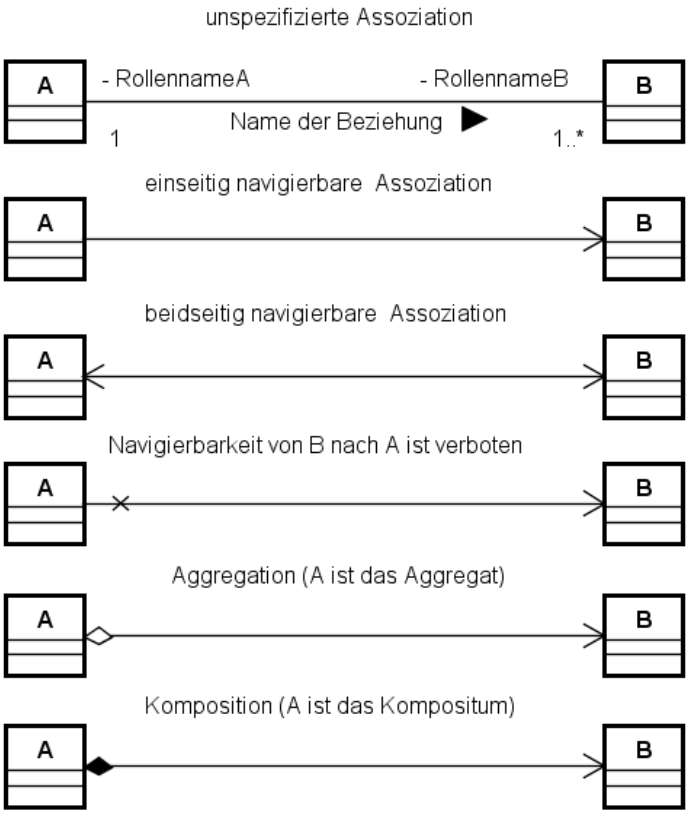
Assoziationsarten[1]

Assoziationen stellen in Klassendiagrammen die Beziehungen zwischen Klassen dar.

Assoziationen werden durch Linien zwischen den beteiligten Klassen hergestellt. Eine Assoziation kann einen Namen haben, der die Art der Assoziation beschreibt. Das schwarze Dreieck gibt hierbei die Leserichtung an.

An den Enden einer Assoziation können Rollennamen (die „Rolle“ der Klasse in dieser Beziehung) und Multiplizitäten angegeben werden. Die Rollennamen dienen zumeist als Attributbezeichnung bei der Implementation.

Die Multiplizität gibt an, mit wie vielen Objekten der gegenüber liegenden Klasse ein Objekt in Beziehung stehen kann. Im obigen Beispiel steht **genau ein Objekt der Klasse A** mit **mindestens einem Objekt der Klasse B** in einer Beziehung; umgekehrt steht **genau ein Objekt der Klasse B** mit **genau einem Objekt der Klasse A** in einer Beziehung.



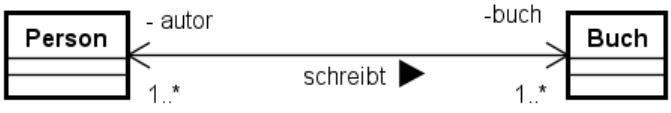
Weitere Beispiele[2] zu Multiplizitäten:

Multiplizität	*Bedeutung*
1	genau einer
0..1	keiner oder einer
1..5	einer bis fünf
*	keiner, einer oder mehrere
0..*	keiner, einer oder mehrere
1..*	mindestens einer

In der Grafik oben ist die Symbolik für verschiedene Arten einer Assoziation dargestellt. Eine unspezifizierte Assoziation ist eine, über deren Navigierbarkeit noch nicht entschieden wurde. In der Regel wird eine solche Beziehung mit beidseitiger Navigierbarkeit implementiert.

Im Folgenden sollen zunächst Beziehungen, die **nicht** Aggregation oder gar Komposition sind, betrachtet werden!

