5.6 Einführung und Klassen/Assoziation, Aggregation, Komposition

5.6.1 Mehrere Klassen in einem Programm

Im vorigen Thema haben wir erstmals mehrere Objekte derselben Klasse benutzt. Wir haben gesehen, dass man sie wie normale Variablen verwenden kann. Jetzt geht es einen Schritt weiter: Wir benutzen Objekte mehrerer Klassen in einem Programm. Natürlich *nicht* so, dass die Objekte nichts miteinander zu tun haben (ein Bruch, eine Person und ein Punkt). Sondern so, dass die Objekte zueinander in Beziehung stehen.

Dazu folgendes Beispiel: Es soll eine Urlaubsverwaltung programmiert werden. Dafür soll die Klasse `zeitraumtyp` erstellt werden. Ein Zeitraum kann aus vielen Tagen bestehen, aber man beschreibt ihn am besten, indem man das Datum des ersten und das Datum des letzten Tages angibt. Die Klasse `datum` gibt es ja schon. Daher wäre es schön, wenn man aus der Klasse `datumtyp` die Klasse `zeitraumtyp` aufbauen könnte, so ähnlich, wie man aus Modulen ein Gerät aufbauen kann.

5.6.2 Assoziation, Aggregation und Komposition

Damit wir richtig bauen, brauchen wir zuerst etwas Theorie. In der Informatik gibt es bei den Beziehungen zwischen Daten/Objekten drei wichtige Fälle, nämlich Assoziation, Aggregation und Komposition.

5.6.2.1 Assoziation Hier ist das Beispiel der Verein und die Person: Ein Verein umfasst im Normalfall mehrere Personen. Aber: Eine Person kann auch ohne Verein leben, und ein Verein hat am Anfang vielleicht noch keine Mitglieder. Andererseits kann eine Person auch in mehreren Vereinen Mitglied sein. Es handelt sich also um eine eher lockere Verbindung.

Abbildung 1 zeigt die UML-Darstellung dieser Verbindung. Eine einfache Linie symbolisiert die Assoziation. Die Begriffe mit den Pfeilen oberhalb und unterhalb der Linie kennzeichnen sprachlich die Art der Assoziation: Das Mitglied bzw. die Person *liebt* ihren Verein (das kleine schwarze Dreieck stellt einen Pfeil von rechts nach links dar). Umgekehrt *hat* der Verein das Mitglied (Pfeil von links nach rechts).

Die Zahl 1 bzw. das Sternchen neben den Kästen geben an, wie viele von diesen Assoziationen ein Objekt haben kann. In diesem UML-Diagramm wäre es so, dass ein Verein viele Mitglieder haben kann, aber jede Person in genau einem Verein Mitglied ist.

Wenn jede Person (wie oben beschrieben) in beliebig vielen Vereinen Mitglied sein kann, muss neben beiden Kästen ein Sternchen stehen.

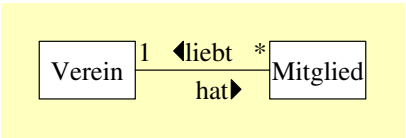

Abbildung 1: Assoziation

5.6.2.2 Aggregation , Hier ist das Beispiel ein Auto und seinen Räder. Räder existieren auch ohne Auto, aber ein Auto enthält vier Räder. Diese Räder gehören in diesem Moment nur diesem einen Auto. Diese Verbindung ist also schon etwas stärker.

Abbildung 2 zeigt die UML-Darstellung der Aggregation. Die einfache Linie ist um ein auf die Spitze gestelltes leeres Quadrat ergänzt[1]. Die Zahlen neben den Kästen zeigen hier: Zu jedem Auto gehören *vier* Räder. Jedes Rad gehört zu genau *einem* Auto.

[1] Als Eselsbrücke kann man sich merken: Auf der Seite, auf der vier Räder gebraucht werden, wird das Viereck gezeichnet.

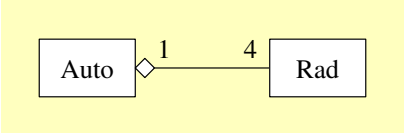

Abbildung 2: Aggregation

5.6.2.3 Komposition Hier ist das Beispiel ein Mensch und sein Gehirn. Ein Gehirn lebt nicht ohne seinen Menschen [2]. Diese Verbindung ist daher unlösbar.

Abbildung 3 zeigt die die UML-Darstellung der Komposition. Hier ist die Linie um ein ausgefülltes auf der Spitze stehendes Quadrat ergänzt.

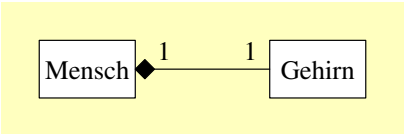

Abbildung 3: Komposition

5.6.3 Assoziation, Aggregation und Komposition in C und C++

Im Fall der Urlaubsverwaltung liegt offenbar eine **Aggregation** vor. Aber wie werden diese Fälle sinnvoll programmiert? Dazu gibt es folgende Richtschnur:

- Assoziation – Beziehung durch Zeiger oder Referenz. Der Verein bekommt Zeiger (oder Referenzen) auf Person-Objekte.

```
1 class verein
2 {
3    person *pmitglied1;
4    person *pmitglied2;
5    /* ... */
6 };
```

  Umgekehrt könnte auch das Person-Objekt einen Zeiger auf das Verein-Objekt haben. Die genaue Realisierung hängt von den Anforderungen ab.

- Aggregation – Bestandteile werden dargestellt, indem die Klasse Auto vier Objekt-Variablen vom Typ Rad bekommt.

```
1 class cauto
2 {
3    rad vornelinks;
4    rad vornerechts;
5    rad hintenlinks;
6    rad hintenrechts;
7    /* ... */
8 };
```

- Komposition – Eine Komposition kann man dadurch realisieren, dass eine Klasse (Mensch) eine Klasse (Gehirn) enthält. Dann kann man von der inneren Klasse nur dann ein Objekt

[2] hoffentlich bleibt das so

erstellen, wenn man ein Objekt der äußeren Klasse erstellt. Man *könnte* auch Vererbung (evtl. Mehrfachvererbung) dazu benutzen.

```
1 class mensch
2 {
3     class gehirn
4     {
5         /* ... */
6     };
7     /* ... */
8 };
```

Im Einzelfall kann es sein, dass man aus Gründen des Speicherplatzes oder des Zeitverhaltens eine andere Wahl trifft. C++ lässt das zu.

5.6.4 Die Klasse `zeitraumtyp`

Wie kann man also `zeitraumtyp` aufbauen?

```
1 class zeitraumtyp{
2 private:
3     datumtyp von;
4     datumtyp bis;
5 public:
6     // diverses gedoens
7 };
```

Wichtig ist in diesem Fall, dass die Klasse `datumtyp` einen Standardkonstruktor enthält, damit die beiden Elemente `von` und `bis` angelegt werden können. Der Konstruktor von `zeitraumtyp` kann danach ja noch weitere Aktionen ausführen[3].

[3] Allerdings sind `zeitraumtyp` und `datumtyp` verschiedene Klassen, so dass der Konstruktor von `zeitraumtyp` nicht direkt auf die Attribute von `von` und `bis` zugreifen kann. Näheres zu dieser Problematik im nächsten Thema.