiesen – es wird befüllt.

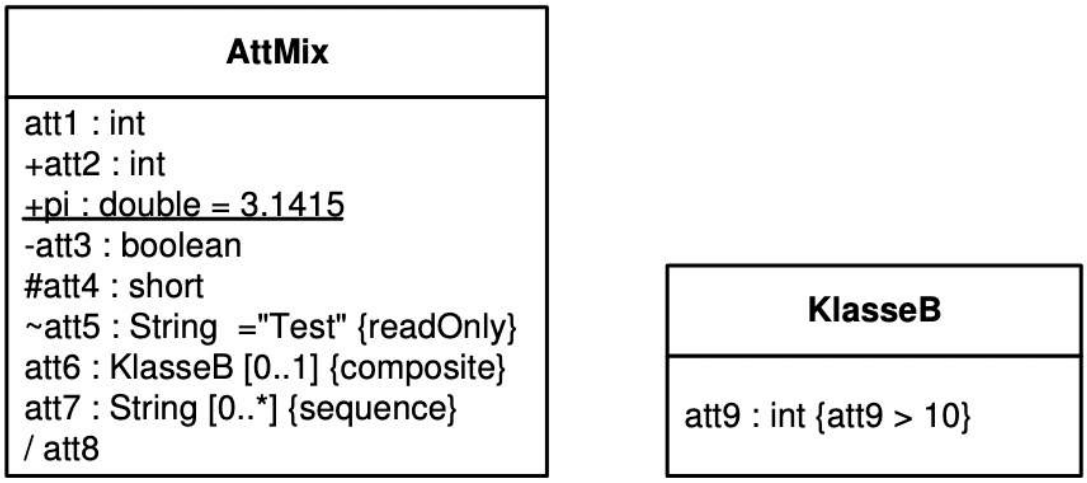
→ Attributdeklaration

[sichtbarkeit] [/] name [: Typ] [[Multiplizität]] [= Vorgabewert] [{eigenschaftswert [, eigenschaftswert]* }]

- sichtbarkeit: definiert, welchen Sichtbarkeits- und Zugriffsrestriktion ein Attribut unterworfen ist, d.h. welche anderen Systemteile das Attribut „sehen“ können bzw. seinen Wert lesen und schreiben dürfen. Vier verschiedene Sichtbarkeitsmodi werden unterschieden:
    - public (+): Jedes andere Element hat uneingeschränkten Zugriff.
    - private (-): Nur Instanzen er Klasse, die das Attribut definiert, dürfen zugreifen.
    - protected (#): Nur Instanzen ist für alle Elemente, die sich im selben Paket wie die definierende Klasse befinden, sicht- und zugreifbar.
    - package (~): Das Attribut ist für alle Elemente, die sich im selben Paket wie die definierende Klasse befinden, sicht- und zugreifbar.
    - / bedeutet, dass es sich um ein sog. „abgeleitetes Attribut“ handelt, dessen Inhalt aus anderen im System vorliegenden Daten zur Laufzeit berechnet werden kann.
- name: der kleingeschriebene Attributname selbst.
- Typ: der Datentyp eines Attributs. Innerhalb eines Klassendiagramms können Sie zur Typisierung der Attribute alle Datentypen und insbesondere durch Klassen eigendefinierte Typen verwenden.
- Miltipilizität: Die Multiplizität legt die Unter- und Obergrenze der Anzahl der unter einem Attributnamen ablegbaren Instanzen fest. Die Multiplizität eines Attributs wird in eckigen Klammern notiert. Jede Multipilizitätsangabe besitzt eine Untergrenze, die 0 oder eine beliebige natürliche Zahl sein kann, sowie eine durch zwei Punkte „...“ abgetrennte Obergrenze, die entweder eine beliebige natürliche Zahl größer oder gleich der Untergrenze sein kann oder das Symbol „*“ für eine nach oben nicht festgelegter Anzahl. Wenn Sie für ein Attribut keine Multiplizität angeben, so wird 1..1 als Vorgabewert angenommen.
  Häufig verwendete Multiplizitäten und ihre Bedeutung:
    - 0..1 : optionales Attribut, d.h. im Objekt ist für das Attribut höchstens ein Wert angegeben
    - 1..1 : bzw. alternativ 1: zwingendes Attribut, d.h. im Objekt wird genau ein wert angegeben.
    - 0..* : optional beliebig, d.h. beliebig viele. Elementanzahl kann auch null sein.
    - 1..* : beliebig viele Werte, aber mindestens einer.
- Vorgabewert: erlaubt die Angabe eines festen oder berechenbaren Ausdrucks mit einem Wert, der für das entsprechende Attribut bei Erzeugung des Objekts der Klasse automatisch gesetzt wird.
- Eingenschaftswert: mit dieser in geschweiften Klammern eingeschlossenen Angabe können Sie besondere Charakteristika des Attributs benennen. Zum Beispiel:
    - readOnly : für Attribute, deren Wert nicht verändert werden darf (Konstanten Schlüsselwort: **final**). pi : Real = 3.1415 {readOnly}
    - orderen : legt fest, dass die Inhalte eines Attributs in geordneter Reihenfolge auftreten, z.B. {1,1,3,5,6,6,9,24}

    - unique : legt fest, dass die Inhalte eines Attributs duplikatfrei auftreten z.B. {3,1,500,4,2,16,89,24}
    - id : legt fest, dass das Attribut Teil des Identifiers der Klasse ist.
- Klassenattribut: Klassenattribute werden durch Unterstreichung hervorgebracht. Schlüsselwort **static**.

Beispiel:



Umsetzung in Java:

```
class AtMix{
int att1;
public int att2;
public static double pi = 3.1415;
private boolean att3;
protected short att4;
final String att5 = “Test”;
KlasseB att6;
java.util.Vector<String> att7;

Object getAtt8(){
        // Berechne wert für att8
        return wert;
        }
}
```

1.3.3 Operation

Eine Operation wird immer durch mindestens ihren Namen sowie weitere Angaben spezifiziert. Mehrere Operationen werden dabei vertikal untereinander linksbündig typischerweise im dritten Abschnitt/Rechteckelement des Klassenysmbols angeschrieben. Abstrakte Operationen sind kursiv gesetzt; Klassenoperationen werden unterstrichen. Der Name der Operation wird im Allgemeinen klein gesetzt.