Die oberen vier dargestellten Assoziationen lassen sich als „kennt“- oder „benutzt“-Beziehungen beschreiben. Die Beziehung zwischen Objekten der beteiligten Klassen ist ein lose Beziehung. **Grundlegend hierbei ist, dass die Existenz der Objekte voneinander unabhängig ist!**



Das Buch existiert auch noch, selbst wenn der oder die Autoren des konkreten Buchs bereits tot sind.

Zwischen zwei bestimmten Klassen können auch mehrere voneinander unabhängige Beziehungen existieren. So könnte zwischen den beiden Klassen Person und Buch auch eine Beziehung vorhanden sein, in der ein Objekt der Klasse **`Person`** in der Rolle des Lesers Bücher liest.

1 Eine Einführung mit verständlichen Beispielen: **`http://www.kstbb.de/informatik/oo/04/index.html`** abgerufen am 17.12.2017

2 Entnommen bei **`http://www.kstbb.de/informatik/oo/04/4_2_Multiplizitaet.html`** abgerufen am 17.12.2017

Hinsichtlich der Navigierbarkeit ist letztendlich das Fachkonzept entscheidend. Während es sinnvoll erscheint, dass ein Buch seinen Autor bzw. seine Autoren kennt und umgekehrt ein Autor die von ihm geschriebenen Bücher kennt, ist es zwar möglicherweise relevant, dass ein Leser die von ihm gelesenen Bücher kennt, aber das konkrete Buch nicht wirklich seine Leser kennen muss.

Weiterhin werden solche lose Beziehungen zwischen Objekten einer Klasse auch als „kann“- oder „muss“-Beziehungen beschrieben. Der Unterschied zwischen diesen beiden Begriffen resultiert aus der Multiplizität. Bei einer „kann“-Beziehung hat die minimale Multiplizität den Wert „0“; bei einer „muss“-Beziehung ist dieser Wert mindestens „1“.

Ein Buch „muss“ mindestens eine Person als Autor haben oder eine Person „muss“ mindestens ein Buch geschrieben haben (sonst wäre sie kein Autor!).

Ein Buch „kann“ von einer Person gelesen werden, „muss“ aber nicht!

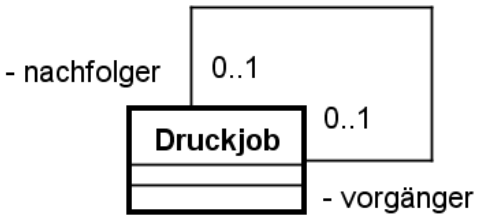
Weiterhin lassen sich Assoziationen nach der Anzahl der an der Assoziation beteiligten Klassen kategorisieren. Man unterscheidet hierbei zwischen:

- unären Assoziationen
- binäre Assoziationen
- mehrwertige Assoziationen (z.B. ternäre Assoziationen)

Unäre Assoziationen

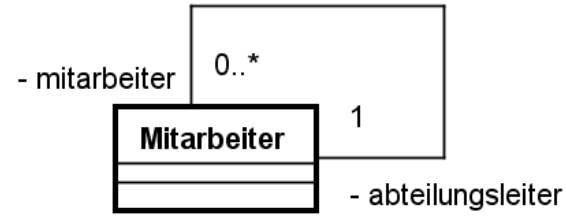
Bei einer unären Beziehung hat ein bestimmtes Objekt einer Klasse Verbindungen zu Objekten der gleichen Klasse. Man nennt solche Beziehungen auch **reflexiv**. Mit unären Assoziationen werden typischerweise Hierarchien bzw. Folgen ausgedrückt.

Im ersten Beispiel gibt es eine Klasse **`Druckjob`**. Druckjobs werden durch einen Druckerspooler abgewickelt, der Druckjob für Druckjob an den Drucker sendet. Jeder Druckjob hat einen Nachfolger (außer, es ist der letzte Druckjob in der Schlange) und einen Vorgänger (außer, es ist der erste Druckjob in der Schlange).



