Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Dr. Daniel Büscher, Hannes Saffrich
Wintersemester 2019

Universität Freiburg
Institut für Informatik

Übungsblatt 12 – Lösungen

Abgabe: Montag, 27.01.2020, 9:00 Uhr morgens

Aufgabe 12.1 (Vererbung; Datei: `fleet.py`; Punkte: 8 = 3+3+2)

Für einen Fuhrpark bestehend aus PKWs, LKWs, Bussen und Fahrrädern soll eine Klassenhierarchie entworfen werden. Verwenden Sie die folgenden Klassen:

```
Fahrzeug
Kraftfahrzeug
Bus
Fahrrad
PKW
LKW
```

Dabei sollen die unterschiedlichen Fahrzeuge sowohl gemeinsame als auch unterschiedliche Attribute besitzen, jedes soll durch ein `int` beschrieben werden:

- Jedes Fahrzeug besitzt ein Leergewicht (Kilogramm).
- Jedes Kraftfahrzeug besitzt eine Leistung (Kilowatt).
- Zu jedem Bus gehören die Angaben: Baujahr, sowie Anzahl der Sitz- und Stehplätze.
- Zu jedem Fahrrad gehören die Angaben: Baujahr und Rahmengröße (cm).
- Zu jedem PKW gehören die Angaben: Baujahr, sowie Anzahl der Sitzplätze.
- Zu jedem LKW gehören die Angaben: Baujahr, Anzahl Sitzplätze, sowie Zuladung (Kilogramm).

Vermeiden Sie - soweit wie möglich - Wiederholungen in den folgenden Aufgabenteilen:

(a) Implementieren Sie die Klassenhierarchie wie oben angegeben. Machen Sie dabei Gebrauch von Vererbung. Halten sie dabei folgende Reihenfolge bei den Argumenten zur Instanzenerstellung ein (soweit zutreffend): Leergewicht, Baujahr, Leistung, Sitzplätze, Stehplätze, Rahmengröße, Zuladung

(b) Implementieren Sie die Methode zur Umwandlung von Objekten jeder dieser Klassen in Strings, die den jeweiligen Typ und zusätzlich die spezifischen Daten des Objektes enthält. Halten Sie dabei ebenso die obige Reihenfolge ein und geben Sie die entsprechenden Einheiten an. Die Ausgabe soll exakt folgenden Beispielen entsprechen:

```
>>> rad = Fahrrad(10, 2019, 55)
>>> print(rad)
Fahrrad Leergewicht: 10kg Baujahr: 2019 Rahmengroesse: 55cm
>>> pkw = PKW(1200, 2016, 90, 5)
>>> print(pkw)
PKW Leergewicht: 1200kg Baujahr: 2016 Leistung: 90kW Sitzplätze: 5
```

(c) Implementieren die Funktion `maut(fzg: Fahrzeug) -> int`, welche den Preis zum Überqueren einer Brücke bestimmt. Dabei soll der Preis (in Cent) aus dem Gewicht des Fahrzeugs geteilt durch 10 plus die Anzahl der möglichen Passagiere mal 20 errechnet werden. Fahrräder dürfen umsonst die Brücke überqueren. Beispiel:

```
>>> print(maut(pkw))
220
```

