LU09a - Mehrfachbeziehung

Aus der realen Welt sind Mehrfachbeziehungen bestens bekannt. So weist eine Schulklasse viele Lernende auf, eine Feriendestination kann von vielen Personen gebucht werden und umgekehrt kann eine Person viele Destinationen besuchen.

Diese beiden Fälle können wie folgt in UML dargestellt werden.

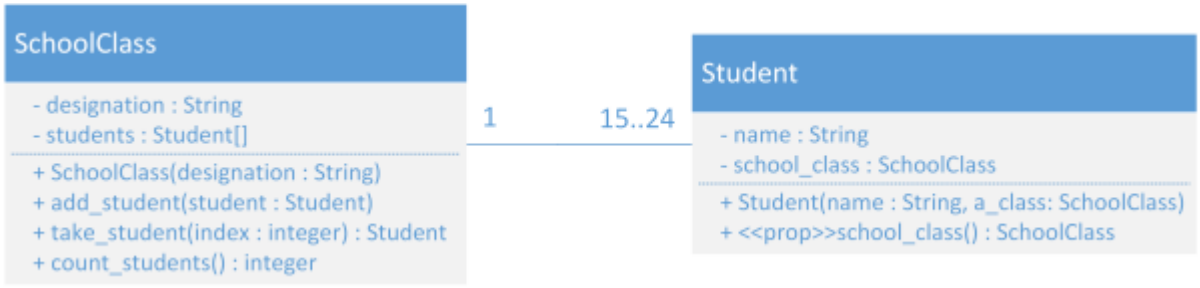

Abb: 1:n Beziehung

Kardinalität

Die Kardinalität gibt an, wie mengenmässig eine Beziehung aussieht. Vereinfacht kann man von

- 1:1 Beziehung (ein Objekt kennt genau ein anderes Objekt)
- 1:n Beziehung (ein Objekt kenntviele andere Objekte)
- n:n Beziehung (ein Objekt kennt viele andere Objekte und umgekehrt)

sprechen. Dabei steht **n** für den Begriff 'viele'. Dies wird in der UML mit einem Stern (*) wiedergegeben. Der Stern steht für **0 bis unendliche viele** Beziehungen. In der UML ist es aber auch möglich, die Kardinalität ganz genau zu spezifizieren. So ist in der Abbildung klar erkennbar, dass eine Schulklasse mindesten 15 aber maximal 24 Studenten haben kann.

Umsetzung in Python

Kennt ein Objekt viele andere, gleichartige Objekte, werden die Referenzen in einer Liste festgehalten. Ein einheitlicher Prefix bei den Methodennamen hilft bei der Orientierung.

- Die Methode für das **Zufügen** wird als **add**-Methode bezeichnet. Gegenüber dem Begriff **set** signalisiert **add**, dass eben mehrere Werte gesetzt werden können.
- Die Methode für das **Zählen** der Listeneinträge wird als **count**-Methode bezeichnet.
- Die Methode um ein Element **abzufragen** wird als **take**-Methode bezeichnet.
- Die Methode für das **Löschen** wird als **remove**-Methode bezeichnet.

Für die oben dargestellte Klasse `SchoolClass` würde das dann wie folgt aussehen (unter Einhaltung der maximalen Grösse der Klasse)

Beziehungen hinzufügen (**add**)

```
class SchoolClass:
  def __init__(self, designation):
   self._designation = designation
   self._students = []  # hier wird eine leere Liste bereitgestellt, die
dann die Referenzen der Lernenden hält.

  def add_student(a_student):
    if len(self._students) < 24:
      self._students.append(a_student)
```

Duplikate vermeiden

Die Methode `add_student` kann erweitert werden, um doppelte Einträge zu vermeiden. Dazu prüfen wir zunächst, ob das `student`-Objekt schon in der Liste vorhanden ist.

```
...
  def add_student(a_student):
    if not a_student in self._students:
      if len(self._students) < 24:
        self._students.append(a_student)
```

Anzahl Listeneinträge abfragen (**count**)

Diese Property liefert die Grösse der Liste.

```
  @property
  def count_students(self):
     return len(self._students)
```

Lesen eines Objekts (**take**)

Diese Methode dient dazu, ein bestimmtes Element aus der Liste zu lesen. Dies erfolgt entweder ...

- ... über den Index des Listenelements
- ... einen eindeutigen Wert (Key)

```
def take_student(self, index=None, key=None):
  if index is not None:   # Index wurde angegeben
    if index < len(self._students):
      return self._students[index]
    else:
      raise StudentIndexError('search')
  else:   # kein Index angegeben, wir suchen nach dem Namen
```

```
    for student in self._students:  # Loop über alle student-objekte
      if student.name == key:
        return student
    return None
```

Löschen einer Beziehung (**remove**)

Diese Methode dient dazu, ein bestimmtes Element aus der Liste zu löschen. Dies erfolgt entweder ...

- ... über den Index des Listenelements
- ... einen eindeutigen Wert (Key)

```
def remove_student(self, index=None, key=None):
  if index is not None:   # Index wurde angegeben
    if index < len(self._students):
      self._students.remove(idx)
    else:
      raise StudentIndexError('remove')
  else:   # kein Index angegeben, wir suchen nach dem Namen
    for idx, student in enumerate(self._students):  # Loop über alle
student-objekte
      if student.name == key:
        self._students.remove(idx)
```

Je nach Entwurf der Applikation können die **take**- und **remove**-Methoden auch nur einen der Parameter `index` bzw. `key` anbieten.

M320-LU09