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 _____ Волков А.А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2024

Цель работы

Изучить основы парадигмы ООП и работу с исключениями в языке программирования Python. Реализовать иерархию наследуемых от общего родителя классов и вспомогательные классы для работы с объектами первых.

Задание

Вариант 3.

Базовый класс - транспорт Transport:

class Transport:

Поля объекта класс Transport:

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

2) максимальная скорость (в км/ч, положительное целое число)

3) цена (в руб., положительное целое число)

4) грузовой (значениями могут быть или True, или False)

5) 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:

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

2) максимальная скорость (в км/ч, положительное целое число)

3) цена (в руб., положительное целое число)

4) грузовой (значениями могут быть или True, или False)

5) цвет (значение может быть одной из строк: w (white), g(gray), b(blue)).

6) мощность (в Вт, положительное целое число)

7) 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:

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

2) максимальная скорость (в км/ч, положительное целое число)

3) цена (в руб., положительное целое число)

4) грузовой (значениями могут быть или True, или False)

5) цвет (значение может быть одной из строк: w (white), g(gray), b(blue)).

6) грузоподъемность (в кг, положительное целое число)

7) 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:

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

2) максимальная скорость (в км/ч, положительное целое число)

3) цена (в руб., положительное целое число)

4) грузовой (значениями могут быть или True, или False)

5) цвет (значение может быть одной из строк: w (white), g(gray), b(blue)).

6) длина (в м, положительное целое число)

7) 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.

Автомобили:

class CarList – список автомобилей - наследуется от класса list.

Конструктор:

1) Вызвать конструктор базового класса.

2) 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:

1) Вызвать конструктор базового класса.

2) 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:

1) Вызвать конструктор базового класса.

2) 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]>

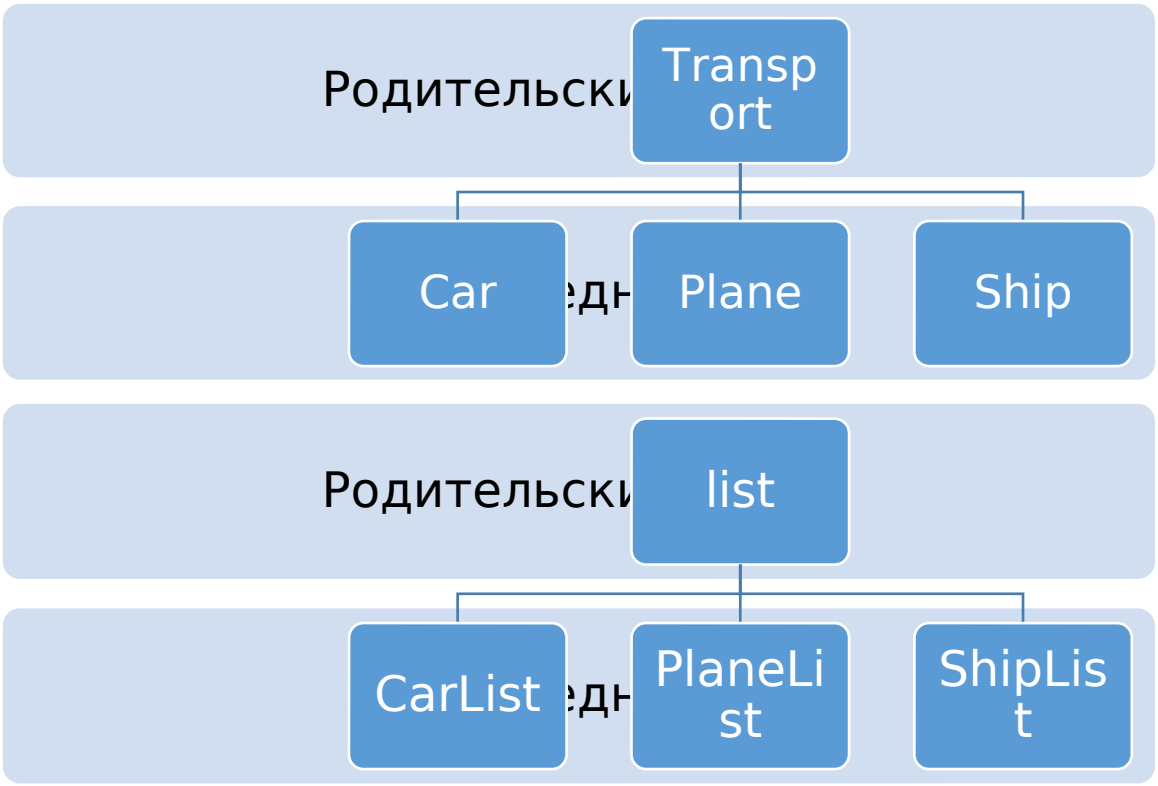
Метод print_ship(): вывести те корабли, чья длина больше 150 метров, в виде строки:

Длина корабля №<i> больше 150 метров

Длина корабля №<j> больше 150 метров

Выполнение работы

Для большей наглядности ниже представлены схемы, отображающие иерархию наследования требуемых классов.



Те классы, которые наследуются используются для того, чтобы расширить или изменить функционал базовых классов, так как они имеют доступ ко всем их атрибутам.

Методы, описанные в классах, унаследованных от класса Transport:

1) Метод *__init__* (конструктор): принимает на вход параметры, большую часть из которых с помощью функции *super* обрабатывает и проверяет на корректность конструктор базового класса. «Особенные» поля, которые принадлежат определенному виду транспорта, инициализируются и проверяются непосредственно в классе наследнике.

2) Метод *__str__* (используется для представления информации об объекте в строковом виде для пользователя, например: функции *print* и *str*): возвращает строку, содержащую информацию о текущем экземпляре класса.

3) Метод *__add__* (при наличии двух экземпляров его возвращенное значение является результатом оператора «+»): в нашем случае метод не получает на вход другого экземпляра, а просто возвращает сумму средней и максимальной скоростей текущего экземпляра.

4) Метод *__eq__* (вызываемый при использовании оператора сравнения «==»): возвращает *True*, если равны требуемые требуемые поля экземпляров класса, *False* в противном случае.

Методы, описанные в классах, унаследованных от класса list:

1) Метод *__init__* (конструктор): принимает на вход параметр *name*, которые присваивается полю *name* после вызова конструктора базового класса.

2) Метод *print_colors*: распечатывает строку по образцу, используя свои элементы.

3) Метод *append* (переопределен в классах *CarList* и *ShipList*): добавляет в список только те объекты, которые являются экземплярами данного класса, иначе выкидывает ошибку.

4) Метод *print_ship* (реализован в классе *ShipList*): выводит строку по образцу.

5) Метод *print_count* (реализован в классе *CarList*): выводит количество элементов данного списка.

6) Метод *extend* (переопределен в классе *PlaneList*): расширяет данный список только элементами класса *Plane* из поданного на вход списка.

7) Метод *total_speed* (реализован в классе *PlaneList*): выводит на экран суммарную среднюю скорость всех самолетов в данном списке.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	try: car = Car(80, 170, 487500, False, 'w', 0, 5) except ValueError as err: print(err) car_list = CarList("List_car") car_list.append(Car(50, 120, 245000, False, 'w', 250, 4)) car_list.append(Car(50, 120, 245000, False, 'b', 250, 4)) print(car_list[1]) print(car_list[0] == car_list[1]) print(car_list[0].__add__()) car_list.print_count() car_list.print_colors()	Invalid value Car: средняя скорость 50, максимальная скорость 120, цена 245000, грузовой False, цвет b, мощность 250, количество колес 4. True 170 2 1 автомобиль: w 2 автомобиль: b	Корректно выкидывается ошибка при попытке создать экземпляр с некорректными данными. Остальные методы классов работают корректно
2.	try: plane = Plane(1000, 1500, 9500000, "нет", 'w', 2000, 56) except ValueError as err: print(err) plane_list = PlaneList("List_plane") plane_list.append(Plane(1000, 1500, 10000000, False, 'w', 2000, 54)) plane_list.append(Plane(1000, 1500, 15000000, False, 'b', 2500, 59)) print(plane_list[0]) plane_list.extend([Plane(1700, 2500, 20000000, False, 'g', 40, 40), 'fkjlads', 21]) plane_list.print_colors() plane_list.total_speed()	Invalid value Plane: средняя скорость 1000, максимальная скорость 1500, цена 10000000, грузовой False, цвет w, грузоподъемность 2000, размах крыльев 54. 1 самолет: w 2 самолет: b 3 самолет: g 3700	Корректно выкидывается ошибка при попытке создать экземпляр с некорректными данными. Остальные методы классов работают корректно