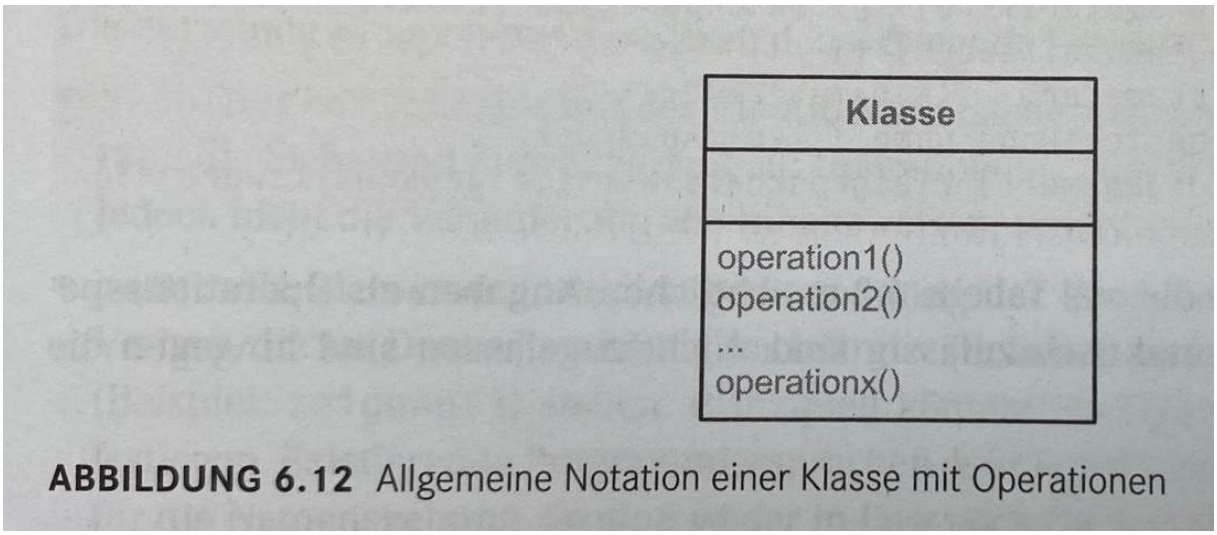


ABBILDUNG 6.12 Allgemeine Notation einer Klasse mit Operationen

Operationen werden eingesetzt, um das Verhalten von Objekten zu beschreiben. Alle Operationen einer Klasse zusammengenommen, definieren die Möglichkeit zur Interaktion mit Objekten dieser Klasse. Gleichzeitung stellen Operationen die einzige Möglichkeit dar, Zustandsänderungen eines Objekts herbeizuführen.

Eine Operation beschreibt die Signatur (gebildet aus dem Operationsnamen sowie den Über- und Rückgabeparametern) einer sie implementierenden Methode.

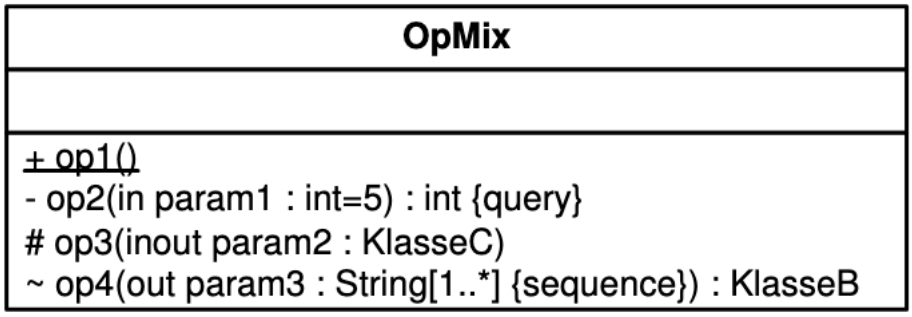
Eine Methode ist die Implementierung der Operation. Operationen werden auf Instanzen von Klassen (häufig) oder der Klasse selbst (selten) ausgeführt.

Tabelle: Syntaktisch zugelassene Operationsdeklarationen

Operationsdeklaration	Anmerkung
halloWelt()	Eine Operation ohne Übergabe- und Rückgabeparameter
+add(summand1, summand2) : Int	Öffentlich zugreifbare Operation mit zwei Übergabe- und einem Rückgabeparameter; der Typ der Übergabeparameter ist nicht festgelegt.
-div(divident : Int, Divisor : Int) : double	Private Operation mit zwei Übergabeparametern vom Typ int und einem Rückgabeparameter
-	-

setLstKl(lohnsteuerklasse : int = 1)	Die Operation definiert, dass, wenn der Parameter Lohnsteuerklasse keinen Wert besitzt, automatisch die Vorgabe 1 angenommen wird
sub(in minuend : double, in subtrahend : double, out resultat : double)	Operation mit drei Übergabeparametern, wobei zwei lesend als Eingabe verarbeitet werden. Der dritte (mit out deklarierte) enthält das Ergebnis.
Inc(Inout wert)	Operation, deren Parameter im Rahme der Ausführung verändert und an den Aufrufer zurückgegeben wird.
mult(summand : Matrix [2..*] {ordered}) : Matrix	Operation, die einen Übergabeparameter besitzt, der eine Menge von mindestens zwei Werten des Typs Matrix beherbergt. Die Angabe der Eigenschaft ordered bestimmt, dass die Elemente der Menge in der übergebenen Reihenfolge verarbeitet werden müssen. Der Rückgabe ist eben falls vom Typ Matrix

Beispiel:



Umsetzung in Java-Code:

```
public class OpMix{
	public static op1(){
		//Implementierung
	}
	private int op2(final int param1){
		//Implementierung
		return wert;
	}
	protected op3(KlasseC param2){

	}
	KlasseB op4(String[] param3){
		//Implementierung
		return wert;
	}
```

Schnittstelle

In der Standardnotation wird die Schnittstelle mit einem Klassensymbol dargestellt, welches mit dem Schlüsselwort interface versehen ist.

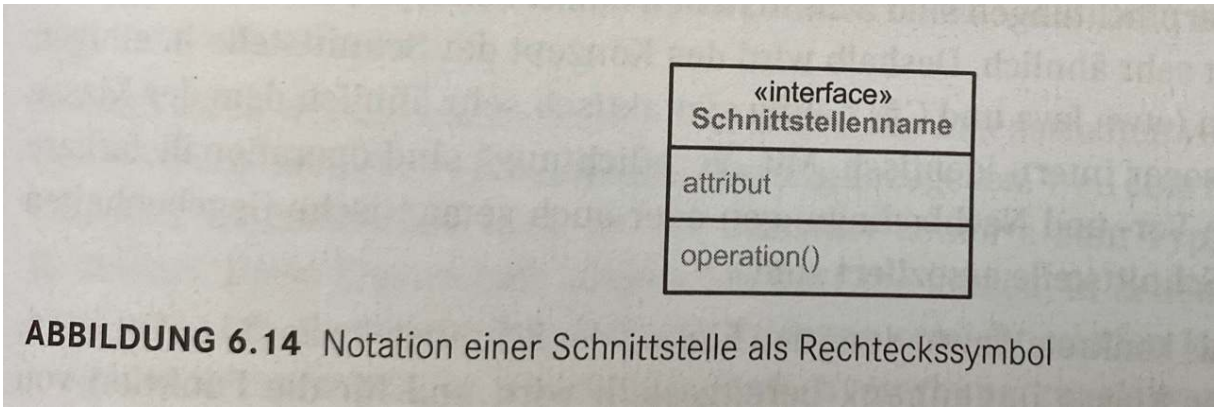


ABBILDUNG 6.14 Notation einer Schnittstelle als Rechteckssymbol

Element1 aus Abbildung 6.15 definiert durch das große, nicht ausgefüllte Kreissymbol (in der UML als Ball bezeichnet), dass es eine gegebene Schnittstelle implementiert. Wegen ihres Aussehens wird diese Notation auch Lollipop-Darstellung genannt. Rechts daneben ist die gleichwertige Notation zu sehen, die die Pfeildarstellung für die Implementierungsbeziehung benutzt. Der Pfeil mit dem leeren Dreieck am Ende weist vom implementierenen Classifier auf die Schnittstelle.

