2. Mehrfachbeziehungen

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.

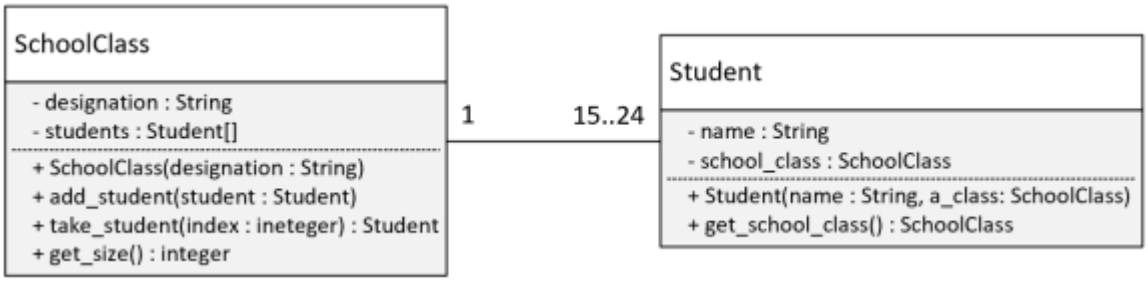

Abb 5.9 : 1:n Beziehung

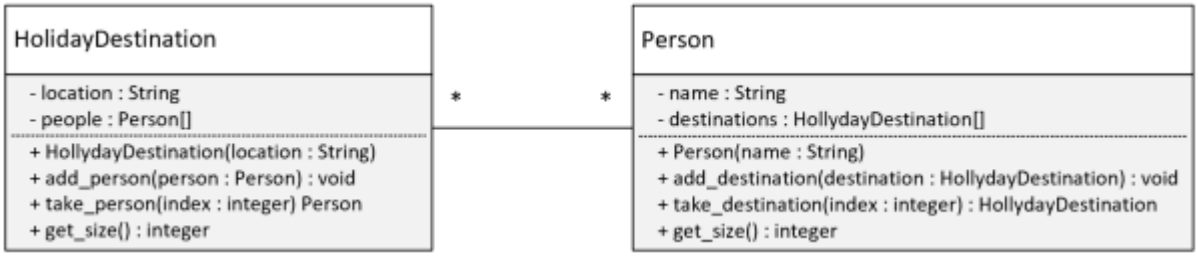

Abb 5.10 : n:n Beziehung

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 5.9 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. In der Regel wird die Methode für das **Zufügen** als **add**-Methode bezeichnet. Gegenüber dem Begriff **set** signalisiert **add**, dass eben mehrere Werte gesetzt werden können.
Für die oben dargestellte Klasse `SchoolClass` würde das dann wie folgt aussehen (unter Einhaltung der maximalen Grösse der Klasse)

```
class SchoolClass:
```

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

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

Weitere wichtige Methoden dienen

- dem **Abfragen** der Grösse der Liste (Anzahl der gespeicherten Objekte)
- dem **Abrufen** einer Referenz über einen Index. Dieser muss abger gegen den size der Liste geprüft werden, damit kein unerlaubter Zugriff erfolgt!
- dem **Entfernen** einer Referenz aus der Liste

```
  @proprty
  def size(self):
     return len(self._students)

  def take_student(self, index):
    if index < len(self._students):
       return self._students[index]
    else:
       raise StudentIndexError('search')

  def remove_student(self, index):
    if index < len(self._students):
       self._students.remove(index)
    else:
      raise StudentIndexError('remove')
```
