МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Информатика»
Тема: Парадигмы программирования

Студент гр. 3342 _____ Малахов А.И.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2024

Цель работы

Изучение парадигм программирования. Написать программу с использованием концепции ООП.

Задание

Вариант 3.

Базовый класс — транспорт Transport:

Поля объекта класс Transport:

- средняя скорость (в км/ч, положительное целое число)
- максимальная скорость (в км/ч, положительное целое число)
- цена (в руб., положительное целое число)
- грузовой (значениями могут быть или True, или False)
- цвет (значение может быть одной из строк: w (white), g(gray), b(blue)).

При создании экземпляра класса Transport необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'.

Класс автомобиль – Car наследуется от класса *Transport*.

Поля объекта класс *Car*:

- средняя скорость (в км/ч, положительное целое число)
- максимальная скорость (в км/ч, положительное целое число)
- цена (в руб., положительное целое число)
- грузовой (значениями могут быть или True, или False)
- цвет (значение может быть одной из строк: w (white), g(gray), b(blue)).
- мощность (в Вт, положительное целое число)
- количество колес (положительное целое число, не более 10)

При создании экземпляра класса Car необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'.

В данном классе необходимо реализовать следующие методы:

- Метод *__str__()*: Преобразование к строке вида: Car: средняя скорость <средняя скорость>, максимальная скорость <максимальная скорость>, цена <цена>, грузовой <грузовой>, цвет <цвет>, мощность <мощность>, количество колес <количество колес>.

- Метод *__add__()*: Сложение средней скорости и максимальной скорости автомобиля. Возвращает число, полученное при сложении средней и максимальной скорости.
- Метод *__eq__()*: Сложение средней скорости и максимальной скорости автомобиля. Возвращает число, полученное при сложении средней и максимальной скорости.

Класс самолет - Plane наследуется от класса *Transport*.

Поля объекта класс *Plane*:

- средняя скорость (в км/ч, положительное целое число)
- максимальная скорость (в км/ч, положительное целое число)
- цена (в руб., положительное целое число)
- грузовой (значениями могут быть или True, или False)
- цвет (значение может быть одной из строк: w (white), g(gray), b(blue)).
- грузоподъемность (в кг, положительное целое число)
- размах крыльев (в м, положительное целое число)

При создании экземпляра класса Plane необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'.

В данном классе необходимо реализовать следующие методы:

- Метод *__str__()*: Преобразование к строке вида: Plane: средняя скорость <средняя скорость>, максимальная скорость <максимальная скорость>, цена <цена>, грузовой <грузовой>, цвет <цвет>, грузоподъемность <грузоподъемность>, размах крыльев <размах крыльев>.
- Метод *__add__()*: Сложение средней скорости и максимальной скорости самолета. Возвращает число, полученное при сложении средней и максимальной скорости.
- Метод *__eq__():* Метод возвращает True, если два объекта класса равны по размерам, и False иначе. Два объекта типа Plane равны по размерам, если равны размах крыльев.

Класс самолет – Ship наследуется от класса *Transport*.

Поля объекта класс *Ship*:

- средняя скорость (в км/ч, положительное целое число)
- максимальная скорость (в км/ч, положительное целое число)
- цена (в руб., положительное целое число)
- грузовой (значениями могут быть или True, или False)
- цвет (значение может быть одной из строк: w (white), g(gray), b(blue)).
- высота борта (в м, положительное целое число)

При создании экземпляра класса Ship необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'.

В данном классе необходимо реализовать следующие методы:

- Метод *__str__()*: Преобразование к строке вида: Ship: средняя скорость <средняя скорость>, максимальная скорость <максимальная скорость>, цена <цена>, грузовой <грузовой>, цвет <цвет>, длина <длина>, высота борта <высота борта>.
- Метод *__add__()*: Сложение средней скорости и максимальной скорости корабля. Возвращает число, полученное при сложении средней и максимальной скорости.
- Метод *__eq__():* Метод возвращает True, если два объекта класса равны по размерам, и False иначе. Два объекта типа Ship равны по размерам, если равны их длина и высота борта.