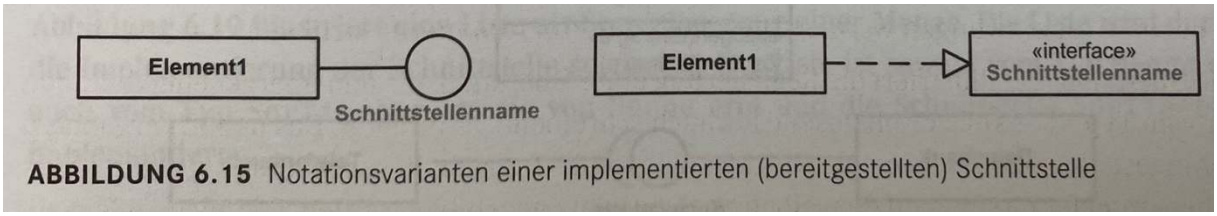


ABBILDUNG 6.15 Notationsvarianten einer implementierten (bereitgestellten) Schnittstelle

Element2 deutet durch das (Socket genannte) Halbkreissymbol an, dass dieser Classifier zur Erfüllung seiner Aufgabe die unterhalb des Sockets angegebene Schnittstelle benötigt. Rechts daneben findet sich die semantisch äquivalente Darstellungsvariante als Abhängigkeitsbeziehung.

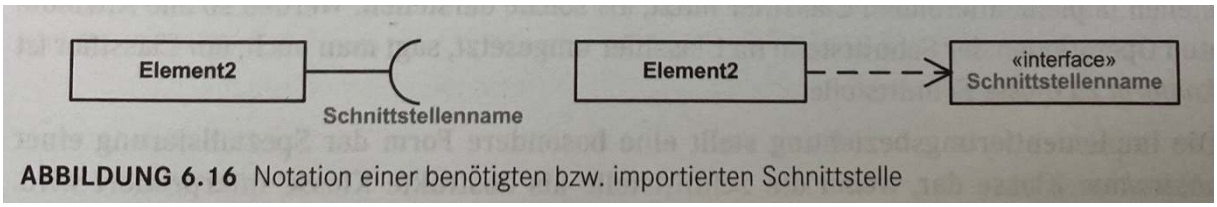


ABBILDUNG 6.16 Notation einer benötigten bzw. importierten Schnittstelle

Durch Kombination des Socket- und des Ball-Symbols lässt sich das technische Ineinandergreifen des Anbieters (Element1) und des zugehörigen Benutzers (Element2) einer Schnittstelle veranschaulichen.

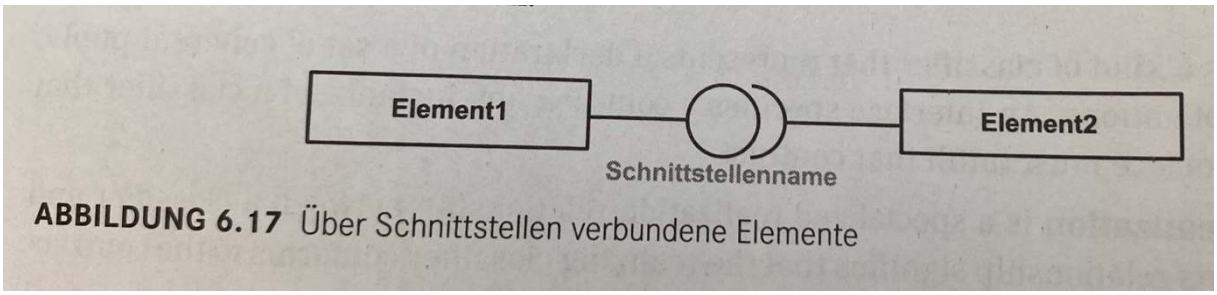


ABBILDUNG 6.17 Über Schnittstellen verbundene Elemente

→ Anwendung

Abbildung 6.19 illustriert eine Liste als Spezialisierung einer Menge. Die *Liste* wird durch die Implementierung der Schnittstelle sortierbar. Die Liste ist sowohl vom Typ Menge als auch vom Typ *Sortierbar*, da sie von *Menge* erbt und die Schnittstelle Sortierbar implementiert.

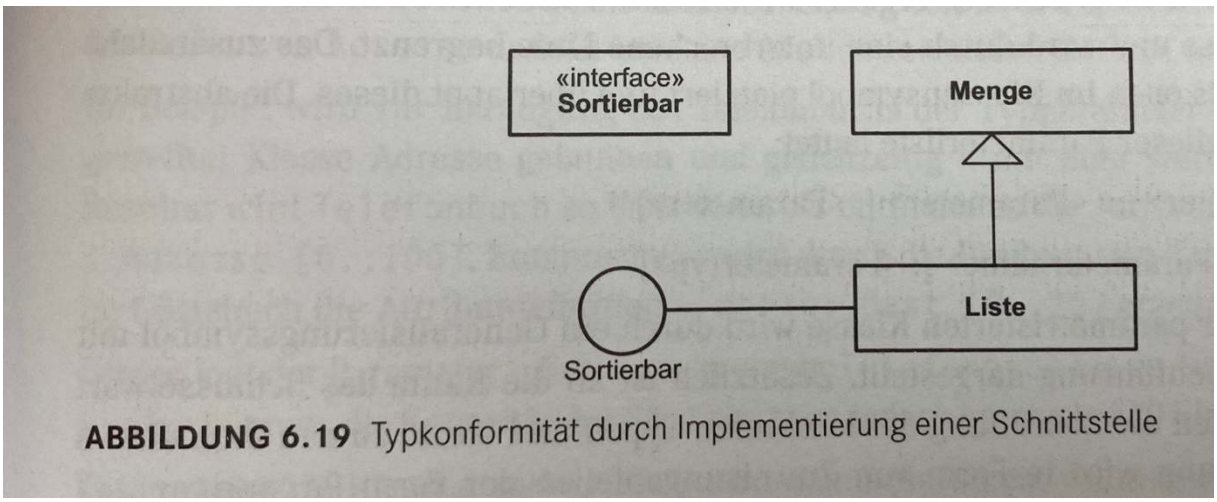
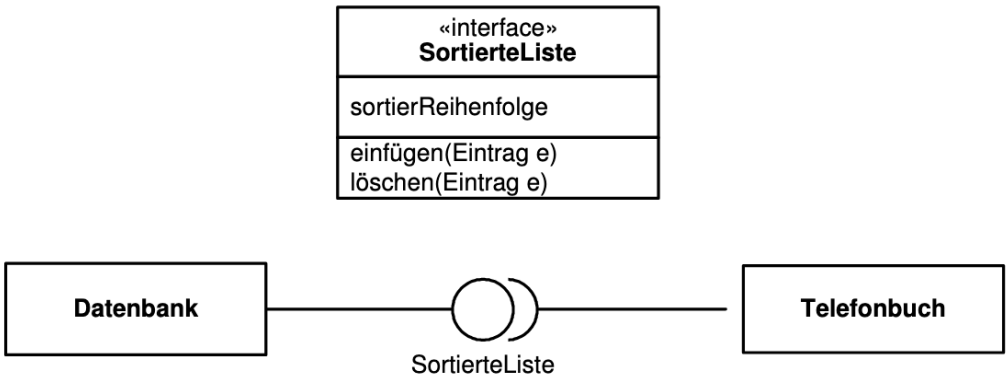


ABBILDUNG 6.19 Typkonformität durch Implementierung einer Schnittstelle

Umsetzung in Java:

```
public class Liste extends Menge implements Sortierbar {
        // Klassenrumpf von Liste
}
```



Umsetzung in Java:

```
interface SortierListe{
        int sortierReihenfolge = 0;
        public void einfühen(Eintrag e);
        public void löschen(Eintrag e);
}

class Datenbank implements SortierListe {
        public void einfügen(Eintrag e) {
                // Implementierung
        }
        public void löschen(Eintrag e) {
                // Implementierung
        }
}
```

1.4 Generalisierung

Die Generalisierungsbeziehung zwischen Classifiern wird mit einem Pfeil mit nicht ausgefüllter dreieckiger Pfeilspitze dargestellt. Er zeigt vom spezialisierten Classifier (Pfeilende) hin zum allgemeinen oder generalisierten Classifier (Pfeilspitze).

