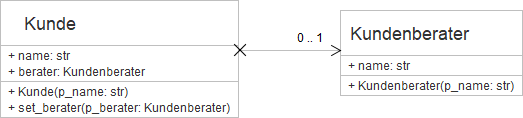
# Assoziationen

## Assoziationen Implementierung

### 0..1-Assoziation (KANN-Assoziation)

Bei der 0..1-Assoziation kann es kein oder ein assoziiertes Objekt geben. Bespielweise kann ein Kunde einer Bank einen Kundenberater haben. Falls der Kunde nur Onlinebanking betreibt und keinen persönlichen Kundenberater haben möchte, soll auch die Möglichkeit bestehen, dass für einen Kunden kein Kundenberater gespeichert wird.



|  |  |  |
| --- | --- | --- |
| kunde.py | Zeile | kundenberater.py |
| class Kunde:  def \_\_init\_\_(self, p\_name):  self.name = p\_name  self.berater = None  def set\_berater(self, p\_berater):  self.berater = p\_berater | 1  2  3  4  5  6  7 | class Kundenberater:  def \_\_init\_\_(self, p\_name):  self.name = p\_name |

Vom Kunden soll eine Navigation zum Kundenberater möglich sein. Da die Untergrenze an Kundenberatern für einen Kunden 0 ist, stellt die Beziehung eine KANN-Assoziation dar. Das heißt, es kann auch vorkommen, dass ein Kunde keinen Kundenberater hat. Das bedeutet, dass ein Attribut *berater* deklariert aber nicht initialisiert wird. Dies geschieht mit der Zuweisung *None*. Für die Zuweisung eines Kundenberaters, also eines Objekts der Klasse *Kundenberater*, wird die set-Methode *set\_berater(self, p\_berater)* implementiert.

|  |  |
| --- | --- |
| Zeile | hauptprogramm.py |
| 1  2  3  4  5  6  7  8 | from kunde import Kunde  from kundenberater import Kundenberater  b1 = Kundenberater("Max Rat")  b2 = Kundenberater("Samuel Lampe")  k1 = Kunde("Simon Amsel")  k1.set\_berater(b1)  print("Kunde:", k1.name, "- Berater:", k1.berater.name) |

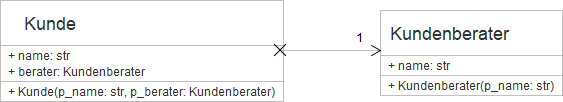
Ausgabe

Kunde: Simon Amsel - Berater: Max Rat

Im Hauptprogramm kann nun die Beziehung zwischen einem Kunden und einem Kundenberater hergestellt werden. Dafür müssen zuerst Objekte beider Klassen deklariert und initialisiert werden (Zeile 4-6). Im Anschluss wird über die Methode *set\_berater* einem Kunden ein Kundenberater zugeordnet. Im implementieren Beispiel wird dem Objekt *k1* (Objekt der Klasse *Kunde*) das Objekt *b1* als *Berater* zugeordnet, indem die Methode *set\_berater* das Objekt *b1* in dem Attribut *self.berater* speichert. In Zeile 8 ist zu sehen, dass über das Objekt *k1* auf den *Namen* des *Beraters* navigiert werden kann. Eine Navigation vom Kunden auf den Berater ist somit möglich.

### 1-Assoziation (MUSS-Assoziation)

Bei der 1-Assoziation muss es genau ein assoziiertes Objekt geben. Beispielsweise könnte eine Bank wollen, dass jedem Kunden ein Kundenberater zugeordnet ist. Ein Kunde aber auch nicht von mehr als einem Bankberater beraten werden darf.



|  |  |  |
| --- | --- | --- |
| kunde.py | Zeile | kundenberater.py |
| class Kunde:  def \_\_init\_\_(self, p\_name, p\_berater):  self.name = p\_name  self.berater = p\_berater | 1  2  3  4 | class Kundenberater:  def \_\_init\_\_(self, p\_name):  self.name = p\_name |

Vom Kunden soll eine Navigation zum Kundenberater möglich sein, weswegen in der Klasse *Kunde* ein Attribut *berater* implementiert wurde. Da es sich jetzt allerdings um eine MUSS-Assoziation handelt, d. h. es genau ein assoziiertes Objekt geben muss, wird der Nutzer beim Konstruktoraufruf gezwungen, einen Berater als Übergabeparameter mitzugeben (*p\_berater*). Es ist folglich nicht mehr möglich einen Kunden anzulegen, ohne ihm einen Berater zuzuweisen.

|  |  |
| --- | --- |
| Zeile | hauptprogramm.py |
| 1  2  3  4  5  6  7 | from kunde import Kunde  from kundenberater import Kundenberater  b1 = Kundenberater("Max Rat")  b2 = Kundenberater("Samuel Lampe")  k1 = Kunde("Simon Amsel", b1)  print("Kunde:", k1.name, "- Berater:", k1.berater.name) |

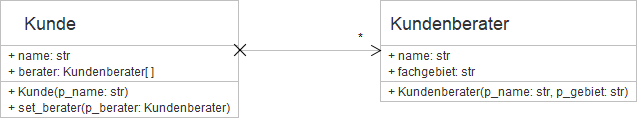
Ausgabe

Kunde: Simon Amsel - Berater: Max Rat

Beim Konstruktoraufruf der Klasse Kunde muss dem Konstruktor ein *Kundenberater* als *Übergabeparameter* übergeben werden. Somit kann erzwungen werden, dass jedem Kunden ein Kundenberater zugeordnet ist (MUSS-Assoziation).