Необходимо определить список *list* для работы с транспортом:

Автомобили:

class CarList – список автомобилей - наследуется от класса *list*.

Конструктор:

Вызвать конструктор базового класса.

Передать в конструктор строку *name* и присвоить её полю *name* созданного объекта

Необходимо реализовать следующие методы:

- Метод *append(p_object)*: Переопределение метода append() списка. В случае, если p_object - автомобиль, элемент добавляется в список, иначе выбрасывается исключение TypeError с текстом: Invalid type <тип_объекта p_object> (результат вызова функции type)
- Метод *print_colors()*: Вывести цвета всех автомобилей в виде строки (нумерация начинается с 1):Метод *print_count()*: Вывести количество книг.

Самолеты:

class PlaneList – список самолетов - наследуется от класса *list*.

Конструктор:

Вызвать конструктор базового класса.

Передать в конструктор строку *name* и присвоить её полю *name* созданного объекта.

Необходимо реализовать следующие методы:

- Метод *extend(iterable)*: Переопределение метода extend() списка. В случае, если элемент iterable - объект класса Plane, этот элемент добавляется в список, иначе не добавляется.
- Метод *print_colors()*: Вывести цвета всех самолетов в виде строки (нумерация начинается с 1)
- Метод *total_speed()*: Посчитать и вывести общую среднюю скорость всех самолетов.

Корабли:

class ShipList – список самолетов - наследуется от класса *list*.

Конструктор:

Вызвать конструктор базового класса.

Передать в конструктор строку *name* и присвоить её полю *name* созданного объекта.

Необходимо реализовать следующие методы:

- Метод *append(iterable)*: Переопределение метода append() списка. В случае, если p_object - корабль, элемент добавляется в список, иначе

выбрасывается исключение TypeError с текстом: Invalid type <тип_объекта p_object>

- Метод *print_colors()*: Вывести цвета всех кораблей в виде строки (нумерация начинается с 1)
- Метод *print_ship()*: Вывести те корабли, чья длина больше 150 метров, в виде строки

Выполнение работы

Класс *Transport*. Конструктор принимает *average_speed*, *max_speed*, *price*, *cargo, color* в качестве параметров, параметры присваиваются полям класса. Производится проверка на тип, у всех параметров, а также на значение: *average_speed* - положительное число, *max_speed* – положительное число, *cargo* – тип bool, цвет – строка “w”, “g” или “b”. В случае, если параметр не соответствует предъявленным требованиям, вызывается исключение ValueError с сообщением: “Invalid value”.

Класс *Car* наследуется от класса *Transport*. Конструктор принимает *average_speed*, *max_speed*, *price*, *cargo, color, power, wheels* в качестве параметров, параметры присваиваются полям класса. Поля *name, average_speed*, *max_speed*, *price*, *cargo, color* передаются конструктору родительского класса. Проводится проверка на соответствие типам и значениям оставшихся параметров: *power* – положительное целое число, *wheels* – положительное число не больше десяти. В случае несоответствия предъявленным требованиям вызывается исключение ValueError с сообщением: “Invalid value”. Переопределяется метод *__str__* для приведению класса к типу string, например при помещении экземпляра класса в функцию print(). Переопределяется метод *__eq__*, в котором сравниваются количество колес, средняя и максимальная скорости, мощность. Переопределяется метод __add__, который возвращает сумму средней и максимальной скоростей.