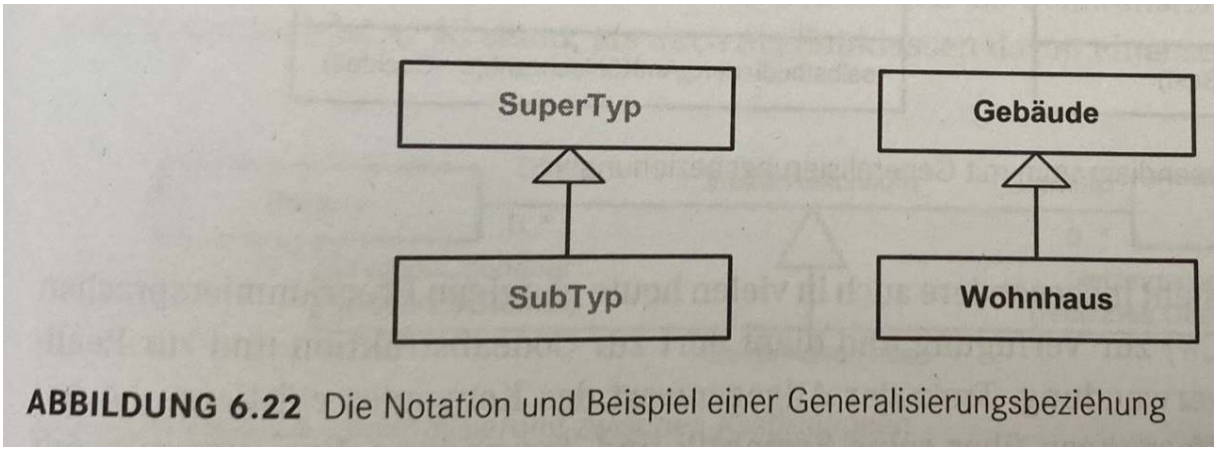


ABBILDUNG 6.22 Die Notation und Beispiel einer Generalisierungsbeziehung

Die Generalisierung stellt ein zentrales Konzept die objektorientierte Modellierung dar. Sie setzt zwei Klassen bzw. allgemeiner zwei Classifier so in Beziehung, dass der eine Classifier eine Verallgemeinerung des anderen darstellt. Die allgemeine Klasse wird häufig als Basis- oder Oberklasse einer spezialisierten (Unter-) Klasse bezeichnet.

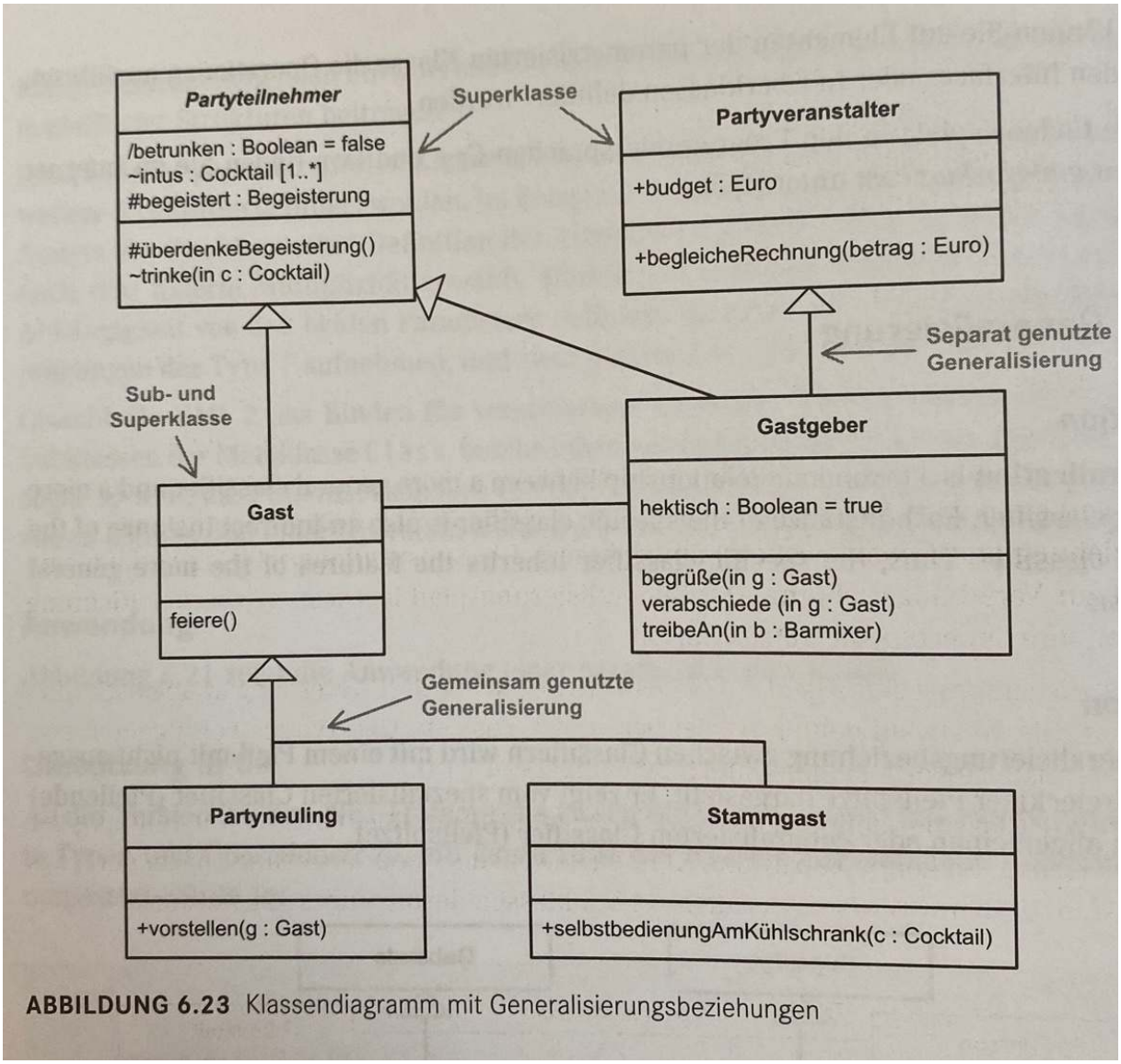


ABBILDUNG 6.23 Klassendiagramm mit Generalisierungsbeziehungen

ACHTUNG!!! Mehrfachvererbung in Java nicht möglich!!

