

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

Русанов 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 соответствие типа у всех параметров, а также на их значение. 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. 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. 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. 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: int, max_speed: int, price: int,
cargo: bool, color: str
    ):
        if average_speed > 0 and isinstance(average_speed, int):
            self.average_speed = average_speed
        else:
            raise ValueError("Invalid value")

        if max_speed > 0 and isinstance(max_speed, int):
            self.max_speed = max_speed
        else:
            raise ValueError("Invalid value")

        if price > 0 and isinstance(price, int):
            self.price = price
        else:
            raise ValueError("Invalid value")

        if isinstance(cargo, bool):
            self.cargo = cargo
        else:
            raise ValueError("Invalid value")

        if color in ["w", "g", "b"]:
            self.color = color
        else:
            raise ValueError("Invalid value")

class Car(Transport): # Н а с л е д у е т с я о т к л а с с а Transport
    def __init__(
```

```

        self,
        average_speed: int,
        max_speed: int,
        price: int,
        cargo: bool,
        color: str,
        power: int,
        wheels: int,
    ):
        super().__init__(average_speed, max_speed, price, cargo,
color)

        if power > 0 and isinstance(power, int):
            self.power = power
        else:
            raise ValueError("Invalid value")

        if 0 < wheels <= 10 and isinstance(wheels, int):
            self.wheels = wheels
        else:
            raise ValueError("Invalid value")

    def __str__(self):
        return f"Car:  с р е д н я я      с к о р о с т ь
{self.average_speed},  м а к с и м а л ь н а я  с к о р о с т ь {self.max_speed},
ц е н а {self.price},  г р у з о в о й {self.cargo},  ц в е т {self.color},  м
о щ н о с т ь {self.power},  к о л и ч е с т в о  к о л е с {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): # Н а с л е д у е т с я   о т   к л а с с а
Transport
    def __init__(
        self,
        average_speed: int,
        max_speed: int,
        price: int,
        cargo: bool,
        color: str,
        load_capacity: int,
        wingspan: int,
    ):
        super().__init__(average_speed, max_speed, price, cargo,
color)

        if load_capacity > 0 and isinstance(load_capacity, int):
            self.load_capacity = load_capacity
        else:
            raise ValueError("Invalid value")

        if wingspan > 0 and isinstance(wingspan, int):
            self.wingspan = wingspan
        else:
            raise ValueError("Invalid value")

    def __str__(self):
        return f'Plane:   с р е д н я я   с к о р о с т ь
{self.average_speed}, м а к с и м а л ь н а я   с к о р о с т ь {self.max_speed},
ц е н а {self.price}, г р у з о в о й {self.cargo}, ц в е т {self.color}, г
р у з о п о д ь е м н о с т ь {self.load_capacity}, р а з м а х   к р ы л ь е в
{self.wingspan}.'

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

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

```

```

class Ship(Transport):    # Н а с л е д у е т с я   о т   к л а с с а
Transport
    def __init__(
        self,
        average_speed: int,
        max_speed: int,
        price: int,
        cargo: bool,
        color: str,
        length: int,
        side_height: int,
    ):
        super().__init__(average_speed, max_speed, price, cargo,
color)

        if length > 0 and isinstance(length, int):
            self.length = length
        else:
            raise ValueError("Invalid value")

        if side_height > 0 and isinstance(side_height, int):
            self.side_height = side_height
        else:
            raise ValueError("Invalid value")

    def __str__(self):
        return f'Ship:   с р е д н я я   с к о р о с т ь
{self.average_speed},   м а к с и м а л ь н а я   с к о р о с т ь {self.max_speed},
ц е н а {self.price},   г р у з о в о й {self.cargo},   ц в е т {self.color},   д
л и н а {self.length},   в ы с о т а   б о р т а {self.side_height}.'

    def __add__(self):
        return self.average_speed + self.max_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 <тип_объекта  

p_object> {type(p_object)}')

    def print_colors(self):
        print('\n'.join([f'{i+ 1} а в т о м о б и л ь :  

{self[i].color}' for i in range(len(self))]))

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

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

    def extend(self, iterable):
        for i in range(len(iterable)):
            if isinstance(iterable[i], Plane):
                super().append(iterable[i])

    def print_colors(self):
        print('\n'.join([f'{i+1} с а м о л е т : {self[i].color}' for  

i in range(len(self))]))

```



```

def total_speed(self):
    summ = 0
    for i in self:
        summ += i.average_speed
    print(summ)

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):
        print('\n'.join([f'{i + 1} корабль: {self[i].color}' for  
i in range(len(self))]))

    def print_ship(self):
        for i in range(len(self)):
            if self[i].length > 150:
                print(f'Длина корабля №{i+1} больше 150  
метров')

```