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 _____ Иванов С.С.

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

Сначала напишем дополнительный data-descriptor (класс DataDescriptor) для удобства проверки условий корректности полей основных классов. В качестве аргумента в инициализаторе будет приниматься функция-валидатор.

Класс *Transport*. Конструктор принимает *average_speed*, *max_speed*, *price*, *cargo, color* в качестве параметров, средняя скорость, максимальная скорость, цена, грузовой, цвет соответственно, параметры присваиваются полям класса. Производится проверка на тип, у всех параметров, а также на значение: средняя скорость - положительное число, максимальная скорость – положительное число, грузовой – тип bool, цвет – строка: w или g или b. В случае несоответствия 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, средняя скорость, максимальная скорость, цена, грузовой, цвет, мощность, количество колес соответственно, параметры присваиваются полям класса. Поля *name, average_speed*, *max_speed*, *price*, *cargo, color* передаются конструктору родительского класса. Проводится проверка на соответствие типам оставшихся параметров, на положительность числа мощности, на количество колес. 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 кол-во колес, средняя и максимальная скорости, мощность, это необходимо для понимания являются ли данные экземпляры одними объектами по смыслу, 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, средняя скорость, максимальная скорость, цена, грузовой, цвет, грузоподъемность, размах крыла соответственно, параметры присваиваются

полям класса. Поля *name, average_speed*, *max_speed*, *price*, *cargo, color* передаются конструктору родительского класса. Проводится проверка на соответствие типам оставшихся параметров, на положительность числа грузоподъемности, на положительность числа размаха крыла. 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 размах крыла, это необходимо для понимания являются ли данные экземпляры одними объектами по смыслу, 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, средняя скорость, максимальная скорость, цена, грузовой, цвет, длина, высотаборта соответственно, параметры присваиваются полям класса. Поля *name, average_speed*, *max_speed*, *price*, *cargo, color* передаются конструктору родительского класса. Проводится проверка на соответствие типам оставшихся параметров, на положительность числа грузоподъемности, на положительность числа размаха крыла. 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, это необходимо для понимания являются ли данные экземпляры одними объектами по смыслу, 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

```
from typing import Callable, Any


class DataDescriptor:
    def __init__(self, is_correct_val: Callable) -> None:
        self._is_correct_val = is_correct_val

    def __set_name__(self, owner, name) -> None:
        self._name = f"_{owner.__name__}__{name}"

    def __get__(self, instance, _) -> Any:
        return getattr(instance, self._name)

    def __set__(self, instance, value) -> None:
        if not self._is_correct_val(value):
            raise ValueError("Invalid value")

        setattr(instance, self._name, value)


class Transport:
    '''Поля объекта класс Transport:

    average_speed - средняя скорость (в км/ч, положительное целое число)

    max_speed - максимальная скорость (в км/ч, положительное  целое числ
о)

    price - цена (в руб., положительное целое число)

    cargo - грузовой (значениями могут быть или True, или False)

    color - цвет (значение может быть одной из строк: W (white), G(gray),
B(blue)).
```