1.4 Assoziation

Im Klassendiagramm dient die Assoziation (naturgemäß) zur Darstellung von Beziehungen zwischen Klassen. Die binäre Assoziation wird durch eine durchgezogene Linie dargestellt, die die teilnehmenden Klassen verbindet. Diese kann sowohl durch textuelle Angaben als auch durch grafische Symbole weiter konkretisiert werden. Die wichtigsten Angaben einer Assoziation sind ihr Name und die Beschriftung der Enden mit deren Namen (meist in Form von Rollen) und der Multipilizität:

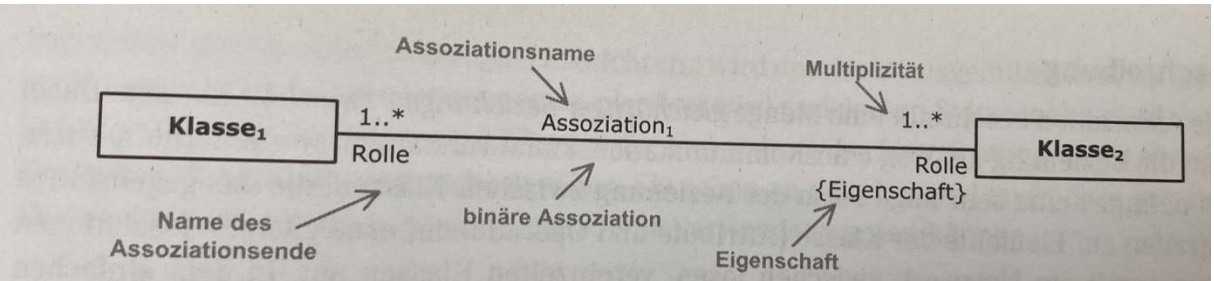


ABBILDUNG 6.32 Standardangaben an einer binären Assoziation

Die Definition der Navigationsrichtung einer Assoziation erfolgt durch einen Pfeil am Assoziationsende in der navigierbaren Richtung. Ein Kreuz weist explizit darauf hin, dass über dieses Ende nicht navigiert werden kann. Fehlt die explizite Angabe (Pfeil, Kreuz), so ist die Navigierbarkeit unspezifiziert. Sie können in einem solchen Fall global für das Modell festlegen, ob es navigierbar oder/und nicht navigierbar ist:

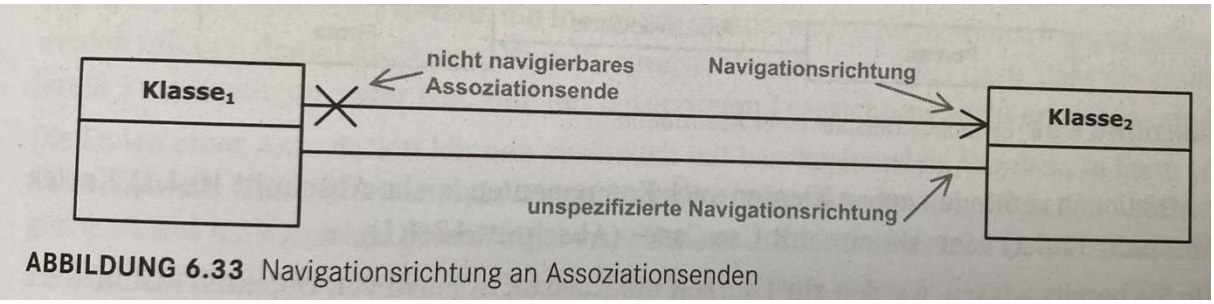


ABBILDUNG 6.33 Navigationsrichtung an Assoziationsenden

Zwischen Assoziationen können Einschränkungen formuliert werden, die als gestrichelte Linien dargestellt werden. An sie wird die einschränkende Bedingung in geschweiften Klammern notiert. Des Weiteren werden Ganzes-Teile-Beziehung durch Rauten am Ende des Ganzen gekennzeichnet:

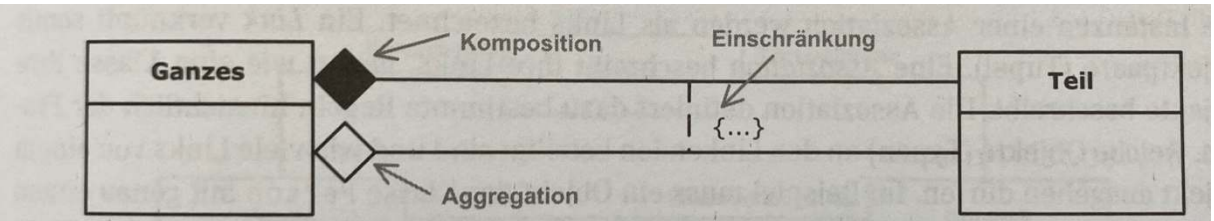


ABBILDUNG 6.34 Teile-/Ganzes-Beziehungen

Die Teile von n-ären Assoziationen, die für n > 2 mehr als zwei Klassen verbinden, sind durch eine große Raute in der Mitte verknüpft. Diese Notation ist auch für binäre Assoziationen zulässig, aber sehr unüblich:

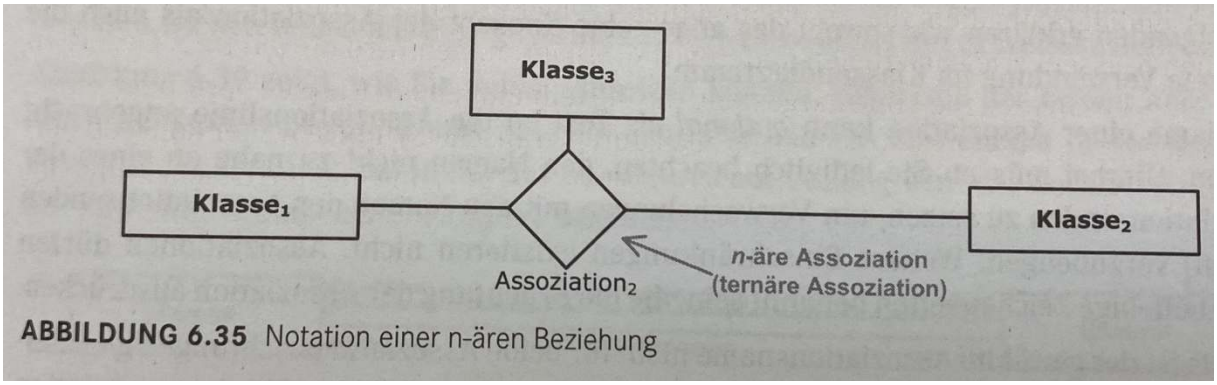


ABBILDUNG 6.35 Notation einer n-ären Beziehung

→ Beschreibung

Eine Assoziation beschreibt eine Menge gleichartiger Beziehungen zwischen Klassen. Dabei kann die Beziehung als Weg oder Kommunikationskanal verstanden werden. Die Assoziation definiert eine sehr enge Form der Beziehung zwischen Klassen, die das gegenseitige Zugreifen auf Elemente der Klasse (Attribute und Operationen) ermöglicht. Assoziationen bauen somit ein Netzwerk zwischen losen, vereinzelten Klassen auf.

