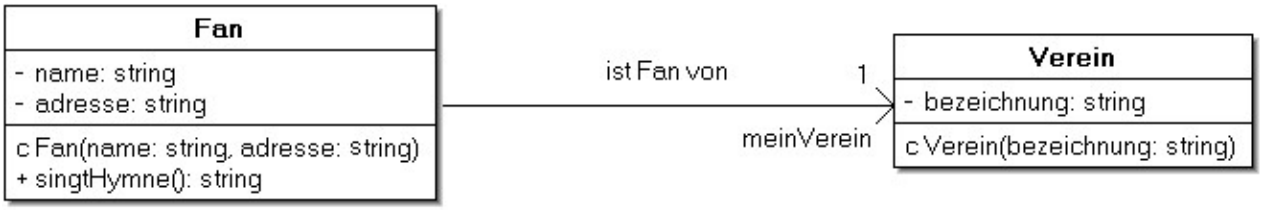
Implementieren von Assoziationen

Assoziationen werden durch zusätzliche Attribute in der Klasse implementiert.
Bei Multiplizität 1 genügt ein einfaches Attribut (in C++ ein Zeiger auf ein Objekt der Klasse). Bei Multiplizität * müssen mehrere Objekte verwaltet werden. Hierfür benötigt man entweder eine Liste von Zeigern auf die jeweiligen Objekte oder ein Feld von Zeigern auf die jeweiligen Objekte. Bei einer niedrigen exakten Multiplizität, z. B. Bowlingcenter mit 8 Bahnen, empfiehlt sich ein Feld (Array).

Zur Erinnerung, eine Assoziation in einer Richtung (unidirektional):



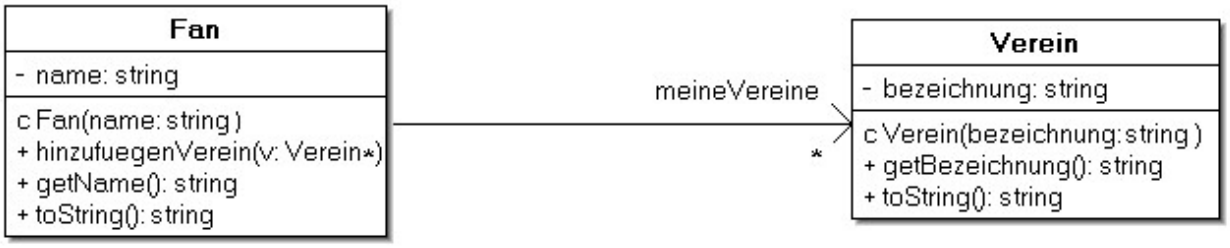
Der Fan kennt seinen Verein, umgekehrt ist das nicht der Fall. Damit ist die Assoziation nur vom Fan zum Verein navigierbar, dargestellt durch die Pfeilspitze beim Verein. Ein Fan ist Fan genau eines Vereins, das ist `meinVerein`. Der Verein weiß nichts über seine Fans.

Implementierung einer unidirektionalen Assoziation mit Multiplizität 1

Die Klasse `Fan` bekommt das neue Attribut `meinVerein` und die Methode `setMeinVerein(Verein* v)`, um dieses Attribut mit einem Wert zu versorgen.

```
class Fan {
   private:
             string name;
             string adresse;
             Verein* meinVerein;  // implementiert die Assoziation

   public:
       Fan(String name, String adresse) {
       this->name = name;
       this->adresse = adresse;
       meinVerein=NULL;
   }

   void setMeinVerein (Verein* v) {
      this->meinVerein = v;
   }

   string singtHymne(){
      return "you never walk alone...";
   }
};

class Verein {
   private:
        string bezeichnung;

   public:
       Verein(string bezeichnung) {
             this->bezeichnung = bezeichnung;
       }

};
```

Es kommt hin und wieder vor, dass ein Objekt mit nur einem Objekt einer anderen Klasse in Beziehung steht. Sollte unser Fan nun Anhänger mehrerer Vereine sein, sähe das Klassendiagramm so aus:



Implementierung einer unidirektionalen Assoziation mit Multiplizität *

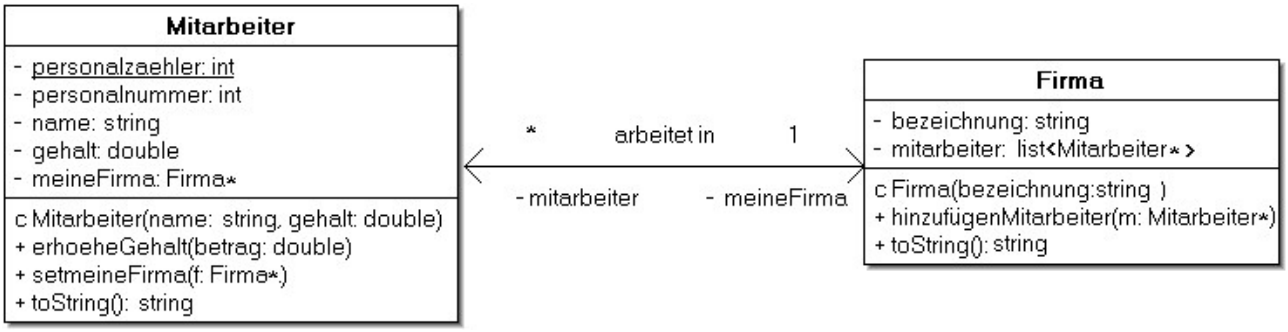
Hier reicht die Implementierung als einfaches Attribut nicht aus. Beim Fan-Objekt müssen mehrere Vereinsobjekte abgelegt werden. Hierfür wird ein Container (Behälter für mehrere Objekt-Referenzen) verwendet. Da es beliebig viele Vereine werden können, bietet sich eine Liste von Zeigern an. Die Methode kann keine einfache set-Methode mehr sein, sie heißt nun `hinzufuegenVerein (Verein* v)`.

