Assoziationen in Java

Michael Dienert

16. Oktober 2018

1 Wiederholung: Gerneralisierung und Vererbung

Gerneralisierung ist das Gegenteil von Vererbung: Eine spezielle Klasse erbt von der allgemeineren Elternklasse und *erweitert* diese um Methoden und Eigenschaften.

Durch diese Erweiterungen wird die Kindsklasse eben spezieller als die allgemeinere Elternklasse.

Die Generalisierung wird mit einem Pfeil, dessen Spitze ein *nicht ausgefülltes, geschlossenes Dreieck* ist, dargestellt. Der Pfeil zeigt vom Speziellen zum Allgemeinen, eben die Generalisierung.

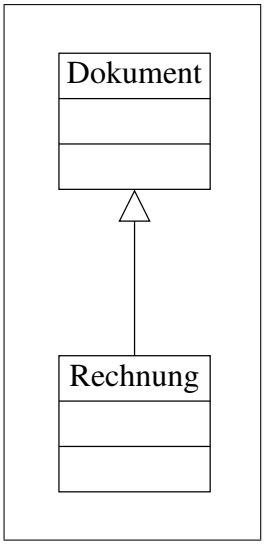

Abbildung 1: Generalisierung

Die Beziehung zwischen Dokument und Rechnung kann man so lesen: Eine Rechnung **ist** ein Dokument. D.h. Generalisierung/Vererbung führt zu *ist-Beziehungen*.

2 Assoziationen oder: Objekte lernen sich kennen

Eine wichtige Eigenschaft objektorientierter Programme ist, dass die Objekte untereinander Nachrichten austauschen, d.h. Methoden eines jeweils anderen Objekts aufrufen.

Dazu muss das Objekt das eine Methode aufrufen will, das Objekt, welches die Methode enthält, *kennen*, also eine *Referenz* auf dieses Objekt besitzen.

In Java wird die Referenz auf ein Objekt beim *Instanzieren* einer Klasse mit `new` erzeugt. Gespeichert wird die Referenz in einer Variablen vom Typ dieser Klasse:

```
Anschrift eineAnschrift = new Anschrift();
```

Die Referenz ist nichts anderes als die Speicheradresse, unter der das Objekt im Speicher abgelegt ist. **Die Referenz ist also ein Pointer**.

Dieses "*Kennen*" eines anderen Objekts nennt man **Assoziation**.

> Bei einer Assoziation **hat** ein Objekt eine Referenz auf ein anderes Objekt. D.h. Eine Assoziationen ist eine *hat-Beziehung*.

Hier werden nun zwei Fälle unterschieden:

Unidirektionale Assoziation - unidirektional heisst: in eine Richtung.

Abb. 2 zeigt ein Beispiel: ein `Rechnung` -Objekt hat eine Referenz auf ein `Anschrift` -Objekt, aber das `Anschrift` -Objekt hat keine Referenz auf das `Rechnung` -Objekt.

Dargestellt wird diese unidirektionale Assoziation mit einer Verbindungslinie zwischen den Klassendiagrammen mit einer **offenen** Pfeilspitze auf die Klasse, deren Objekte referenziert werden. Hier eine Grafik (mit etwas übertrieben gross dargestellter Pfeilspitze):

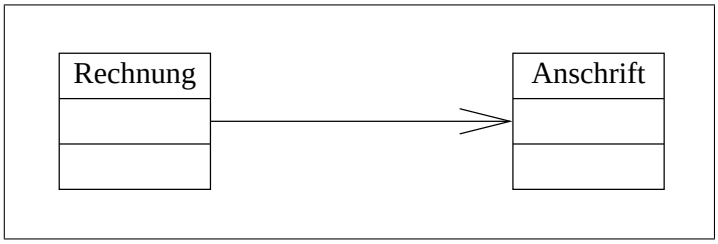

Abbildung 2: Unidirektionale Assoziation

Eine unidirektionale Assoziation lässt sich in Java sehr leicht programmieren:

```
class Rechnung{
  String rechnungsnummer;
  String kunde;

  Anschrift anschrift;
}

class Anschrift{
  // ...
}
```

Bididirektionale Assoziation - bei der bidirektionalen Assoziation kennen sich beide Objekte gegenseitig. Eine bidirektionale Assoziation sollte immer als zwei entgegengesetzte, unidirektionale Assoziationen dargestellt werden.