In dem einfachen Beispiel verbindet die Assoziation Arbeitsverhältnis die beiden Klassen Person und Firma:

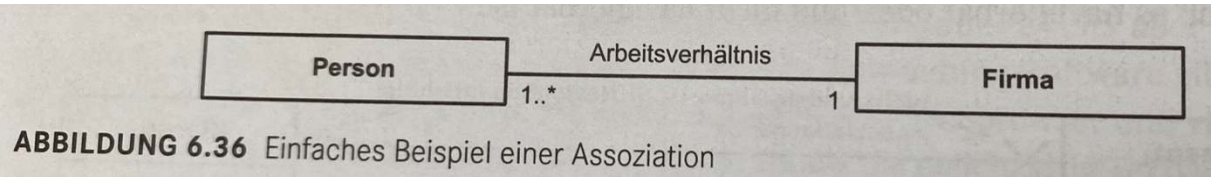


ABBILDUNG 6.36 Einfaches Beispiel einer Assoziation

Die Angabe 1 am Assoziationsende (Firma) definiert, dass eine Person mit genau einer Firma im Arbeitsverhältnis steht. Umgekehrt stehen mit der Firma mindestens eine, aber auch beliebige viele Person(-en) in einem Arbeitsverhältnis. Dies wird durch die Angabe 1..* gefordert.

→ Leserichtung

Oftmals ist der gewählte Assoziationsname nicht für beide Assoziationsrichtungen gleichermaßen treffend. Ist dies der Fall, so kann eine bevorzugte Leserichtung für eine Assoziation angegeben werden. Die bevorzugte Leserichtung wird durch ein ausgefülltes Dreieck dargestellt, welches nach dem Assoziationsnamen platziert wird und dessen Spitze in die gewünschte Leserichtung weist:

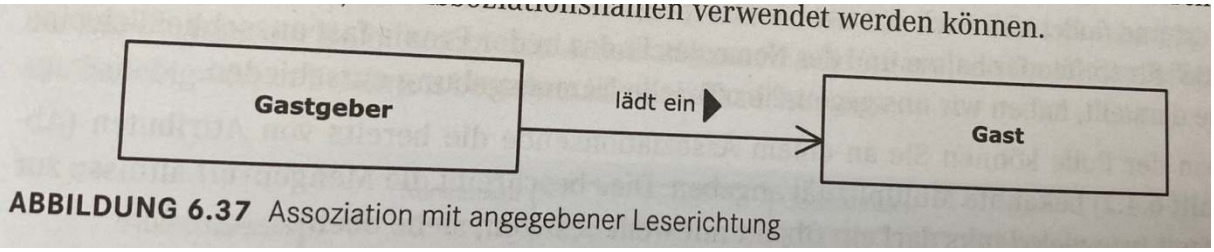


ABBILDUNG 6.37 Assoziation mit angegebener Leserichtung

Die Enden einer Assoziation können zusätzlich mit beschreibenden Angaben, in Form von Rollen (Name), Multipilizitäten und Navigierbarkeiten, versehen werden, wie in den Abbildungen 6.38 und 6.39 gezeigt. Diese Rollen erläutern die Semantik einer an der Assoziation teilnehmenden Klasse und die Form der Teilnahme näher.

Mit Hilfe von Rollen kann die spezifische Verwendung einer Klasse näher beschrieben werden. Daher sollten Sie dieses Element der Assoziationsmodellierung intensiv einsetzen. Insbesondere dann, wenn Klassen durch mehrere verschiedenartigen Assoziationen verbunden sind, stellen Rollen das einzige Mittel zur Unterscheidung der Einzelassoziationen dar:

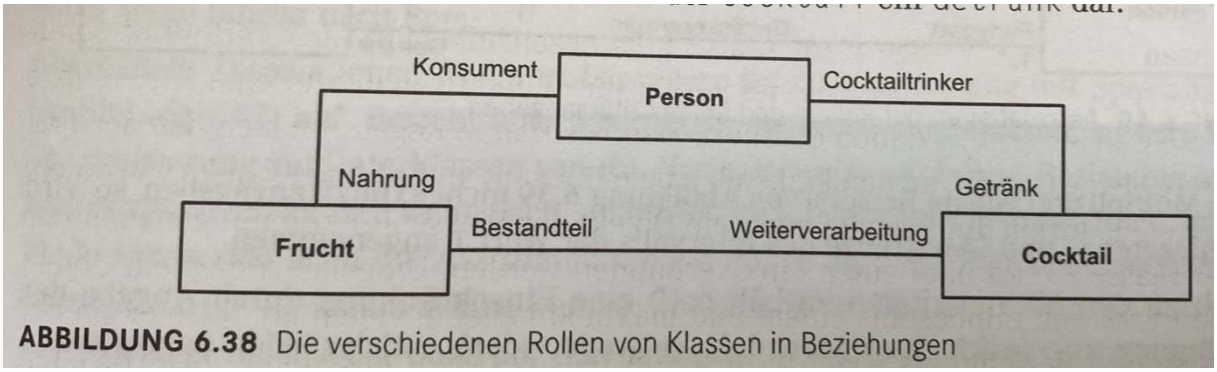


ABBILDUNG 6.38 Die verschiedenen Rollen von Klassen in Beziehungen

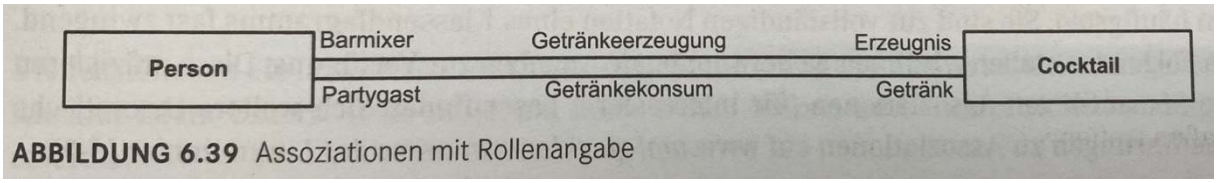


ABBILDUNG 6.39 Assoziationen mit Rollenangabe

→ Multiplizitäten

Die Multipilizitäten beschreiben die Mengenverhältnisse zur Laufzeit.
Die zu einer Rolle gehörenden Multipilizitätsangabe wird am „gegenüberliegenden“ Assoziationsende angetragen.

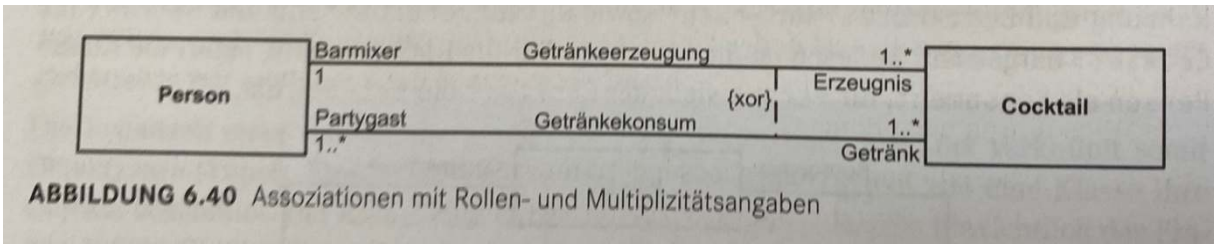


ABBILDUNG 6.40 Assoziationen mit Rollen- und Multiplizitätsangaben

- Jeder Partygast konsumiert mindestens einen Cocktail.
- Jeder Cocktail wird von mindestens einem Partygast konsumiert.
- Jeder Barmixer mixt mindestens einen Cocktail.
- Jeder Cocktail wird durch genau einen Barmixer gemixt.