In der Klasse **`Druckjob`** muss es demnach zwei Attribute geben, nämlich **`nachfolger`** und **`vorgänger`**, mit denen auf das entsprechende **`Druckjob`**-Objekt verwiesen wird.

Im zweiten Beispiel ist eine Klasse **`Mitarbeiter`** dargestellt. Jedes **`Mitarbeiter`**-Objekt hat genau einen Abteilungsleiter, der aber im Grunde auch ein Mitarbeiter ist. Jeder Abteilungsleiter verfügt über keine oder viele **`Mitarbeiter`**-Objekte.

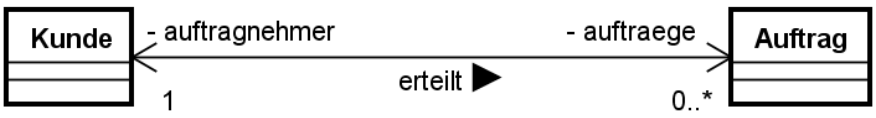


Somit muss es in der Klasse **`Mitarbeiter`** ein Attribut **`abteilungsleiter`** geben, welches auf den Mitarbeiter verweist, der die Rolle eines Abteilungsleiters einnimmt. Weiterhin muss es ein Attribut **`mitarbeiter`** geben, dass vom Typ eines Containers ist, der Verweise auf die Mitarbeiter des betreffenden Abteilungsleiters enthält.

Binäre Assoziation

Alle eingangs vorgestellten Assoziationen sind binäre Assoziationen, d.h. ein Objekt einer Klasse hat eine Verbindung zu einer Anzahl von Objekten einer anderen Klasse, je nach Multiplizität.

Im dargestellten Beispiel erteilt ein Kunde keine oder viele Aufträge, wobei die Kunden unabhängig von den Aufträgen existieren und umgekehrt.



Ein erstellter Auftrag kann einem Kunden zugeordnet werden. Aufgrund der beidseitigen Navigierbarkeit wird in der Klasse **`Kunde`** ein Attribut **`auftraege`** mit dem Datentyp eines Containers für Objekte des Typs **`Auftrag`** vorgesehen. In der Klasse **`Auftrag`** existiert ein Attribut **`auftragnehmer`**, welches den betreffenden Kunden referenziert.

Für die Verwaltung einer Assoziation existieren typischerweise drei Methoden:

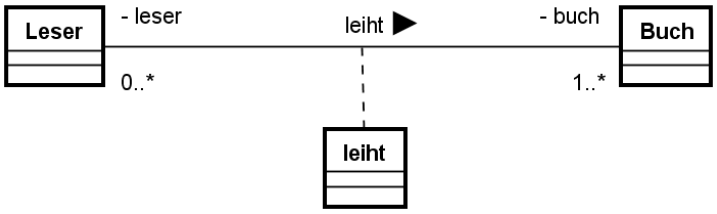
- Herstellen der Objektbeziehung; (z.B. einem Auftrag einen Auftragnehmer zuordnen)
- Lösen der Objektbeziehung; (z.B. bei einem Auftrag den Auftragnehmer löschen)
- Liefern des Objektes; (z.B. bei einem Auftrag den Auftragnehmer liefern)

Dies kann über speziell vorgesehene Methoden erfolgen oder -zumindest teilweise- über Properties bei C# abgewickelt werden.

Binäre Assoziationen mit Attributen und Methoden

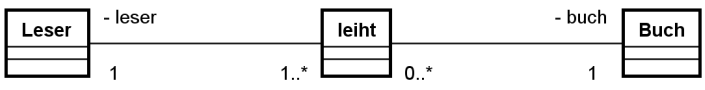
In manchen Situationen ist eine Assoziation selbst mit Attributen und Methoden behaftet.