Класс *Plane* наследуется от класса *Transport*. Конструктор принимает *average_speed*, *max_speed*, *price*, *cargo, color, load_capacity, wingspan* в качестве параметров, параметры присваиваются полям класса. Поля *name, average_speed*, *max_speed*, *price*, *cargo, color* передаются конструктору родительского класса. Проводится проверка на соответствие типам и значениям оставшихся параметров, они должны быть целыми положительными числами. В случае несоответствия предъявленным требованиям вызывается исключение ValueError с сообщением: “Invalid value”. Переопределяется метод *__str__* для приведению класса к типу string, например при помещении экземпляра класса в функцию

print(). Переопределяется метод __*eq*__, в котором сравнивается размах крыла. Переопределяется метод __add__, который возвращает сумму средней и максимальной скоростей.

Класс *Ship* наследуется от класса *Transport*. Конструктор принимает *average_speed*, *max_speed*, *price*, *cargo, color, length, side_height* в качестве параметров, параметры присваиваются полям класса. Поля *name, average_speed*, *max_speed*, *price*, *cargo, color* передаются конструктору родительского класса. Проводится проверка на соответствие типам и значениям оставшихся параметров, они должны быть целыми положительными числами. В случае несоответствия предъявленным требованиям вызывается исключение ValueError с сообщением: “Invalid value”. Переопределяется метод __*str*__ для приведению класса к типу string, например при помещении экземпляра класса в функцию print(). Переопределяется метод __*eq*__, в котором сравниваются длина и высота борта. Переопределяется метод __add__, который возвращает сумму средней и максимальной скоростей.

Класс *CarList* наследуется от класса *list*. В конструктор передается имя списка, в нем вызывается родительский конструктор, а затем присваивается параметр *name*. Переопределяется метод *append*, в котором проверяется тип добавляемого объекта, в случае несоответствия, вызывается TypeError, иначе вызывается *append* у родительского метода. Метод *print_colors* печатает цвет каждого автомобиля. Метод *print_count* печатает количество автомобилей в списке.

Класс *PlaneList* наследуется от класса *list*. В конструктор передается имя списка, в нем вызывается родительский конструктор, а затем присваивается параметр *name*. Переопределяется метод *extend*, в цикле проверяется все ли элементы *iterable* корректного типа, в случае несоответствия метод завершается, иначе вызывается родительский *extend*. Метод *print_colors* печатает цвет каждого автомобиля. Метод *total_speed* печатает суммарную среднюю скорость самолетов из списка.

Класс *ShipList* наследуется от класса *list*. В конструктор передается имя списка, в нем вызывается родительский конструктор, а затем присваивается параметр *name*. Переопределяется метод *append*, в котором проверяется тип добавляемого объекта, в случае несоответствия, вызывается TypeError, иначе вызывается *append* у родительского метода. Метод *print_colors* печатает цвет каждого корабля. Метод *print_ship* печатает номера тех кораблей, у которых длина больше 150 м.

Разработанный программный код см. в приложении А.

Выводы

Были изучены парадигмы программирования. Написана программа с использованием концепции ООП.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Transport:
    def __init__(self, average_speed, max_speed, price, cargo, color):
        self.validate_parameters(average_speed,  max_speed,  price,
cargo, color)
        self.average_speed = average_speed
        self.max_speed = max_speed
        self.price = price
        self.cargo = cargo
        self.color = color

    @staticmethod
    def validate_parameters(average_speed, max_speed, price, cargo,
color):
        if not (isinstance(average_speed, int) and average_speed > 0
and
                isinstance(max_speed, int) and max_speed > 0 and
                isinstance(price, int) and price > 0 and
                isinstance(cargo, bool) and
                color in ['w', 'g', 'b']):
            raise ValueError('Invalid value')


class Car(Transport):
    def __init__(self, average_speed, max_speed, price, cargo, color,
power, wheels):
        super().__init__(average_speed,  max_speed,  price,  cargo,
color)
        self.validate_car_parameters(power, wheels)
        self.power = power
        self.wheels = wheels

    @staticmethod
    def validate_car_parameters(power, wheels):
```

```
            if not (isinstance(power, int) and power > 0 and
                    isinstance(wheels, int) and 0 < wheels <= 10):
                raise ValueError('Invalid value')

        def __str__(self):
            return f"Car: средняя скорость {self.average_speed}, максимальная скорость {self.max_speed}, цена {self.price}, грузовой {self.cargo}, цвет {self.color}, мощность {self.power}, количество колес {self.wheels}."

