**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Информационные технологии»**

Тема: **Парадигмы программирования**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Хангулян С. К. |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2024

## Цель работы

Целью работы является изучение основных парадигм программирования, ознакомление с ООП и написание кода с несколькими классами и собственными методами.

## Задание

**Вариант 3**

Базовый класс - транспорт Transport:

class 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'.

Автомобиль - Car:

class 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\_\_():

Метод возвращает True, если два объекта класса равны, и False иначе. Два объекта типа Car равны, если равны количество колес, средняя скорость, максимальная скорость и мощность.

Самолет - Plane:

class 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.

Корабль - Ship:

class 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)).

длина (в м, положительное целое число)

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):

<i> автомобиль: <color[i]>

<j> автомобиль: <color[j]> ...

Метод print\_count(): Вывести количество автомобилей.

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):

<i> самолет: <color[i]>

<j> самолет: <color[j]> ...

Метод total\_speed(): Посчитать и вывести общую среднюю скорость всех самолетов.

Корабли:

class ShipList – список кораблей - 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 - 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):

<i> корабль: <color[i]>

<j> корабль: <color[j]> ...

Метод print\_ship(): Вывести те корабли, чья длина больше 150 метров, в виде строки:

Длина корабля №<i> больше 150 метров

Длина корабля №<j> больше 150 метров ...

## Выполнение работы

**Группа 1**

В группе 1 находятся классы Transport, Car, Plane, Ship. Первый класс – родительский, остальные – дочерние.

В классе Transport определен метод \_\_init\_\_, инициализирующий следующие поля: average\_speed, max\_speed, price, cargo, color. В случае несоответствия какого-либо из полей появляется ошибка ValueError('Invalid value').

В дочерних классах с помощью метода \_\_init\_\_ и функции super() определены все поля родительского класса, а также по паре особенных полей.

**Особенные поля класса Car** - power, wheels. В случае несоответствия появляется ошибка ValueError('Invalid value').

Прочие методы класса Car:

* \_\_str\_\_ - возвращает строку в формате «Car: средняя скорость <средняя скорость>, максимальная скорость <максимальная скорость>, цена <цена>, грузовой <грузовой>, цвет <цвет>, мощность <мощность>, количество колес <количество колес>».
* \_\_add\_\_ - суммирует среднюю и максимальную скорость, возвращает полученное значение.
* \_\_eq\_\_ - сравнивает два экземпляра класса и возвращает True или False. Два объекта типа Car равны, если равны количество колес, средняя скорость, максимальная скорость и мощность.

**Особенные поля класса Plane** - load\_capacity, wingspan. В случае несоответствия появляется ошибка ValueError('Invalid value').

Прочие методы класса Plane:

* \_\_str\_\_ - возвращает строку в формате «Plane: средняя скорость <средняя скорость>, максимальная скорость <максимальная скорость>, цена <цена>, грузовой <грузовой>, цвет <цвет>, грузоподъемность <грузоподъемность>, размах крыльев <размах крыльев>».
* \_\_add\_\_ - суммирует среднюю и максимальную скорость, возвращает полученное значение.
* \_\_eq\_\_ - сравнивает два экземпляра класса и возвращает True или False. Два объекта типа Plane равны по размерам, если равны размах крыльев.

**Особенные поля класса Ship** - length, side\_height. В случае несоответствия появляется ошибка ValueError('Invalid value').

Прочие методы класса Ship:

* \_\_str\_\_ - возвращает строку в формате «Ship: средняя скорость <средняя скорость>, максимальная скорость <максимальная скорость>, цена <цена>, грузовой <грузовой>, цвет <цвет>, длина <длина>, высота борта <высота борта>».
* \_\_add\_\_ - суммирует среднюю и максимальную скорость, возвращает полученное значение.
* \_\_eq\_\_ - сравнивает два экземпляра класса и возвращает True или False. Два объекта типа Ship равны по размерам, если равны их длина и высота борта.

**Группа 2**

В группе 2 находятся классы CarList, PlaneList, ShipList – дочерние классы стандартного класса list. Все инициализируются с помощью метода \_\_init\_\_ и super(). Каждый класс имеет свои особенные методы.

**Особенные методы класса CarList:**

* append – добавляет в конец списка экземпляр класса, иначе выводится ошибка TypeError с текстом: Invalid type <тип\_объекта p\_object>.
* print\_colors – печатает порядковые номера всех автомобилей и их цвет путем итерации и обращения к полю color.
* print\_count – выводит количество автомобилей (длину экземпляра).

**Особенные методы класса PlaneList:**

* extend - переопределение метода extend() списка. В случае, если элемент iterable - объект класса Plane, этот элемент добавляется в список, иначе не добавляется.
* print\_colors – печатает порядковые номера всех самолетов и их цвет путем итерации и обращения к полю color.
* total\_speed – считает и выводит суммарную среднюю скорость всех самолетов.