Nun ist es beim Entwurf eines UML-Diagramms noch nicht vorgeschrieben, mit einer Pfeilspitze festzulegen, wer wen kennt. Man kann eine Assoziation auch ohne Pfeilspitzen zeichnen und legt damit noch nicht fest, welches Objekt wessen Referenzen enthält.

3 Multipizität, oder: Wieviele Objekte kenne ich

Nun kann es sein, dass ein Objekt gleich mehrere Referenzen auf andere Objekte besitzt. Im UML-Diagramm kann man eine Angabe zu den maximal möglichen (kann-Beziehung) und minimal nötigen (muss-Beziehung) angeben: Abb. 3.

Abbildung 3: Kardinalitäten bei einer Assoziation

Gelesen wird diese Assoziation folgendermassen:

1 Firma beschäftigt * Mitarbeiter

Diese Angabe wird *Multipizität* (multiplicity) oder auch *Kardinalität* genannt.
Hier ein paar Beispiele:

1 genau eins (muss)

0,1 null oder eins (kann)

0..4 null bis 4

10,20 genau 10 oder genau 20

***** viele, einschliesslich 0

1..* viele, ohne 0

4 Rollen und Leserichtung

Das Beispiel aus dem vorherigen Kapitel lässt sich noch erweitern:
In diesem Beispiel wird die Assoziation noch um einen Namen (`beschaeftigt`) erweitert. Das kleine, schwarze Dreieck hinter dem Namen gibt nur die *Leserichtung* an!

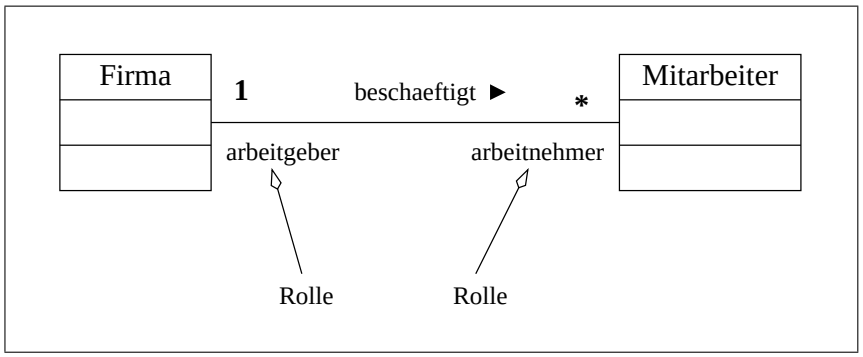

Abbildung 4: Vollständiges UML-Diagramm einer Assoziation

Zusätzlich zum Namen wurden noch zwei *Rollenbezeichnungen* eingeführt: aus Sicht der Firma hat ein Mitarbeiter *die Rolle des Arbeitnehmers* und umgekehrt ist spielt für den Mitarbeiter die Firma *die Rolle des Arbeitgebers*.

Man sollte gezielt nach sinnvollen Rollennamen suchen, da diese bei der Ausprogrammierung als Namen für die Objektrefrenzen dienen können.
Wichtig: die mit der Assoziation referenzierten Objekte, das also eine Klasse Objekte einer anderen Klasse besitzt (hat-Beziehung), werden **nicht zusätzlich** als Eigenschaften im Klassendiagramm aufgeführt.