3.	ship_lst = ShipList('ship list') try: ship_lst.append('str') except TypeError as msg: print(msg) ship_lst.append(Ship(50, 100, 10000000, False, 'w', 160, 40)) ship_lst.append(Ship(50, 100, 10000000, False, 'b', 180, 40)) ship_lst.append(Ship(50, 100, 10000000, False, 'g', 140, 40)) print(ship_lst[0] == ship_lst[1]) ship_lst.print_ship() ship_lst.print_colors()	Invalid type <class 'str'> False Длина корабля №1 больше 150 метров Длина корабля №2 больше 150 метров 1 корабль: w 2 корабль: b 3 корабль: g	Корректно выкидывается ошибка при попытке добавить экземпляр иного типа. Остальные методы классов работают корректно

Выводы

Изучены основы работы с ООП и обработка исключений в языке программирования Python. Разработаны две системы классов, наследуемых от общего родителя, также реализовано их элементарное взаимодействие. Освоена парадигма наследования в ООП. Для некорректных данных предусмотрен механизм выкидывания ошибок.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb_1.py

```
class Transport:
    def __init__(self, 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 "wgb"
        ):
            raise ValueError("Invalid value")
        self.average_speed = average_speed
        self.max_speed = max_speed
        self.price = price
        self.cargo = cargo
        self.color = color


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 not (
            isinstance(power, int)
            and power > 0
            and isinstance(wheels, int)
            and 1 <= wheels <= 10
        ):
            raise ValueError("Invalid value")
        self.power = power
        self.wheels = wheels

    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):
        requirements = (
            self.wheels == other.wheels,
            self.average_speed == other.average_speed,
            self.max_speed == other.max_speed,
            self.power == other.power,
        )
        return all(requirements)
```

```
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 not (
            isinstance(load_capacity, int)
            and load_capacity > 0
            and isinstance(wingspan, int)
            and wingspan > 0
        ):
            raise ValueError("Invalid value")
        self.load_capacity = load_capacity
        self.wingspan = wingspan

    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):
        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)
        if not (
            isinstance(length, int)
            and length > 0
            and isinstance(side_height, int)
            and side_height > 0
        ):
            raise ValueError("Invalid value")
        self.length = length
        self.side_height = side_height

    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):
```

```
            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 not isinstance(p_object, Car):
            raise TypeError(f"Invalid type {type(p_object)}")
        super().append(p_object)

    def print_colors(self):
        colors_cars = []
        for i in range(len(self)):
            colors_cars.append(f'{i+1} автомобиль: {self[i].color}')
        print('\n'.join(colors_cars))

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


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

    def extend(self, lst):
        super().extend(filter(lambda x: isinstance(x, Plane), lst))

    def print_colors(self):
        colors_planes = []
        for i in range(len(self)):
            colors_planes.append(f'{i+1} самолет: {self[i].color}')
        print('\n'.join(colors_planes))

    def total_speed(self):
        print(sum(i.average_speed for i in self))


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

    def append(self, p_object):
        if not isinstance(p_object, Ship):
            raise TypeError(f"Invalid type {type(p_object)}")
        super().append(p_object)

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

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