4. Ist Klasse gleich Klasse?

Warum stellen wir uns diese Frage?

Sie haben im Modul 319 in der LU12 (Link) einen ersten Einblick erhalten. Dabei haben Sie eine Klasse als Ansammlung vieler Attribute kennengelernt. Man nennt solche Klassen (informell) auch Datenklassen (POJO - Plain Old Java Object).
In manchen Programmiersprachen oder Frameworks wird der Begriff „Datenklasse“ verwendet, um eine Klasse zu beschreiben, die hauptsächlich dazu dient, Daten zu speichern, ohne viel Verhalten zu haben. Solche Klassen haben normalerweise private Datenfelder (Variablen) und öffentliche Getter- und Setter-Methoden, um auf die Daten zuzugreifen und sie zu setzen. Sie enthalten in der Regel keine komplexe Geschäftslogik oder speziellen Verhaltensmethoden.
Eine „allgemeine“ Klasse kann ein komplexeres Objekt mit verschiedenen Verhaltensmethoden aufweisen.

Beispiel 1.3: Dataklasse und Standardklasse

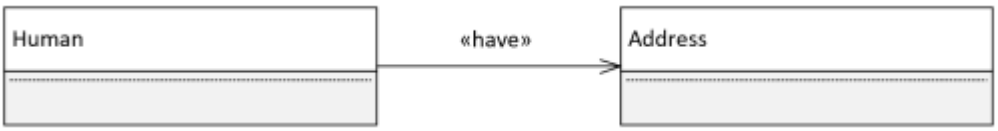

Abb 1.7: Beschreibung einer Person mit deren Adresse
Wir finden in Abbbildung 1.7 die zwei Klassen `Human` und `Address`, die in einer einseitigen Beziehung „Mensch besitzt Adresse“ stehen.
Die Klasse `Human` beschreibt eine Person mit all ihren Eigenschaften und Fähigkeiten, was wohl erkennbar eine sehr komplexe Klasse sein wird, während `Address` die Adresse der Person beschreibt. Die Adresse wird einige wenige Attribute halten wie

- street
- number
- postal_code
- city

die durch set- und get-Methoden geschrieben (set) bzw. gelesen (get) werden. Weitere Methoden werden kaum benötigt. Es handelt sich hier um eine typische Datenklasse.
Dagegen wird die Person durch eine Vielzahl von Eigenschaften wie z.B.

- size
- hair_color
- weight
- …

und Methoden wie

- breath()
- eat()
- go()
- …

beschrieben. Zu diesen Attributen finden sich dann auch wieder set- und get-Methoden.

Was aber bedeutet das nun für die Programmierung in Python?
Wir betrachten uns hier die beiden Klassen `Human` und `Address` sowie deren Umsetzung in Code. Für die Person werden einige wenige Attribute und Methoden stellvertretend angeschrieben.

Beispiel 1.4: Codierung einer Datenklasse und einer „standard“ Klasse

Address	**Human**

```
from dataclasses import dataclass


@dataclass
class Address:
    """
    Beschreibt eine Privatadresse mit Strasse,
Hausnummer, Postleitzahl
    und Ort.
    Die Attribute können über den Konstruktor
und/oder die set-Methoden
    geschrieben und über die get-Methoden gelesen
werden.
    """
    street: str = None
    number: str = None  # als Zeichenkette
deklariert,
    # da auch Hausnummern wie 11A möglich sind
    postal_code: int = 0
    city: str = None

    # Durch den Dekorierer (Decorator) @dataclass
sind sowohl Konstruktor
    # wie auch get- und set-Methoden implizit
verfügbar
    # und müssen nicht extra angeschrieben werden.


"""
Hier folgt die main-Methode über die das Programm
ausgeführt werden kann.
"""

if __name__ == '__main__':
    # Instanzieren eines Objektes über den
Konstruktor.
    # address ist eine Objekt-Variable, während
Adress(...) den Konstruktor
    # der Klasse ausführt, um das Objekt zu
erzeugen
    address = Address(street='Musterstrasse',
                      number='11A',
                      postal_code=9999,
                      city='Musterdorf')
    #
    # Ausgabe der Werte über impliziten Aufruf der
jeweiligen Methoden.
    print('Werte des Objekts address:')
    print(f'{address.street} {address.number} '
          f'{address.postal_code} {address.city}')
# nutzen der get-Methoden
    #
    address.postal_code = 1001  # nutzen der set-
Methoden
    print(f'{address.street} {address.number} '
          f'{address.postal_code} {address.city}')
```

```
from address import Address

class Human:
    """
    Beschreibt eine Person mit ihren Eigenschaften
und Fähigkeiten.
    Hier in einer gekürzten Version, da eine
umfassende Beschreibung
    zu viel Platz einnehmen würde.
    """

    # Hier wird ein expliziter Konstruktor
angeschrieben.
    # Mehr zur Bedeutung des Konstruktors folgt
später.
    def __init__(self, name, weight, eye_color,
one_address):
        self._name      = name
        self._weight    = weight
        self._eye_color = eye_color
        # hier können ganz viele weitere Attribute
stehen, die durch
        # den Konstruktor initialisiert werden.
        self._address = one_address #eine Referenz
zu einem Address-Objekt

    # Attribute werden über get-Methoden gelesen.
    # Python kennt dazu den Dekorator @property.
    # Hier als Beispiel drei Methoden.
    @property
    def weight(self):
        return self._weight

    @property
    def name(self):
        return self._name

    @property
    def address(self):
        return self._address

    # Attribute werden über set-Methoden gesetzt.
    # ACHTUNG: Es gibt Attribute, die einmalig bei
der Initialisierung
    # gesetzt werden, hier z.B. die Augenfarbe.
    # Python nutzt hier den @.setter Dekorator.
    # Hier als Beispiel wieder nur eine Methode.
    @address.setter
    def address(self, new_address):
        self._address = new_address

    # Und hier folgen Methoden, die das Verhalten
des Objekts
    # beeinflussen.
    def eat(self, what_ever):
         self._weight += 0.3

"""
Und nun ein Stück Programmcode, um den Effekt zu
testen
"""
if __name__ == '__main__':
    address = Address(street='Musterstrasse',
                      number='11A',
                      postal_code=9999,
                      city='Musterdorf')
    milli   = Human('Milli', 50, 'blue', address) #
Initialgewicht ist 50
    milli.eat('chips')                            #
wer isst nimmt zu
    print(f'Name: {milli.name}')
    print(f'Adresse: {milli.address}')
    print(f'Gewicht: {milli.weight}')             #
das Ergebnis sieht man ;-)
```

Ausgabe:
```
Werte des Objekts address:
Musterstrasse 11A 9999 Musterdorf
Musterstrasse 11A 1001 Musterdorf
```

Ausgabe:
```
Name: Milli
Adresse: Address(street='Musterstrasse',
number='11A', postal_code=9999, city='Musterdorf')
Gewicht: 50.3
```

Sie haben nun an Hand der Klasse Human den Aufbau einer Klasse mit all ihren Details gesehen. Ebenso könenn Sie feststellen, dass eine „Datenklasse“ in Python mittels dem `@dataclass` Dekorator um ein vielfaches einfacher zu erstellen ist, da weder Konstruktor noch Methoden notwendig sind.
Anmerkung: Sie können natürlich auch einer „Datenklasse“ eigene Methoden oder angpasste setter und getter zufügen.
