# Sichtbarkeit von Attributen

Wir haben inzwischen intensiv mit Klassen und Objekten gearbeitet. Dabei haben wir schon viele Methoden erstellt. Nun beschäftigen wir uns mit einem Thema, das vor allem professionelle Softwareentwickler betrifft – der Kapselung.

## Theoretische Grundlagen

Die Idee der Kapselung besteht darin, die Implementierung der Attribute vor dem Nutzer des Objektes zu verbergen. Sie sind dann privat. Der Nutzer kann die Attributwerte eines Objektes durch set- und get-Methoden jederzeit festlegen und auslesen.

**Beispiel Kontoverwaltung**

|  |  |
| --- | --- |
|  | Die Attribute sind statt wie gewöhnlich mit einem Pluszeichen (+) nun mit einem Minuszeichen (-) gekennzeichnet. Das bedeutet, dass der Nutzer im Hauptprogramm keinen Zugriff darauf hat. Er muss zwingend eine entsprechende get-Methode zum Auslesen und eine set-Methode zum Festlegen eines Attributwertes anwenden. |
| Fachklasse Konto |  |

## Quellcode in Python (siehe *L1\_6\_Info.py*)

### Private Attribute deklarieren

In Python werden private Attribute wie folgt deklariert und initialisiert.

|  |  |
| --- | --- |
| 1  2  3  4  5 | class Konto:  def \_\_init\_\_(self):  self.\_\_inhaber = ""  self.\_\_kontonummer = 0  self.\_\_kontostand = 0 |

Zeilen 3-5: Statt inhaber, kontonummer und kontostand heißen die Attribute nun \_\_inhaber, \_\_kontonummer und \_\_kontostand (mit zwei Unterstrichen). Dadurch wird gekennzeichnet, dass sie privat sind und im Hauptprogramm nicht direkt aufgerufen werden können.

|  |  |
| --- | --- |
| 1  2  3 | # Hauptprogramm  k = Konto()  print(k.\_\_kontostand) |

Zeile 3: Dieser Aufruf führt zwangsläufig zu einem Fehler, da das Attribut \_\_kontostand privat ist und aus dem Hauptprogramm nicht aufgerufen werden kann.

### Set- und get-Methoden definieren

Um den Zugriff auf die Attribute dennoch sicherzustellen, erfolgt dieser über eine get- und set-Methode, die zunächst definiert werden müssen.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | class Konto:  ...  def get\_inhaber(self):  return self.\_\_inhaber  def set\_inhaber(self, p\_inhaber):  self.\_\_inhaber = p\_inhaber |

Zeile 4: Die Methode get\_inhaber() wird definiert. Parameter ist nur das Objekt selbst.

Zeile 5: Diese Methode liefert das Attribut \_\_inhaber des aktuellen Objektes zurück.

Zeile 7: Die Methode set\_inhaber() wird definiert. Parameter ist zusätzlich eine Variable p\_inhaber, in die beim Aufruf der zu setzende Wert geschrieben wird.

Zeile 8: Der Wert des Parameters p\_inhaber wird dem Attribut \_\_inhaber des aktuellen Objektes zugewiesen.

### Set- und get-Methoden anwenden

Im Hauptprogramm werden set- und get-Methoden wie folgt aufgerufen:

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Hauptprogramm  k = Konto()  k.set\_inhaber("Paul Schmidt")  print(k.get\_inhaber()) |

Zeile 4: Die Methode set\_inhaber() wird für das Objekt k mit dem zu setzenden Wert "Paul Schmidt" aufgerufen. Somit wird dieser Wert durch die Methode in dem Attribut \_\_inhaber des Objektes k gespeichert.

Zeile 5: Die Methode get\_inhaber() wird für das Objekt k aufgerufen. Der Wert des Attributes wird zurückgegeben und mit print() in die Konsole geschrieben.