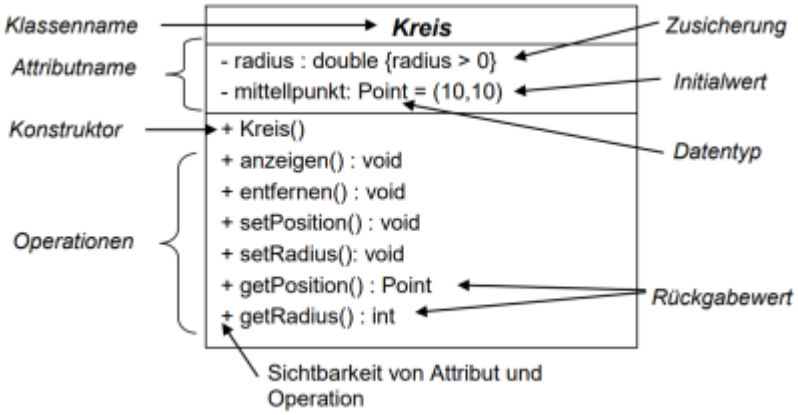
1 Klasse

Definition

Eine Klasse ist die Definition der Attribute, Operationen (Methoden) und der Semantik für eine Menge von Objekten. Alle Objekte einer Klasse entsprechen dieser Definition.

UML-Notation



Klassenname

Er stellt zum einen eine sinnvolle Beschreibung der Klasse dar, ist aber auch der Datentyp der Klasse. Der Name der Klasse soll durch ein Substantiv ausgedrückt werden.

```
class Circle:
  ....
```

Attributname und Datentyp

Attribute stellen die in der Klasse benötigten Variablen dar.
In der OO-Welt gilt die Regel des „data hiding“. D.h. dass die Sichtbarkeit der Attribute (Variablen) auf private (- im Klassendiagramm) lauten sollte.
Attribute die private sind, müssen mit `self.__` im Namen beginnen und werden im Konstruktor initialisiert.
Hinweis: In Python wird der Datentyp zur Laufzeit festgelegt.

```
class Kreis:
  def __init__(self, value):
    self.radius   = value                # set-Methode für Attribut radius
aufrufen
    self.__center = Point(10.0, 10.0)    # Mittelpunkt wird als Objekt vom
```

```
Typ Point referenziert.
```

Konstruktor und Initalisierung

Der Konstruktor wird ausgeführt, wenn eine Klasse erzeugt wird. Er muss gleich heissen wie die Klasse.
Wird für ein Attribut ein Initialwert verlangt, so muss dieser im Konstruktor übergeben werden.

```
  circel = Circle(15.5);
```

Operationen und Rückgabewert

Operationen dienen dazu, die Attributswerte eines Objektes zu setzen, zu lesen oder den inneren Zustand des Objekts zu verändern.
Operationen können je nach Bedarf unterschiedliche Sichtbarkeiten erhalten (private, friendly, protected, public)
Entsprechend der Notation liefert eine Methode einen Rückgabewert (ausser bei void)

```
  # Methode ohne Wertrückgabe (oft als Prozedur bezeichnet)
  def move_point(self, to_x, to_y):
      self.x = to_x
      self.y = to_y
  # Methode mit Wertrückgabe (oft als Funktion bezeichnet)
  def is_radius_set(self):
    if radius != 0.0:
      return True
    else:
      return False
```

setter und getter

Methoden die zum Schreiben bzw. Lesen eines Attributs verwendet werden, werden als setter und getter bezeichnet.
In Python spricht man bei privaten Attributen auch von Properties (Eigenschaften) und hat für deren Nutzung eine eigene Deklaration.

getter

Getter dienen dem Auslesen des Wertes eines Attributs.

```
  @property
  def radius(self):
    return self.__radius
```

setter

Setter dienen dem Setzen des Wertes eines Attributs.

```
  @radius.setter
  def radius(self, value):
    self.__radius = value
```

Nutzung

Properties können im Code wie Variablen genutzt werden. Es braucht keinen expliziten Methodenaufruf, um deren Werte zu lesen oder zu schreiben.
Implizit wird aber der entsprechende Code ausgeführt. Somit sind auch Prozesse wie z.B. eine Wertzusicherung möglich.

```
  # lesen eines Property (über die getter Methode)
  print("Radius des Kreises " + str(circle.radius))
  # schreiben/setzen eines Property (über setter Methode)
  circle.radius = 22.45
```

Zusicherung

Der Code muss diese Anforderung erfüllen. Dafür ist in den entsprechenden Operationen zu sorgen.

```
  @radius.setter
  def radius(self, value):
    if radius > 0.0:
      self.__radius = value
```

Sichtbarkeit

Die Sichtbarkeit sagt aus, wer auf die entsprechenden Attribute und Operationen Zugriff hat

- - private Zugriff nur innerhalb der Klasse
- # protected Zugriff innerhalb des Vererbungsbaums
- friendly Zugriff im gleichen Paket (gilt bei Java)
- + public Zugriff von überall

—- © René Probst

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/archiv/m320/merkblaetter/merkblatt_1

Last update: **2026/08/30 14:09**