5. Klassendiagramm und Klasse

In einem ersten Schritt beschränken wir uns auf die Darstellung einer Klasse in der Notationssprache UML (Unified Modelling Language).

Abb 1.8: Klassendigramm nach UML
https://de.wikipedia.org/wiki/Klassendiagramm

Die «Übersetzung» des Klassendiagramms gemäss Abb. 1.8 in den entsprechenden Code finden Sie unten in Beispiel 1.5.

Erklärung zu den Elementen des Klassendiagramms:

- **Attribute** sollten immer private deklariert werden, um deren Wertebereich garantieren zu können. [*Die Erklärung zu data hiding folgt in Kapitel 5*]
- **Konstruktoren** werden bei der Erzeugung eines Objekts als erstes ausgeführt und dienen primär dem Initilaisieren der Attribute.
- **Methoden** stellen die Funktionalität der Klasse dar. Oft wird die Gesamtheit aller Methoden auch als «die Schnittstelle» der Klasse bezeichnet.

Im folgenden Code-Stück werden diese Elemente zusätzlich beschrieben.

Beispiel 1.5: Abstraktion einer Türe

```
class Door:
    """
    Diese Klasse beschreibt eine Türe mit der Eigenschaft color (Farbe) und
den Zuständen
    door_is_open (für geöffnete Türe) sowie door_is_locked (für verriegelte
Türe).
    Die Türe überwacht die beiden Zustände und verhindert so Aktionen, die
nicht möglich sind.
    Das Verriegeln selber delegiert die Türe an ein Objekt vom Typ Door_lock
(Türschloss).
    """
# Mit dem Keyword def wird eine Funktion bzw. eben ein Konstruktor
deklariert.
# Der Konstruktor trägt IMMER den Namen __init__ und weist als ersten
Parameter den Wert self auf.
# Danach folgen die Übergabeparameter, deren Werte dann den Attributen
zugewiesen werden.
# Attribute können aber auch mit einem fixen Wert initialisiert werden.
# Konstruktoren werden als Erstes im Programm angeschrieben.
    def __init__(self, ref2door_lock, base_color):
        """
        Erzeugt ein Tür-Objekt.
        :param ref2door_lock:
        :param base_color:
```

```
        """
# ein privates Attribut muss im Konstruktor initialisiert werden und ist
dann in der Klasse
# über self.__name_des_Attributs ansprechbar.
        self._the_door_lock  = ref2door_lock
# Hier wird der Setter eines Attributs aufgerufen (siehe unten)
        self.color           = base_color
        self._door_is_open   = False
        self._door_is_locked = False

# Nach den Konstruktoren folgen Methoden, die eine Verarbeitung auslösen.
# Danach folgen Methoden, die auf ein Ereignis reagieren
    def open_the_door(self):
        """
        Methode für das öffnen der Türe.
        Das ist aber nur möglich, wenn die Türe nicht verriegelt ist.
        """
        if self._door_is_locked == False:
            self._door_is_open = True

    def close_the_door(self):
        """
        Methode für das schliessen der Türe.
        Das geht immer, auch wenn die Türe schon geschlossen oder verriegelt
ist. Der Zustand ändert dann nämlich nicht.
        """
        self._door_is_open = False

    def lock_the_door(self):
        """
        Methode für das verriegeln der Türe.
        Das ist nur möglich, wenn die Türe nicht offen ist.
        Für das verriegeln ist aber das Türschloss zuständig. Es weiss wie
das geht.
        """
        if self._door_is_open == False:
            self._door_is_locked = self._the_door_lock.lock()

    def unlock_the_door(self):
        """
        Methode für das entriegeln der Türe
        Das ist nur möglich, wenn die Türe verriegelt ist.
        Für das entriegeln ist aber das Türschloss zuständig. Es weiss wie
das geht.
        """
        if self._door_is_locked:
            self._door_is_locked = self._the_door_lock.unlock()

    def test(self):
        """
```

```
        schreibt alle Attribute in den StdOut
        """
        print(f'Türfarbe : {self.color}')
        print(f'Türe offen: {self._door_is_open}')
        print(f'Türe verriegelt: {self._door_is_locked}')

# Am Ende folgen die getter- und setter-Methoden für die Attribute der
Klasse
# getter werden mit der Anotation @property markiert.
    @property
    def door_is_open(self):
        """
        getter-Methode für den Zustand door_is_open
        :return: true, wenn die Türe offen ist, sonst false
        """
        return self._door_is_open

    @property
    def door_ist_locked(self):
        """
        getter-Methode für den Zustand door_is_locked
        :return: true, wenn die Türe verriegelt ist, sonst false
        """
        return self._door_is_locked

    @property
    def color(self):
        """
        getter-Methode für die Eigenschaft color
        :return: die Farbe des Objekts
        """
        return self._color

# setter werden mit der Anotation @name.setter markiert.
    @color.setter
    def color(self, new_color):
        """
        setter-Methode für die Eigenschaft color
        :param new_color:
        """
        self._color = new_color



"""
nur für die korrekte Übersetzung und Ausführung
"""
class DoorLock:
    """
    dummy Klasse, damit in der Klasse Tuere kein Fehler auftritt
    """
```

```
    def __init__(self):
        print("ein Schloss erzeugt")

    def lock(self):
        return True

    def unlock(self):
        return False

# Hier die main-Methode festlegen
if __name__  == '__main__':
    print('Test für Tür-Objekt')
    the_door_lock = DoorLock()
    the_door      = Door(the_door_lock, 'grün')
    the_door.test()
    print('-- Türe jetzt öffnen')
    the_door.open_the_door()
    the_door.test()
```

© René Probst

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

Permanent link:
https://wiki.bzz.ch/modul/archiv/m320/learningunits/lu01/theorie/lu1-kapitel_4?rev=1788091600

Last update: **2026/08/30 14:06**