Ein Leser leiht in einer Bibliothek Bücher aus. Zu einem Leih-Vorgang ist festzuhalten, wer, wann, welches Buch für wie lange ausleiht. Diese Informationen sind sinnvollerweise an einen Leih-Vorgang zu knüpfen.



Deshalb wird hier eine Assoziationsklasse erstellt, die nun die Informationen zu dem Leih-Vorgang aufnimmt.

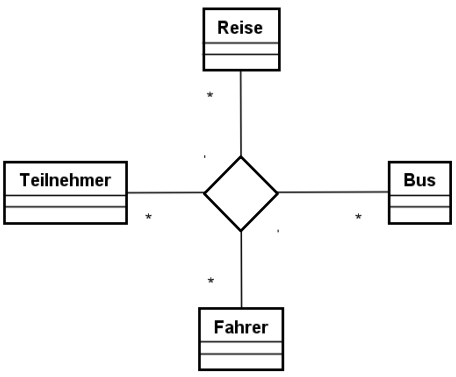
Hierdurch ändern sich die Multiplizitäten. An den ursprünglichen Assoziationsenden gilt nun die Multiplizität „1“. An den Assoziationsenden an der Assoziationsklasse gelten nun die Multiplizitäten der ursprünglichen Assoziation.



Mehrwertige Assoziationen

Hierunter fallen alle Assoziationen, bei denen ein Objekt von einer der assoziierten Klassen eine Verbindung zu mehreren Objekten der anderen Klassen haben kann.

Solche Assoziationen werden ebenfalls durch Assoziationsklassen aufgelöst.



Aufgaben

1. Laden Sie die Projektmappe „Beziehungen zwischen Klassen“ und wählen Sie das Projekt „Binäre Assoziation“ aus.

   Erstellen Sie das Projekt und führen Sie es „ohne Debugging“ aus.

   Analysieren Sie die Art der Umsetzung einer Binären Assoziation in dem Projekt.
   Sehen Sie sich an, wie bei einem Objekt die Beziehung zum referenzierten Objekt hergestellt, geändert und gelöst wird.

2. Entwerfen Sie die zwei Klassen **`Fahrzeug`** und **`Halter`** mit einigen sinnvollen Attributen. Ein Halter kann keine oder mehrere Fahrzeuge haben und kennt seine Fahrzeuge. Ein Fahrzeug gehört keinem oder einem Halter und kennt diesen nicht.

   Implementieren Sie diese binäre Assoziation und schreiben Sie ein Testprogramm, in dem mindestens zwei Halter mit jeweils mindestens zwei ihnen zugeordneten Fahrzeugen erzeugt werden.

3. Ändern Sie die Beziehung so, dass ein **`Fahrzeug`**-Objekt sein **`Halter`**-Objekt kennt.

4. Verwalten Sie sowohl die **`Halter`**-Objekte als auch die **`Fahrzeug`**-Objekte mittels einer Liste (z.B. **`ArrayList`** oder **`List<T>`**).

   Durchlaufen (iterieren) Sie die Liste mit den **`Halter`**-Objekten und lassen Sie sich die Daten des Halters sowie die ihm zugeordneten **`Fahrzeug`**-Objekte anzeigen.

   Durchlaufen Sie die Liste mit den **`Fahrzeug`**-Objekten und lassen Sie sich die Daten des Fahrzeugs sowie die zugeordneten **`Halter`**-Objekte anzeigen.

5. Entwerfen Sie zu dem Adressbuch-Projekt die Klassen **`Address`** und **`Phone`**. Jeder Datensatz soll ein Property bzw. Attribut **`Id:int`**, welches in einer Klassenvariablen **`lastid:int`** hochgezählt wird, enthalten.Jedem **`Person`**-Objekt sollen beliebig viele **`Adress`**- und **`Phone`**-Objekte zugeordnet werden können. Die **`Person`**-Objekte sollen die ihnen zugeordneten **`Adress`**- und **`Phone`**-Objekte kennen, umgekehrt nicht.

   Zeichnen Sie ein entsprechendes UML-Klassendiagramm.