→ Weiterführende Angaben

Die bisherigen Angaben (Assoziationsname, Roll, Multiplizität) braucht man in der Praxis am häufigsten. Sie sind zur vollständigen Notation eines Klassendiagramms fast zwingend.

1.4.1 Navigierbare Assoziation

Die Navigierbarkeit von Assoziationen wird durch eine Pfeilspitze am Ende einer Assoziation ausgedrückt. Die Pfeilrichtung zeigt an, dass die Instanzen der Klasse A die Instanzen der Klasse B in Pfeilrichtung „kennen“.

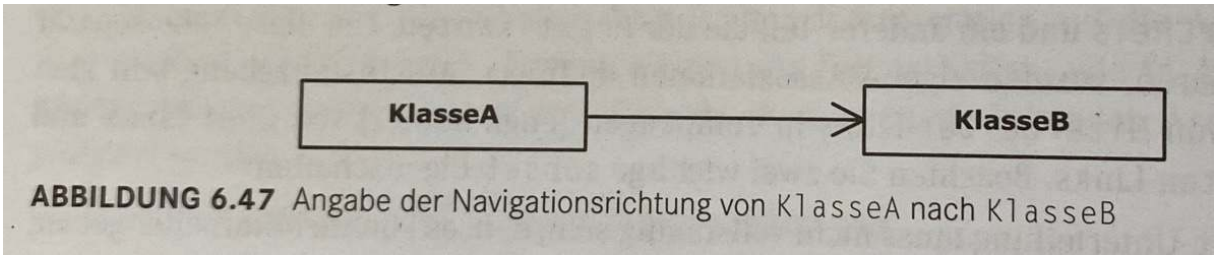


ABBILDUNG 6.47 Angabe der Navigationsrichtung von `KlasseA` nach `KlasseB`

Entsprechend der Navigierbarkeit wird zwischen unidirektionalen und bidirektionalen Assoziationen unterschieden. Bidirektionale Assoziationen können in beide Richtungen navigiert werden, unidirektionale hingegen nur in eine:

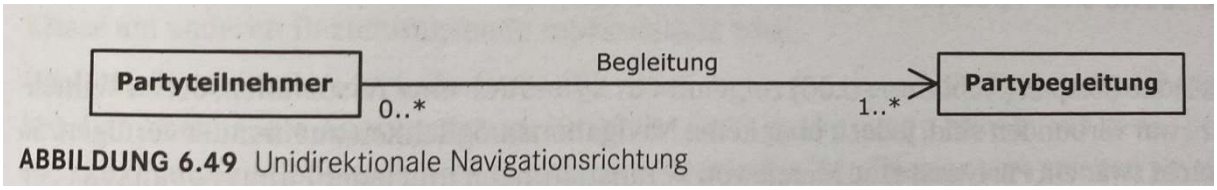


ABBILDUNG 6.49 Unidirektionale Navigationsrichtung

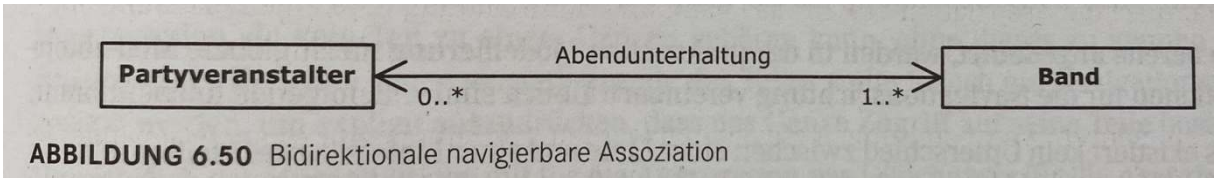


ABBILDUNG 6.50 Bidirektionale navigierbare Assoziation

→ Verbot der Navigierbarkeit durch Kreuzsymbol

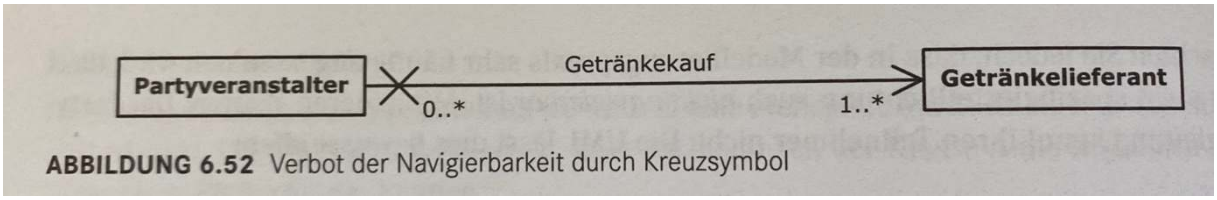


ABBILDUNG 6.52 Verbot der Navigierbarkeit durch Kreuzsymbol

Ein Beispiel hierfür zeigt die Assoziation Getränkekauf. Zwar besitzt der Partyveranstalter Kenntnis seiner Getränkelieferanten (es können auch mehrere sein, da das entsprechende Assoziationsende mit der Multiplizität 1..* versehen ist), da ihn eine navigierbare Assoziation mit diesen verbindet. Jedoch möchte der Lieferant keine Speicherung seiner möglichen Kundenbeziehungen (z.B. Supermarkt) vornehmen. Aus diesem Grund ist das zu Partyveranstalter führende Assozationsende auf nicht navigierbar gesetzt, was durch das Kreuzsymbol ausgedrückt wird.

Zu merken:

- Es existiert kein Unterschied zwischen einer Linie und einer Linie mit zwei Pfeilen, d.h. die komplett unspezifierte und die bidirektionale Assoziation ist auch nur in Pfeilrichtung navigierbar und in Gegenrichtung nicht navigierbar.
- Es existiert kein Unterschied zwischen einer Linie mit einem Pfeil und einer Linie mit einem Kreuz und Pfeil, d.h. die unidirektionale Assoziation ist auch nur in Pfeilrichtung navigierbar und in Gegenrichtung nicht navigierbar.

→ Diamantsymbol

Die „kleinen Diamanten“ am Ende einer Assoziation drücken verschiedene Typen des Zusammenhanges zwischen assoziierten Objekten aus: *Aggregation* und *Kompositionsaggregation* (oder kurz *Komposition*). Beide sind nur für binäre Assoziation zulässig.