        def __add__(self):
            return self.max_speed + self.average_speed

        def __eq__(self, other):
            return (
                    (self.wheels == other.wheels)
                    and (self.average_speed == other.average_speed)
                    and (self.max_speed == other.max_speed)
                    and (self.power == other.power)
            )


    class Plane(Transport):
        def __init__(self, average_speed, max_speed, price, cargo, color, load_capacity, wingspan):
            super().__init__(average_speed, max_speed, price, cargo, color)
            self.validate_plane_parameters(load_capacity, wingspan)
            self.load_capacity = load_capacity
            self.wingspan = wingspan

        @staticmethod
        def validate_plane_parameters(load_capacity, wingspan):
            if not (isinstance(load_capacity, int) and load_capacity > 0 and
                    isinstance(wingspan, int) and wingspan > 0):
                raise ValueError('Invalid value')
```

```
        def __str__(self):
            return f"Plane: средняя скорость {self.average_speed}, максимальная скорость {self.max_speed}, цена {self.price}, грузовой {self.cargo}, цвет {self.color}, грузоподъемность {self.load_capacity}, размах крыльев {self.wingspan}."

        def __add__(self):
            return self.max_speed + self.average_speed

        def __eq__(self, other):
            return self.wingspan == other.wingspan


    class Ship(Transport):
        def __init__(self, average_speed, max_speed, price, cargo, color, length, side_height):
            super().__init__(average_speed, max_speed, price, cargo, color)
            self.validate_ship_parameters(length, side_height)
            self.length = length
            self.side_height = side_height

        @staticmethod
        def validate_ship_parameters(length, side_height):
            if not (isinstance(length, int) and length > 0 and
                    isinstance(side_height, int) and side_height > 0):
                raise ValueError('Invalid value')

        def __str__(self):
            return f"Ship: средняя скорость {self.average_speed}, максимальная скорость {self.max_speed}, цена {self.price}, грузовой {self.cargo}, цвет {self.color}, длина {self.length}, высота борта {self.side_height}."

        def __add__(self):
            return self.max_speed + self.average_speed
```

```
        def __eq__(self, other):
            return self.length == other.length and self.side_height ==
other.side_height


    class CarList(list):
        def __init__(self, name):
            super().__init__()
            self.name = name

        def append(self, p_object):
            if isinstance(p_object, Car):
                super().append(p_object)
            else:
                raise TypeError(f"Invalid type <т и п _ о б ъ е к т а
p_object> {type(p_object)}")

        def print_colors(self):
            for i, car in enumerate(self):
                print(f"{i + 1} а в т о м о б и л ь : {car.color}")

        def print_count(self):
            print(len(self))


    class PlaneList(list):
        def __init__(self, name):
            super().__init__()
            self.name = name

        def extend(self, iterable):
            planes_only = filter(lambda x: isinstance(x, Plane), iterable)
            super().extend(planes_only)

        def print_colors(self):
            for i, plane in enumerate(self):
                print(f"{i + 1} с а м о л е т : {plane.color}")
```

```
        def total_speed(self):
            total_speed = sum(plane.average_speed for plane in self)
            print(total_speed)


    class ShipList(list):
        def __init__(self, name):
            super().__init__()
            self.name = name

        def append(self, p_object):
            if isinstance(p_object, Ship):
                super().append(p_object)
            else:
                raise TypeError(f"Invalid type <тип_объекта p_object> {type(p_object)}")

        def print_colors(self):
            for i, ship in enumerate(self):
                print(f"{i + 1} корабль: {ship.color}")

        def print_ship(self):
            for i, ship in enumerate(self):
                if ship.length > 150:
                    print(f'Длина корабля №{i + 1} больше 150 метров')
```