При создании экземпляра класса `Transport` необходимо убедиться, что пе
реданные в конструктор параметры удовлетворяют требованиям, иначе выбросить иск
лючение `ValueError` с текстом `'Invalid value'`.

```
    '''

    average_speed = DataDescriptor(
        lambda val: isinstance(val, int) and val > 0
    )

    max_speed = DataDescriptor(
        lambda val: isinstance(val, int) and val > 0
    )

    price = DataDescriptor(
        lambda val: isinstance(val, int) and val > 0
    )

    cargo = DataDescriptor(
        lambda val: isinstance(val, bool)
    )

    color = DataDescriptor(
        lambda val: val == 'w' or \
                    val == 'g' or \
                    val == 'b'
    )

    def __init__(
            self,
            average_speed: int,
            max_speed: int,
            price: int,
            cargo: bool,
            color: str
        ) -> None:

        self.average_speed = average_speed
```

```
        self.max_speed = max_speed
        self.price = price
        self.cargo = cargo
        self.color = color


class Car(Transport): #Наследуется от класса Transport
    '''Поля объекта класс Car:
    average_speed – средняя скорость (в км/ч, положительное целое число)
    max_speed – максимальная скорость (в км/ч, положительное целое числ
о)
    price – цена (в руб., положительное целое число)
    cargo – грузовой (значениями могут быть или True, или False)
    color – цвет (значение может быть одной из строк: W (white), G(gray),
B(blue)).
    power – мощность (в Вт, положительное целое число)
    wheels – количество колес (положительное целое число, не более 10)
    При создании экземпляра класса Car необходимо убедиться, что переданн
ые в конструктор параметры удовлетворяют требованиям, иначе выбросить исключени
е ValueError с текстом 'Invalid value'.
    '''

    power = DataDescriptor(
        lambda val: isinstance(val, int) and val > 0
    )

    wheels = DataDescriptor(
        lambda val: isinstance(val, int) and val > 0 and val <= 10
    )

    def __str__(self) -> str:
```

```
            '''Преобразование к строке вида: Car: средняя скорость <средняя с
корость>, максимальная скорость <максимальная скорость>, цена <цена>, грузовой <
грузовой>, цвет <цвет>, мощность <мощность>, количество колес <количество коле
с>.'''

            return f"Car: средняя скорость {self.average_speed}, максималь
ная скорость {self.max_speed}, цена {self.price}, грузовой {self.cargo}, цве
т {self.color}, мощность {self.power}, количество колес {self.wheels}."

        def __add__(self) -> int:
            '''Сложение средней скорости и максимальной скорости автомобиля.
Возвращает число, полученное при сложении средней и максимальной скорости.'''

            return self.average_speed + self.max_speed

        def __eq__(self, other) -> bool:
            '''Метод возвращает True, если два объекта класса равны, и False
иначе. Два объекта типа Car равны, если равны количество колес, средняя скорост
ь, максимальная скорость и мощность.'''

            return self.wheels == other.wheels and \
                   self.average_speed == other.average_speed and \
                   self.max_speed == other.max_speed and \
                   self.power == other.power

        def __init__(
            self,
            average_speed: int,
            max_speed: int,
            price: int,
            cargo: bool,
            color: str,
```

```
                power: int,
                wheels: int
            ) -> None:

            super().__init__(average_speed, max_speed, price, cargo,
color)

            self.power = power
            self.wheels = wheels


    class Plane(Transport): #Наследуется от класса Transport
        '''Поля объекта класс Plane:
        average_speed - средняя скорость (в км/ч, положительное целое число)
        max_speed - максимальная скорость (в км/ч, положительное целое числ
о)
        price - цена (в руб., положительное целое число)
        cargo - грузовой (значениями могут быть или True, или False)
        color - цвет (значение может быть одной из строк: W (white), G(gray),
B(blue)).
        load_capacity - грузоподъемность (в кг, положительное целое число)
        wingspan - размах крыльев (в м, положительное целое число)
        При создании экземпляра класса Plane необходимо убедиться, что переда
нные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключе
ние ValueError с текстом 'Invalid value'.
        '''

        load_capacity = DataDescriptor(
            lambda val: isinstance(val, int) and val > 0
        )

        wingspan = DataDescriptor(
```

```
            lambda val: isinstance(val, int) and val > 0
        )

        def __str__(self) -> 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>.'''

            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) -> int:
            '''Сложение средней скорости и максимальной скорости самолета. Возвращает число, полученное при сложении средней и максимальной скорости.'''

            return self.average_speed + self.max_speed

        def __eq__(self, other) -> bool:
            '''Метод возвращает True, если два объекта класса равны по размерам, и False иначе. Два объекта типа Plane равны по размерам, если равны размах крыльев.'''

            return self.wingspan == other.wingspan

        def __init__(
                self,
                average_speed: int,
                max_speed: int,
                price: int,
```

```
                cargo: bool,
                color: str,
                load_capacity: int,
                wingspan: int
            ) -> None:

            super().__init__(average_speed, max_speed, price, cargo,
color)

            self.load_capacity = load_capacity
            self.wingspan = wingspan


    class Ship(Transport): #Наследуется от класса Transport
        '''Поля объекта класс Ship:
        average_speed – средняя скорость (в км/ч, положительное целое число)
        max_speed – максимальная скорость (в км/ч, положительное целое числ
о)
        price – цена (в руб., положительное целое число)
        cargo – грузовой (значениями могут быть или True, или False)
        color – цвет (значение может быть одной из строк: W (white), G(gray),
B(blue)).
        length – длина (в м, положительное целое число)
        side_height – высота борта (в м, положительное целое число)
        При создании экземпляра класса Ship необходимо убедиться, что передан
ные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключен
ие ValueError с текстом 'Invalid value'.
        '''

        length = DataDescriptor(
            lambda val: isinstance(val, int) and val > 0
        )
```

```
    side_height = DataDescriptor(
        lambda val: isinstance(val, int) and val > 0
    )

    def __str__(self) -> 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>.'''

        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) -> int:
        '''Сложение средней скорости и максимальной скорости корабля. Возвращает число, полученное при сложении средней и максимальной скорости.'''

        return self.average_speed + self.max_speed

    def __eq__(self, other) -> bool:
        '''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.'''

        return self.length == other.length and \
               self.side_height == other.side_height

    def __init__(
            self,
            average_speed: int,
            max_speed: int,
            price: int,
```

```
                cargo: bool,
                color: str,
                length: int,
                side_height: int
            ) -> None:

            super().__init__(average_speed, max_speed, price, cargo,
color)

            self.length = length
            self.side_height = side_height


    class CarList(list): # – список автомобилей – наследуется от класса list.
        def __init__(self, name: str) -> None:
            '''1. Вызвать конструктор базового класса.
               2. Передать в конструктор строку name и присвоить её полю name
созданного объекта'''

            super().__init__()

            self.name = name

        def append(self, p_object: Car) -> None:
            '''Переопределение метода append() списка. В случае, если
p_object – автомобиль, элемент добавляется в список, иначе выбрасывается исключ
ение TypeError с текстом: Invalid type <тип_объекта p_object> (результат вызов
а функции type)'''

            if not isinstance(p_object, Car):
                raise TypeError(f"Invalid type {type(p_object)}")

            super().append(p_object)
```

```
        def print_colors(self) -> None:
            '''Вывести цвета всех автомобилей в виде строки:
                <i> автомобиль: color[i]
                <j> автомобиль: color[j] ...'''

            for i, car in enumerate(self, 1):
                print(f"{i} автомобиль: {car.color}")

        def print_count(self) -> int:
            '''Вывести количество автомобилей.'''

            print(len(self))


    class PlaneList(list): # – список самолетов – наследуется от класса list.
        def __init__(self, name: str) -> None:
            '''1. Вызвать конструктор базового класса.
               2. Передать в конструктор строку name и присвоить её полю name
созданного объекта'''

            super().__init__()

            self.name = name

        def extend(self, iterable) -> None:
            '''Переопределение метода extend() списка. В случае, если элемен
т iterable – объект класса Plane, этот элемент добавляется в список, иначе не до
бавляется.'''

            for el in iterable:
                if not isinstance(el, Plane):
                    continue
```

```
                self.append(el)

        def print_colors(self) -> None:
            '''Вывести цвета всех самолетов в виде строки:
                <i> самолет: color[i]
                <j> самолет: color[j] ...'''

            for i, plane in enumerate(self, 1):
                print(f"{i} самолет: {plane.color}")

        def total_speed(self) -> int:
            '''Посчитать и вывести общую среднюю скорость всех самолетов.'''

            print(sum(plane.average_speed for plane in self))


    class ShipList(list): # – список кораблей - наследуется от класса list.
        def __init__(self, name: str) -> None:
            '''1. Вызвать конструктор базового класса.
               2. Передать в конструктор строку name и присвоить её полю name созданного объекта'''

            super().__init__()

            self.name = name

        def append(self, p_object: Ship) -> None:
            '''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>'''
```

```
        if not isinstance(p_object, Ship):
            raise TypeError(f"Invalid type {type(p_object)}")

        super().append(p_object)

    def print_colors(self) -> None:
        '''Вывести цвета всех кораблей в виде строки:
            <i> корабль: color[i]
            <j> корабль: color[j] ...'''

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

    def print_ship(self) -> None:
        '''Вывести те корабли, чья длина больше 150 метров.'''

        for i, ship in enumerate(self, 1):
            if ship.length <= 150:
                continue

            print(f"Длина корабля №{i} больше 150 метров")
```