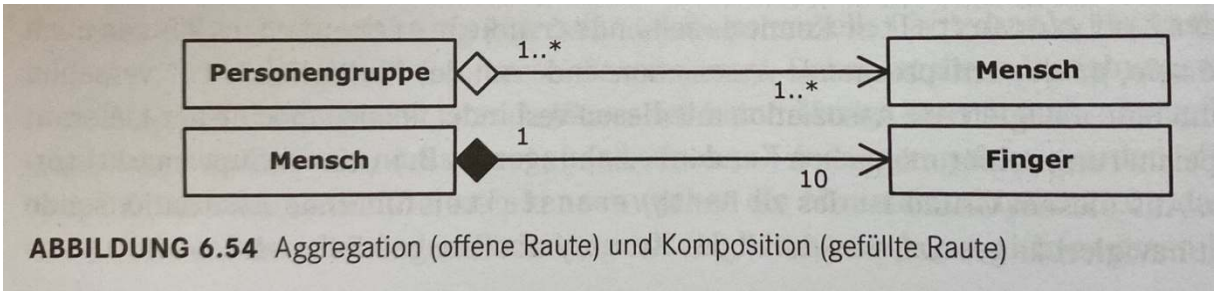


ABBILDUNG 6.54 Aggregation (offene Raute) und Komposition (gefüllte Raute)

→ Aggregation

Eine Aggregation drückt eine „Teile-Ganzes-Beziehung“ aus. Die aggregierten Instanzen einer Klasse sind dabei Teil eines Ganzen, welches durch die Klasse am anderen Beziehungsende repräsentiert wird.

Die UML-Spezifikation legt die Semantik einer Aggregation auf die Leseart „besteht aus“ fest. Demnach ist die Aggregation zu lesen als „Personengruppe *besteht aus* Menschen“. Dasjenige Assoziationsende, welches mit dem leeren kleinen Diamanten versehen ist, wird als „Ganzes“ aufgefasst, seine Teile befinden sich als Klasse am anderen Assoziationsende.

Am Assoziationsende des Teiles sollte jedoch ein Navigationspfeil gesetzt werden, um explizit auszudrücken, dass das Ganze Zugriff auf seine Teile besitzt.

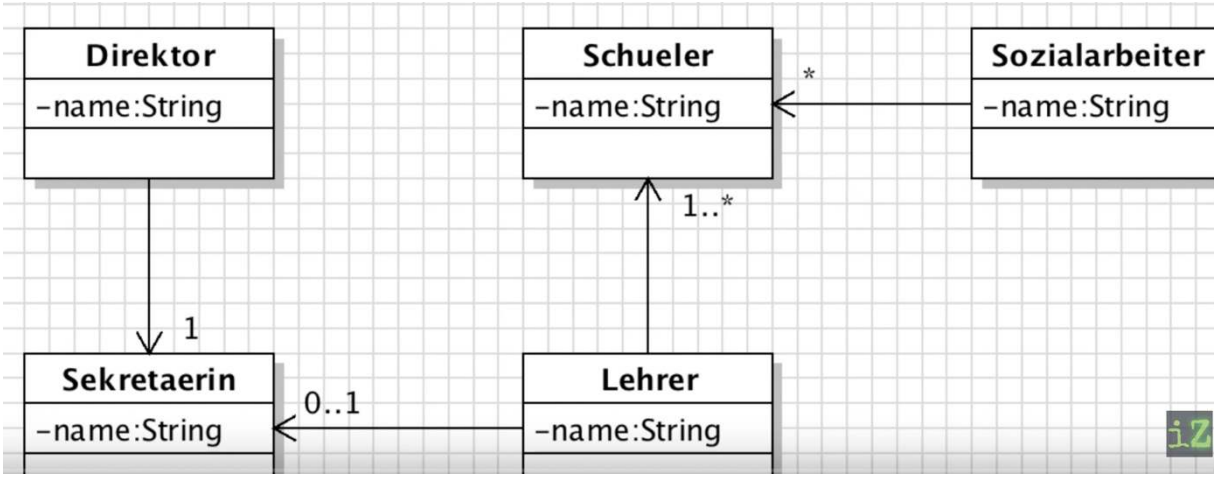
→ Komposition (-saggegation)

Eine strengere Form des Zusammenhanges wird durch die Komposition definiert. Sie drückt im Gegensatz zur Aggregation die physische Inklusion der Teile im Ganten aus. Teile und Ganzes bilden eine Einheit, deren Auflösung durchaus die Zerstörung des Ganzen zur Folge haben kann. Im Beispiel besitzt der Mensch 10 Finger, die zusammen eine Einheit bilden.

Die Lebensdauer des Teils hängt von der des Ganzen ab. Wird das Ganze zerstört, so „sterben“ die konstituierenden Teile mit dem Ganten, es sei denn, das Teil wurde vor der Zerstörung vom Ganten entfernt.

Im Unterschied zur Aggregation ist für die Kompositionen die Auswahl der verwendbaren Multipilizitäten beschränkt. So darf zwar jedes Ganze über beliebige Teile verfügen, doch darf ein Teil lediglich zu genau einem Ganzen beitragen. (Die 10 Finger können ja nicht gleichzeitig 2 Menschen gehören) Aus diesem Grund ist die am ausgefüllten kleinen Diamanten angetragene Multiplizität auf genau 0, 0..1 oder 1 fixiert und wird daher oft auch weggelassen, was automatisch zur Multiplizität 1 führt.3

2. Anwendungsbeispiel




```
public class Direktor {
        private String name;
        private Sekretaerin sekretaerin;

        public Direktor (String name, Sekretaerin sekretaerin) {
                this.name = name;
                this.sekretaerin = sekretaerin;
        }
}
public class Sekretaerin {
        private String name;

        public Sekretaerin (String name) {
                this.name = name;
        }
}
public class Lehrer {
        private String name;
        private Sekretaerin sekretaerin;
        private ArrayList<Schueler> schuelerListe; //Schülerliste
        public Lehrer (String name, ArrayList<Schueler> schuelerListe) {
                if(schuelerListe.isEmpty(){
                System.out.println("Schuelerliste muss mindestens ein Element enthalten");
                }
                this.name = name;
                this.schuelerListe = schuelerListe;
        }
        // Wir brauchen noch Getter und Setter für das Attribut sekretaerin
        public void sekretaerin(Sekretaering sekretaerin){
                this.sekretaerin = sekretaerin;
        }
```

```
        public Sekretaerin getSekretaerin(){
                return this.sekretaerin;
        }
}



public class Schueler {
        private String name;

        public Schueler (String name) {
                this.name = name;
        }
}
public class Sozialarbeiter {
        private String name;
        private ArrayList<Schueler> schuelerListe = new ArrayList<Schueler>();

        public Sozialarbeiter (String name) {
                this.name = name;
        }

}
```

→ Bezeichnungen Multiplizitäten

- 0..1 : Lehrer Klasse bekommt nur ein Attribut vom Typ Sekretaerin und wir instanziieren es nicht, da es 0 oder 1 sein kann.
- 1 : Der Direktor muss auf jeden Fall eine Sekretärin zugewiesen bekommen, dies geschieht am besten im Konstruktor und es darf keine null Instanz sein!
- 0..* : Bei beliebig viele eignen sich ArrayLists am besten. Da es 0 sein kann ist die ArrayListe nicht im Konstruktor.
- 1..* : Wie definieren ArrayList-Attribut und erzeugen noch kein Listeobjekt. Die Liste muss mindestens 1 Element enthalten. Dies machen wir erneut über dem Konstruktor. Die übergebene Liste wird seiner schuelerListe zugewiesen. → Man kann dabei abfangen, ob die schuelerListe, die übergeben wird Leer ist.