**Особенные методы класса ShipList:**

* append – добавляет в конец списка экземпляр класса, иначе выводится ошибка TypeError с текстом: Invalid type <тип\_объекта p\_object>.
* print\_colors – печатает порядковые номера всех автомобилей и их цвет путем итерации и обращения к полю color.
* print\_ship – печатает номера всех кораблей, длина которых больше 150 метров.

**Тестирование**

Результаты тестирования представлены в таблице 1.

Таблица 1 – Результаты тестирования

|  |  |  |
| --- | --- | --- |
| Ожидание | Выходные данные | Комментарии |
| 70 200 50000 True w  70 200 50000 True w 100 4  Car: средняя скорость 70, максимальная скорость 200, цена 50000, грузовой True, цвет w, мощность 100, количество колес 4.  270  True  70 200 50000 True w 1000 150  Plane: средняя скорость 70, максимальная скорость 200, цена 50000, грузовой True, цвет w, грузоподъемность 1000, размах крыльев 150.  270  True  70 200 50000 True w 200 100  Ship: средняя скорость 70, максимальная скорость 200, цена 50000, грузовой True, цвет w, длина 200, высота борта 100.  270  True  1 автомобиль: w  2 автомобиль: w  2  1 самолет: w  2 самолет: w  140  1 корабль: w  2 корабль: w  Длина корабля №1 больше 150 метров  Длина корабля №2 больше 150 метров | 70 200 50000 True w  70 200 50000 True w 100 4  Car: средняя скорость 70, максимальная скорость 200, цена 50000, грузовой True, цвет w, мощность 100, количество колес 4.  270  True  70 200 50000 True w 1000 150  Plane: средняя скорость 70, максимальная скорость 200, цена 50000, грузовой True, цвет w, грузоподъемность 1000, размах крыльев 150.  270  True  70 200 50000 True w 200 100  Ship: средняя скорость 70, максимальная скорость 200, цена 50000, грузовой True, цвет w, длина 200, высота борта 100.  270  True  1 автомобиль: w  2 автомобиль: w  2  1 самолет: w  2 самолет: w  140  1 корабль: w  2 корабль: w  Длина корабля №1 больше 150 метров  Длина корабля №2 больше 150 метров | Корректно |

**Выводы**

Были изучены основные парадигмы программирования, был написан код, создающий несколько классов и методы для работы с ними.

# ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Khangulyan\_Sargis\_lb1.py

class Transport:

def \_\_init\_\_(self, average\_speed, max\_speed, price, cargo, color):

if (isinstance(average\_speed, int) and isinstance(max\_speed, int)

and isinstance(price, int) and isinstance(cargo, bool)

and average\_speed > 0 and max\_speed > 0 and price > 0

and (color == "w" or color == "b" or color == "g")):

self.average\_speed = average\_speed

self.max\_speed = max\_speed

self.price = price

self.cargo = cargo

self.color = color

else: 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)

if (isinstance(power, int) and isinstance(wheels, int)

and power > 0 and 0 < wheels < 11):

self.power = power

self.wheels = wheels

else: 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.average\_speed + self.max\_speed

def \_\_eq\_\_(self, other):

if (self.wheels == other.wheels

and self.average\_speed == other.average\_speed

and self.max\_speed == other.max\_speed

and self.power == other.power):

return True

return False

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)

if (isinstance(load\_capacity, int) and isinstance(wingspan, int)

and load\_capacity > 0 and wingspan > 0):

self.load\_capacity = load\_capacity

self.wingspan = wingspan

else: 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.average\_speed + self.max\_speed

def \_\_eq\_\_(self, other):

if (self.wingspan == other.wingspan):

return True

return False

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)

if (isinstance(length, int) and isinstance(side\_height, int)

and length > 0 and side\_height > 0):

self.length = length

self.side\_height = side\_height

else: 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.average\_speed + self.max\_speed

def \_\_eq\_\_(self, other):

if (self.length == other.length and self.side\_height == other.side\_height):

return True

return False

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 {type(p\_object)}")

def print\_colors(self):

for i in range(len(self)):

print(f"{i+1} автомобиль: {self[i].color}")

def print\_count(self):

print(len(self))

class PlaneList(list):

def \_\_init\_\_(self, name):

super().\_\_init\_\_()

self.name = name

def extend(self, iterable):

filtered\_iterable = []

for i in iterable:

if (isinstance(i, Plane)):

filtered\_iterable.append(i)

super().extend(filtered\_iterable)

def print\_colors(self):

for i in range(len(self)):

print(f"{i+1} самолет: {self[i].color}")

def total\_speed(self):

sum = 0

for i in range(len(self)):

sum += self[i].average\_speed

print(sum)

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 {type(p\_object)}")

def print\_colors(self):

for i in range(len(self)):

print(f"{i+1} корабль: {self[i].color}")

def print\_ship(self):

for i in range(len(self)):

if self[i].length > 150:

print(f'Длина корабля №{i+1} больше 150 метров')