Die Klasse **`Fan`:**

```
#include <list>

class Fan {
   private:
      string name;
      string adresse;

   std::list<Verein*> meineVereine; // implementiert die Assoziation

   public:
      Fan(string name, string adresse) {
      this->name = name;
      this->adresse = adresse;
      meinVerein=NULL;
   }

   void hinzufuegenVerein (Verein* v) {
      meineVereine.push_front(v);
   }

        string singtHymne(){
           return " you never walk alone...";
   }
};
```

Beim Verein hat sich nichts geändert, er weiß nichts vom Fan.

Implementierung einer bidirektionalen Assoziation



Hier muss ein zusätzliches Attribut in beiden Klassen eingefügt werden.

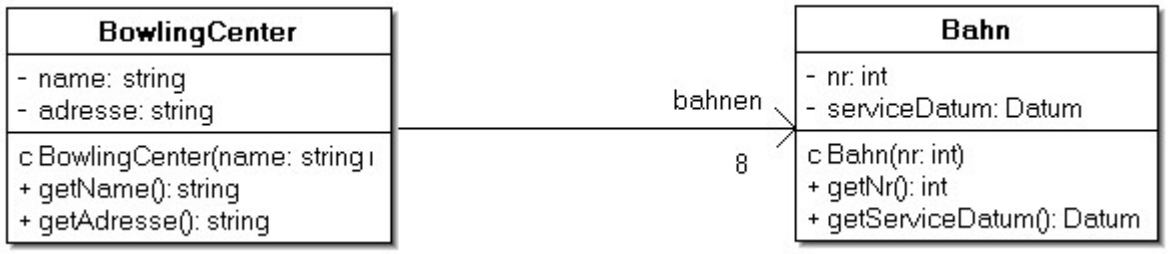
```
class Mitarbeiter {
   private:
      static int personalzaehler;
      int personalnummer;
      string name;
      double gehalt;
      Firma* meineFirma;

   public:
      Mitarbeiter(string name, double gehalt) {
      this->name = name;
      this->gehalt = gehalt;
      this->personalnummer = ++personalzaehler;
      meineFirma=NULL;
   }
   public:
      void erhoeheGehalt(double betrag) {gehalt = gehalt + betrag ;}

    void setMeineFirma(Firma* f) {this->meineFirma = f;}
   …
};
int Mitarbeiter::personalzaehler=0;

class Firma {
   private:
      string bezeichnung;
      std::list<Mitarbeiter*> mitarbeiter;

   public:
      Firma(String bezeichnung) {
            this->bezeichnung = bezeichnung;
       }
      void hinzufuegenMitarbeiter(Mitarbeiter m){
            mitarbeiter.push_front(m);
      }
…
};
```

Und eine einfache Anwendung dazu:

```
   void main() {
      // Objekte erzeugen
      Firma* haribo = new Firma("Haribo");
      Mitarbeiter* m1 = new Mitarbeiter("Bart Simpson", 1000);
      // Objekte verknüpfen
      haribo->hinzufuegenMitarbeiter(m1);
      m1->setMeineFirma(haribo);
   }
```

Implementieren einer Assoziation mit einer exakten Anzahl

Beispiel: BowlingCenter mit 8 Bahnen



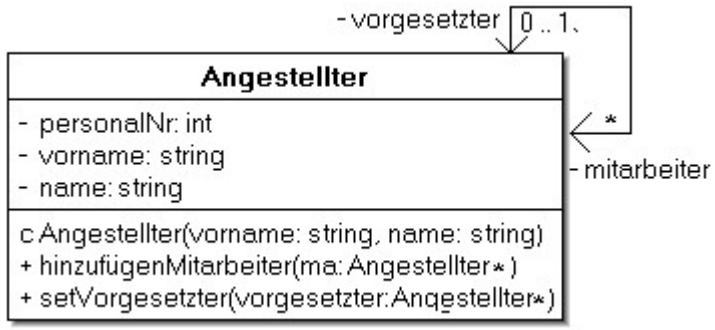
Da es genau 8 Bahnen sind und nicht damit zu rechnen ist, dass eine Bahn hinzukommt oder abgerissen wird, empfiehlt sich die Implementierung der Assoziation als Array. Damit ist ein einfacher Zugriff über einen Index möglich. Es wird genauso viel Speicherplatz reserviert wie nötig und der Zugriff ist schnell und effizient.

```
class BowlingCenter {
   private:
      string name;
      string adresse;
      Bahn* bahnen[8];

   public:
      BowlingCenter(string name) {
      this->name = name;
      this-adresse = "";

      for (int i=0; i<=7; i++ ) {
         bahnen[i] = new Bahn(i+1);     // Bahn-Objekte erzeugen
      }
   }
   …
};

class Bahn {
   private:
            int nr;
            Datum serviceDatum;

   public:
      Bahn(int nr) {
      this->nr = nr;
   }
   …
};
```

Implementieren einer reflexiven Assoziation

Hier müssen beide zusätzlichen Attribute in der einen Klasse eingefügt werden. Im Beispiel erhält die Klasse `Angestellter` zwei weitere Attribute, ein privates Attribut `vorgesetzter` vom Typ `Angestellter*` und ein privates Attribut `mitarbeiter` als List mit Zeigern auf Angestellte.

Es funktioniert aber genauso wie bei zwei verschiedenen Klassen.



Aufgabe: Implementiere die Klasse in C++!