### \*-Assoziation (KANN-Assoziation)

Bei der \*-Assoziation gibt es kein, ein oder beliebig viele assoziierte Objekte. Demnach könnte eine Bank das Prinzip des Fachberaters festlegen. Das bedeutet, dass jeder Berater ein Fachgebiet hat (bspw. Anlageberater, Kreditberater usw.). Diese neuen Rollen haben zur Folge, dass ein Kunde mehrere Berater haben kann. Für alle Geldanlagen hat der Kunde einen Anlageberater. Dagegen steht dem Kunden bei der Kreditberatung ein anderer Berater mit dem entsprechenden Fachgebiet zur Seite. Außerdem soll es wieder möglich sein, dass Kunden Onlinebanking betreiben und dabei keinen Berater haben.



|  |  |  |
| --- | --- | --- |
| kunde.py | Zeile | kundenberater.py |
| class Kunde:  def \_\_init\_\_(self, p\_name):  self.name = p\_name  self.berater = []  def add\_berater(self, p\_berater):  self.berater.append(p\_berater) | 1  2  3  4 | class Kundenberater:  def \_\_init\_\_(self, p\_name, p\_gebiet):  self.name = p\_name  self.fachgebiet = p\_gebiet |

Da bei einer \*-Assoziation nicht nur ein, sondern viele Objekte gespeichert werden können, wird ein Array (Liste) zum Implementieren der Beziehung verwendet. Die Datenstruktur Array (Liste) bietet die Möglichkeit beliebig viele Elemente vom Typ Kundenberater zu speichern. Da ein Kunde auch keinen Kundenberater hat, wird im Konstruktor ein leerer Array implementiert (vgl. Zeile 4). In der Methode *set\_berater* wird auf dem Array *berater* die Methode *append* ausgeführt, die ein Objekt der Klasse Kundenberater dem Array hinzufügt. Da es sich um eine KANN-Assoziation handelt, d. h. ein Kunde auch keinen Berater haben kann, muss im Konstruktor kein Berater übergeben werden. Der Array kann somit auch leer bleiben.

|  |  |
| --- | --- |
| Zeile | hauptprogramm.py |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | from kunde import Kunde  from kundenberater import Kundenberater  b1 = Kundenberater("Max Rat", "Anlageberater")  b2 = Kundenberater("Samuel Lampe", "Kreditberater")  k1 = Kunde("Simon Amsel")  k1.add\_berater(b1)  k1.add\_berater(b2)  print("Kunde:", k1.name)  print("Berater:")  for berater in k1.berater:  print(berater.name, "(", berater.fachgebiet, ")") |

Ausgabe

Kunde: Simon Amsel

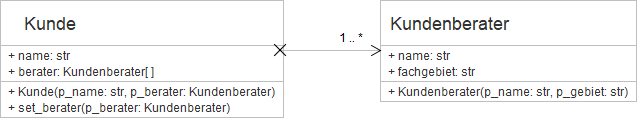
Berater:

Max Rat ( Anlageberater )

Samuel Lampe ( Kreditberater )

### 1..\*-Assoziation (MUSS-Assoziation)

Die 1..\* Assoziation legt fest, dass ein oder beliebig viele assoziierte Objekte bestehen. Bezogen auf das Bankbeispiel könnte die Bank wiederum festlegen, dass ein Kunde mehrere Berater haben kann (Anlageberater und Kreditberater), allerdings mindestens einen Berater haben muss.



|  |  |  |
| --- | --- | --- |
| kunde.py | Zeile | kundenberater.py |
| class Kunde:  def \_\_init\_\_(self, p\_name, p\_berater):  self.name = p\_name  self.berater = [p\_berater]  def add\_berater(self, p\_berater):  self.berater.append(p\_berater) | 1  2  3  4  5  6  7 | class Kundenberater:  def \_\_init\_\_(self, p\_name, p\_gebiet):  self.name = p\_name  self.fachgebiet = p\_gebiet |

Im Vergleich zur \*-Assoziation unterscheidet sich die Implementierung nur darin, dass dem Benutzer „verboten“ werden muss, ein Objekt der Klasse *Kunde* zu erzeugen, dass keinen Berater hat. Somit muss in den Konstruktor der Übergabeparameter *p\_berater* aufgenommen werden. Der Array *berater* wird dann mit dem Übergabeparameter *p\_berater* initialisiert.

|  |  |
| --- | --- |
| Zeile | hauptprogramm.py |
| 1  2  3  4  5  6  7  8  9  10  11  12 | from kunde import Kunde  from kundenberater import Kundenberater  b1 = Kundenberater("Max Rat", "Anlageberater")  b2 = Kundenberater("Samuel Lampe", "Kreditberater")  k1 = Kunde("Simon Amsel", b1)  k1.add\_berater(b2)  print("Kunde:", k1.name)  print("Berater:")  for berater in k1.berater:  print(berater.name, "(", berater.fachgebiet, ")") |

Ausgabe

Kunde: Simon Amsel

Berater:

Max Rat ( Anlageberater )

Samuel Lampe ( Kreditberater )