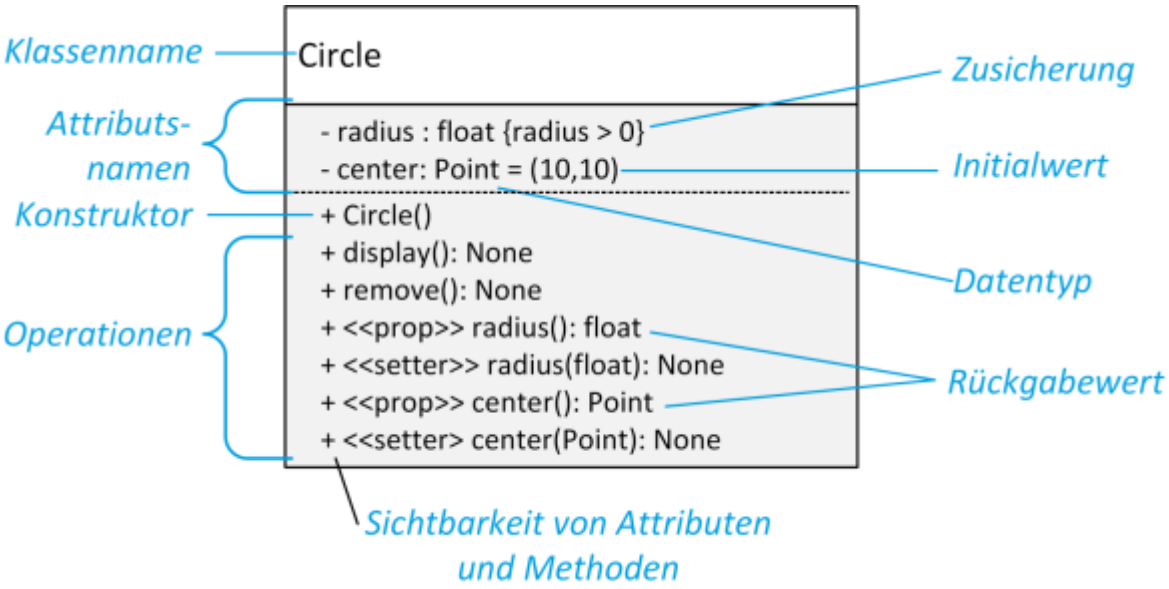
Merkblatt: Klasse

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 wird durch ein Substantiv ausgedrückt.

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

Attributname und Datentyp

Attribute stellen die in der Klasse benötigten Variablen dar.
In der OO-Welt gilt die Regel des „data hiding“. Das heisst, dass die Sichtbarkeit der Attribute (Variablen) **private** (**-** im Klassendiagramm) sein sollte.

In Python werden private Attribute gekennzeichnet, indem der Attributsname mit einem Underscore (z.B. `_radius`) beginnen. Dies zeigt an, dass dieses Attribut nicht direkt gelesen bzw. verändert werden sollte.

In Python wird der Datentyp in der Regel 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.

In Python heisst der Konstruktor `__init__`.

Wird für ein Attribut ein Initialwert verlangt, so muss dieser im Konstruktor übergeben werden.

```
  circle = 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 None)

```
  # 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
```

Im UML-Klassendiagramm kennzeichnen wir diese Methoden mit «`prop`».

setter

Setter dienen dem Setzen des Wertes eines Attributs.

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

Im UML-Klassendiagramm kennzeichnen wir diese Methoden mit «`setter`».

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 Verarbeitungen 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 value > 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
- **+** public ⇒ Zugriff von überall

Im Gegensatz zu anderen Programmiersprachen erzwingt Python keine Sichtbarkeit. Bei privaten Attributen und Methoden ist es üblich, einen Underscore am Anfang des Namens zu schreiben. Dies ist aber nur informativ und verhindert den direkten Zugriff nicht.