Hier das ganze in Java:

```
public class Firma {
  //ein paar eigenschaften
  ArrayList arbeitnehmer = new ArrayList();

  //assoziation zum arbeitnehmer herstellen
  public void einstellen(Mitarbeiter knecht) {
    arbeitnehmer.add(knecht);
    //da diese assoz bidirektional ist muss auch gelten:
    knecht.anheuern(this);
  }

  //assoziation abbauen, knecht rausschmeissen
  public void rausschmeissen(Mitarbeiter knecht) {
    arbeitnehmer.remove(knecht);
    knecht.kuendigen();
  }
}
```

```
public class Mitarbeiter {
  // ...
  Firma arbeitgeber;

  //assoziation zum arbeitgeber herstellen
  //wird ausschliesslich vom Firma-objekt aufgerufen!!
  public void anheuern(Firma arbeitgeber) {
    this.arbeitgeber = arbeitgeber;
  }
  //assoziation abbauen, also dem knecht kuendigen
  //wird ausschliesslich vom Firma-objekt aufgerufen!!
  public void kuendigen() {
    this.arbeitgeber = null;
  }
}
```

5 Aggregation und Komposition

Bei Assoziationen wird nochmals zwischen Aggregation und Komposition unterschieden, wobei die Komposition wiederum ein Spezialfall der Aggregation ist.

Bei beiden hat ein Objekt Referenzen auf andere Objekte (beide sind ja Assoziationen). Bei der Aggregation können diese Objekte jedoch auch für sich existieren, bei der Komposition werden die Objekte aus dem Speicher entfernt, wenn auch das besitzende Objekt entfernt wird.

Die entscheidende Frage bleibt: gibt es Assoziationen, die keine Aggregationen oder Kompositionen sind?

Eine sog. *schwache Assoziation* liegt immer dann vor, wenn eine Klasse mindestens eine Methode enthält, von der ein Parameter ein Objekt einer anderen Klasse ist:

```
public class KlasseA{
  ...
}

public class KlasseB{
  ...
  public void eineMethode(KlasseA ojektA){
    ...
    tuwas mit objektA
  }
}
```

Die schwache Assoziation ist ein Fall, bei der die Assoziation keine Aggregation ist: die `KlasseA` hat kein Referenz auf Objekte der `KlasseB`. Nur wenn `eineMethode(KlasseA)` aufgerufen wird, wird eine Referenz übergeben.

5.1 Was ist nun für die Prüfung wichtig?

Wichtig ist, die Symbole mit denen in der UML Aggregation und Komposition dargestellt werden, zu kennen:

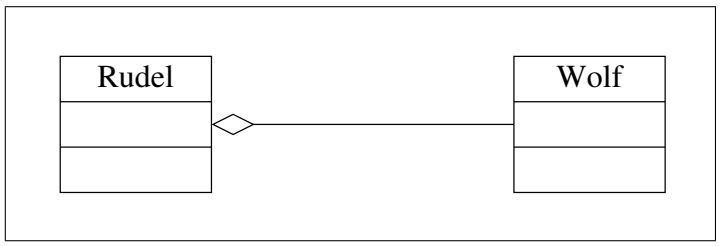

Abbildung 5: Aggregation: nicht ausgefüllte Raute

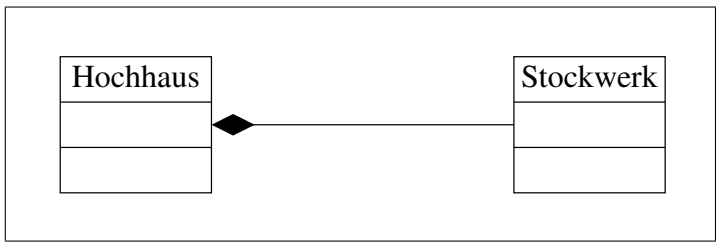

Abbildung 6: Komposition: ausgefüllte Raute

Die Raute sitzt immer an der besitzenden Klasse!!!