**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:

* cредняя 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*:

* cредняя 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*:

* cредняя 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*:

* cредняя 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 <тип\_объекта p\_object> {type(p\_object)}")

def print\_colors(self):

for i, car in enumerate(self):

print(f"{i + 1} автомобиль: {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} самолет: {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')