Lösung:

```
class Fahrzeug:
    def __init__(self,
                 leergewicht: int,
                 baujahr: int):
        self.leergewicht = leergewicht
        self.baujahr = baujahr
    def data_str(self):
        return \
            "Leergewicht: " + repr(self.leergewicht) + "kg" + \
            " Baujahr: " + repr(self.baujahr)
    def __str__(self):
        return "Fahrzeug " + self.data_str()


class Kraftfahrzeug(Fahrzeug):
    def __init__(self,
                 leergewicht: int,
                 baujahr: int,
                 leistung: int,
                 sitzplaetze: int):
        super().__init__(leergewicht, baujahr)
        self.leistung = leistung
        self.sitzplaetze = sitzplaetze
    def data_str(self):
        return super().data_str() + \
            " Leistung: " + repr(self.leistung) + "kW" + \
            " Sitzplätze: " + repr(self.sitzplaetze)
    def __str__(self):
        return "Kraftfahrzeug " + self.data_str()
```

```
class Bus(Kraftfahrzeug):
    def __init__(self,
                 leergewicht: int,
                 leistung: int,
                 baujahr: int,
                 sitzplaetze: int,
                 stehplaetze: int):
        super().__init__(leergewicht, leistung, baujahr, sitzplaetze)
        self.stehplaetze = stehplaetze
    def data_str(self):
        return super().data_str() + \
            " Stehplätze: " + repr(self.stehplaetze)
    def __str__(self):
        return "Bus " + self.data_str()

class Fahrrad(Fahrzeug):
    def __init__(self,
                 leergewicht: int,
                 baujahr: int,
                 rahmengroesse: int):
        super().__init__(leergewicht, baujahr)
        self.rahmengroesse = rahmengroesse
    def data_str(self):
        return super().data_str() + \
            " Rahmengroesse: " + repr(self.rahmengroesse) + "cm"
    def __str__(self):
        return "Fahrrad " + self.data_str()

class PKW(Kraftfahrzeug):
    def __init__(self,
                 leergewicht: int,
                 leistung: int,
                 baujahr: int,
                 sitzplaetze: int):
        super().__init__(leergewicht, leistung, baujahr, sitzplaetze)
    def data_str(self):
        return super().data_str()
    def __str__(self):
        return "PKW " + self.data_str()

class LKW(Kraftfahrzeug):
    def __init__(self,
                 leergewicht: int,
                 leistung: int,
```

```
                 baujahr: int,
                 sitzplaetze: int,
                 zuladung: int):
        super().__init__(leergewicht, leistung, baujahr, sitzplaetze)
        self.zuladung = zuladung
    def data_str(self):
        return super().data_str() + \
            " Zuladung: " + repr(self.zuladung) + "kg"
    def __str__(self):
        return "LKW " + self.data_str()

def maut(fzg: Fahrzeug) -> int:
    if isinstance(fzg, Fahrrad):
        return 0
    gewicht = fzg.leergewicht
    plaetze = 0
    if isinstance(fzg, LKW):
        gewicht += fzg.zuladung
    if isinstance(fzg, Kraftfahrzeug):
        plaetze = fzg.sitzplaetze
    if isinstance(fzg, Bus):
        plaetze += fzg.stehplaetze
    return gewicht // 10 + plaetze * 20
```

Aufgabe 12.2 (Objekte; Datei: `fraction.py`; Punkte: 10 = 2+1+3+2+2)

Entwerfen und implementieren Sie eine Klasse `Fraction` zum exakten Rechnen mit Brüchen auf der Basis der vorhandenen arithmetischen Operationen auf `int`. Die interne Repräsentation soll dabei verkapselt sein. Instanzen von `Fraction` sollen nicht veränderlich sein. **Die Implementierung soll komplett ohne Gleitkommazahlen auskommen!**

Implementieren Sie folgende Methoden:

(a) Der Konstruktor soll sicherstellen, dass der Nenner niemals 0 ist (Invariante).

(b) Getter für Zähler und Nenner als Properties: `numerator` und `denominator`.

(c) Implementieren Sie die Operatoren `+`, `-` und `*` auf Brüchen.

(d) Erweitern Sie die Implementierung von `*`, sodass sie auch funktioniert, wenn das zweite Argument vom Typ `int` ist.

(e) Implementieren Sie Methoden für die Vergleichsoperatoren `==` und `<` auf Brüchen.

Sie dürfen ausschließlich folgenden Import nutzen: `from fractions import gcd`. Einige Teilaufgaben erfordern die Definition von geeigneten magischen Methoden. Eine Zusammenstellung dieser Methoden für Operatoren finden Sie hier: www.python-

course.eu/python3_magic_methods.php

Lösung:

```
from fractions import gcd

class Fraction:
    def __init__ (self, numerator, denominator=1):
        assert denominator != 0, "denominator must not be 0"
        if denominator < 0:
            (numerator, denominator) = (-numerator, -denominator)
        gcden = gcd(numerator, denominator)
        self.__num = numerator // gcden
        self.__den = denominator // gcden

    @property
    def numerator(self):
        return self.__num
    @property
    def denominator(self):
        return self.__den

    def __add__ (self, other):
        new_num = self.numerator * other.denominator + other.numerator * self.denominato
        new_den = self.denominator * other.denominator
        return Fraction (new_num, new_den)

    def __sub__ (self, other):
        return self + other * (-1)

    def __mul__ (self, other):
        if isinstance (other, Fraction):
            new_num = self.numerator * other.numerator
            new_den = self.denominator * other.denominator
            return Fraction (new_num, new_den)
        else:
            new_num = self.numerator * other
            new_den = self.denominator
            return Fraction (new_num, new_den)

    def __eq__ (self, other):
        return self.numerator == other.numerator and \
            self.denominator == other.denominator

    def __lt__ (self, other):
        return (self - other).numerator < 0
```

Aufgabe 12.3 (Erfahrungen; Datei: `erfahrungen.txt`; Punkte: 2)

Legen Sie im Unterverzeichnis `sheet12` eine Textdatei `erfahrungen.txt` an. Notieren Sie in dieser Datei kurz Ihre Erfahrungen beim Bearbeiten der Übungsaufgaben (Probleme, Bezug zur Vorlesung, Interessantes, benötigter Zeitaufwand, etc.).