МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Информатика»
Тема: Парадигмы программирования.

Студент гр. 3341 Ступак А.А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2024

Цель работы

Изучить и научиться применять принципы объектно-ориентированного программирования для написания программы с использованием нескольких классов. Использовать классы для решения поставленных задач, а так же обработать исключения.Задание

На вход программе подается текст, представляющий собой набор предложений с новой строки. Текст заканчивается предложением "Fin." В тексте могут встречаться примеры запуска программ в командной строке Linux. Требуется, используя регулярные выражения, найти только примеры команд в оболочке суперпользователя и вывести на экран пары <имя пользователя> - <имя_команды>. Если предложение содержит какой-то пример команды, то гарантируется, что после нее будет символ переноса строки.

Примеры имеют слеующий вид:

Сначала идет имя пользователя, состоящее из букв, цифр и символа _

Символ @

Имя компьютера, состоящее из букв, цифр, символов _ и -

Символ : и ~

Символ $, если команда запущена в оболочке пользователя и #, если в оболочке суперпользователя. При этом между двоеточием, тильдой и $ или # могут быть пробелы.

Пробел

Сама команда и символ переноса строки.

Задание

Вариант 3.

Базовый класс - транспорт Transport:

class 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'.

Автомобиль - Car:

class 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__(): 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:

- 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.

Корабль - Ship:

class 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)).
- 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

Описание методов классов:

- Transport
  - __init__(self, average_speed, max_speed, price, cargo, color): создает экземпляр класса с полями, заданными аргументами, проверяя их значения на допустимые.
- Car. Родительский класс - Transport
  - __init__(self, average_speed, max_speed, price, cargo, color, power, wheels): создает экземпляр класса с полями, заданными аргументами, проверяя их значения на допустимые.
  - __str__(self): вызывается, когда требуется строковое представление объекта.
  - __add__(self): возвращает результат сложения значений полей average_speed и max_speed.
  - __eq__(car1, car2): используется для сравнения двух объектов класса, возвращает True, если они равны.
- Plane. Родительский класс - Transport
  - __init__(self, average_speed, max_speed, price, cargo, color, load_capacity, wingspan): создает экземпляр класса с полями, заданными аргументами, проверяя их значения на допустимые.
  - __str__(self): вызывается, когда требуется строковое представление объекта.
  - __add__(self): возвращает результат сложения значений полей average_speed и max_speed.
  - __eq__(car1, car2): используется для сравнения двух объектов класса, возвращает True, если они равны.
- Ship. Родительский класс - Transport

- __init__(self, average_speed, max_speed, price, cargo, color, length, side_height): создает экземпляр класса с полями, заданными аргументами, проверяя их значения на допустимые.
- __str__(self): вызывается, когда требуется строковое представление объекта.
- __add__(self): возвращает результат сложения значений полей average_speed и max_speed.
- __eq__(car1, car2): используется для сравнения двух объектов класса, возвращает True, если они равны.

- CarList. Родительский класс - list
  - __init__(self, name): создает экземпляр класса list, добавляя к нему поле name.
  - append(self, p_object): добавляет в список новый элемент, если тот является экземпляром класса Car.
  - print_colors(self): выводит значение поля color для всех элементов списка.
  - print_count(self): выводит количество элементов в списке.
- PlaneList. Родительский класс - list
  - __init__(self, name): создает экземпляр класса list, добавляя к нему поле name.
  - extend(self, iterable): добавляет в список несколько элементов, если те являются экземплярами класса Plane.
  - print_colors(self): выводит значение поля color для всех элементов списка.
  - total_speed(self): подсчитывает и выводит среднее значение поля average_speed всех элементов списка.
- ShipList. Родительский класс - list
  - __init__(self, name): создает экземпляр класса list, добавляя к нему поле name.

- append(self, p_object): добавляет в список новый элемент, если тот является экземпляром класса Ship.
- print_colors(self): выводит значение поля color для всех элементов списка.
- print_ship(self): выводит элементы списка, у которых значение поля length больше 150.

Переопределенные методы класса list для CarList, PlaneList и ShipList будут работать, т. к. родительский метод вызывается с помощью функции super(). Так, например, метод append(self, p_object) класса ShipList только проверяет класс добавляемого элемента, после чего вызывает метод append() класса list.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии

1.	transport = Transport(70, 200, 50000, True, 'w') #транспорт
	car1 = Car(70, 200, 50000, True, 'w', 100, 4) #авто
	car2 = Car(70, 200, 50000, True, 'w', 100, 4)
	car_list = CarList(Car) #список авто
	car_list.append(car1)
	car_list.append(car2)
	car_list.print_colors()
	car_list.print_count()
		1 автомобиль: w
		2 автомобиль: w
		2
			Выходные данные соответствуют ожиданиям.

2.	try: #неправильные данные для транспорта
	transport = Transport(-70, 200, 50000, True, 'w')
	except (TypeError, ValueError):
	print('OK')
		OK	Выходные данные соответствуют ожиданиям.

3.	plane1 = Plane(70, 200, 50000, True, 'w', 1000, 150) plane2 = Plane(70, 200, 50000, True, 'w', 1000, 150) print(plane1.average_speed, plane1.max_speed, plane1.price, plane1.cargo, plane1.color,	Plane: средняя скорость 70, максимальная скорость 200, цена 50000, грузовой True, цвет w, грузоподъемность	Выходные данные соответствуют ожиданиям.

	plane1.load_capacity, plane1.wingspan) print(plane1.__str__()) print(plane1.__add__()) print(plane1.__eq__(plane2))	1000, размах крыльев 150. 270 True	

Выводы

В ходе выполнения лабораторной работы были изучены и освоены необходимые навыки для реализации программы, используя принципы объектно-ориентированного программирования. Были написаны классы и обработаны исключения.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Transport:
    def __init__(self, average_speed, max_speed, price, cargo, color):
                    if not isinstance(average_speed, int) or not
isinstance(max_speed, int) or not isinstance(price, int) or not
isinstance(cargo, bool):
            raise ValueError("Invalid value")

        if average_speed <= 0 or max_speed <= 0 or price <= 0 or color
not 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) or not isinstance(wheels, int):
            raise ValueError("Invalid value")

        if power <= 0 or wheels <= 0 or 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__(car1, car2):
             if car1.wheels == car2.wheels and car1.average_speed ==
car2.average_speed and car1.max_speed == car2.max_speed and car1.power
== car2.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 not isinstance(load_capacity, int) or not
isinstance(wingspan, int):
            raise ValueError("Invalid value")

        if load_capacity <= 0 or 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__(plane1, plane2):
        return plane1.wingspan == plane2.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) or not isinstance(side_height,
int):
            raise ValueError("Invalid value")

        if length <= 0 or 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__(ship1, ship2):
          return ship1.length == ship2.length and ship1.side_height ==
ship2.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):
        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):
           if len([i for i in iterable if isinstance(i, Plane)]) ==
len(iterable):
            super().extend(iterable)

    def print_colors(self):
        for i in range(len(self)):
            print(f"{i+1} самолет: {self[i].color}")

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

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):
        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")
```