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(ка) гр. 3383 Логинова А. Ю.

Преподаватель Иванов Д. В.

Санкт-Петербург

2024

Цель работы.

Целью данной практической работы является развитие навыков написания классов на языке программирования Python, что позволит углубить понимание объектно-ориентированного программирования и принципов работы с данными в Python. Работа направлена на практическое применение теоретических знаний, полученных в процессе изучения языка программирования, и на развитие навыков работы с Python в реальных задачах.

Задание.

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 ...

В отчете укажите:

1. Изображение иерархии описанных вами классов.
2. Методы, которые вы переопределили (в том числе методы класса object).
3. В каких случаях будут использованы методы __str__() и __eq__().
4. Будут ли работать переопределенные методы класса list для CarList, PlaneList и ShipList? Объясните почему и приведите примеры.

Выполнение работы.

Данная лабораторная работа по программированию на Python включает в себя несколько классов, каждый из которых представляет собой транспортное средство с различными характеристиками и методами. Рассмотрим каждый класс по очереди, начиная с базового класса Transport и заканчивая классами CarList, PlaneList и ShipList.

Класс Transport

- Инициализация: Класс Transport определяет базовые атрибуты для всех транспортных средств: среднюю скорость, максимальную скорость, цену, грузоподъемность и цвет. Конструктор __init__ проверяет, что все входные значения корректны, иначе выбрасывает исключение ValueError.
- Метод __add__: Этот метод позволяет складывать среднюю и максимальную скорость транспортного средства.

Класс Car

- Наследование: Класс Car наследует от Transport и добавляет атрибуты power (мощность) и wheels (количество колес), а также переопределяет метод __str__ для вывода информации о автомобиле.
- Метод __eq__: Сравнивает автомобили по количеству колес и скоростям.

Класс Plane

- Наследование: Класс Plane также наследует от Transport и добавляет атрибуты load_capacity (грузоподъемность) и wingspan (размах крыльев), а также переопределяет метод __str__ для вывода информации о самолете.
- Метод __eq__: Сравнивает самолеты по размаху крыльев.

Класс Ship

- Наследование: Класс Ship наследует от Transport и добавляет атрибуты length (длина) и side_height (высота борта), а также переопределяет метод __str__ для вывода информации о корабле.
- Метод __eq__: Сравнивает корабли по длине и высоте борта.

Класс CarList

- Наследование: Класс CarList наследует от list и добавляет атрибут name для идентификации списка автомобилей.
- Методы: append добавляет автомобиль в список, если он является экземпляром класса Car. print_colors выводит цвета всех автомобилей в списке. print_count выводит количество автомобилей в списке.

Класс PlaneList

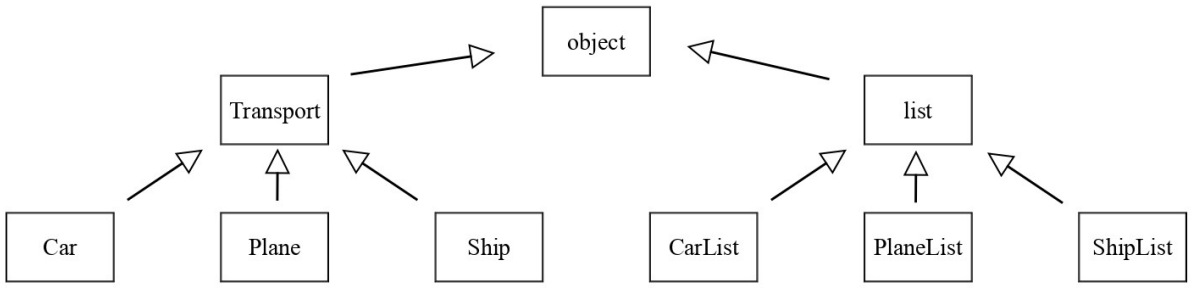
- Наследование: Класс PlaneList наследует от list и добавляет атрибут name для идентификации списка самолетов.
- Методы: extend добавляет в список только экземпляры класса Plane. print_colors выводит цвета всех самолетов в списке. total_speed вычисляет и выводит общую среднюю скорость всех самолетов в списке.

Класс ShipList

- Наследование: Класс ShipList наследует от list и добавляет атрибут name для идентификации списка кораблей.
- Методы: append добавляет корабль в список, если он является экземпляром класса Ship. print_colors выводит цвета всех кораблей в списке. print_ship выводит информацию о кораблях, длина которых больше 150 метров.

Задание 1

Иерархия классов



Задание 2

В лабораторной работе были переопределены следующие методы класса list: append(self, value: _T), extend(self, values: Iterable[_T]), класса object: __eq__(self, __value: object), __str__(self), __init__(self).

Задание 3

Метод __str__() вызывается при использовании функций print(), str() и format() с экземпляром класса. Метод __eq__() используется для определения поведения оператора равенства (==) для объектов класса.

Задание 4

Переопределенные методы класса list будут работать для CarList, PlaneList и ShipList, но согласно новой логике, описанной в подклассе, наследующемся от класса list. Переопределенные методы работают потому, что переопределение методов — ключевой аспект объектно-ориентированного программирования, позволяющий расширять и/или изменять поведение методов. Примеры использования переопределенного метода append() класса list:

```
class Car:
  def __init__(self, make, model):
      self.make = make
      self.model = model
class CarList(list):
```

```
    def append(self, car):
        if not isinstance(car, Car):
            raise TypeError("Only Car objects can be added to the list")
        super().append(car)
# Создаем экземпляры класса Car
car1 = Car("Toyota", "Corolla")
car2 = Car("Honda", "Civic")
# Создаем экземпляр класса CarList и добавляем в него автомобили
car_list = CarList()
car_list.append(car1)
car_list.append(car2)
# Попытка добавить не-Car объект вызовет ошибку
try:
    car_list.append("Not a Car object")
except TypeError as e:
print(e)
```

Тестирование.

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

try:
    transport = Transport(70, -200, 50000,
True, 'w')
except (TypeError, ValueError):
    print('OK')

try:
    transport = Transport(70, 200, -50000,
True, 'w')
except (TypeError, ValueError):
    print('OK')

try:
    transport = Transport(70, 200, 50000,
-1, 'w')
except (TypeError, ValueError):
    print('OK')

try:
    transport = Transport(70, 200, 50000,
True, -1)
except (TypeError, ValueError):
    print('OK')

try:
    transport = Transport('a', 200, 50000,
True, 'w')
except (TypeError, ValueError):
    print('OK')

try:
    transport = Transport(70, 'a', 50000,
True, 'w')
except (TypeError, ValueError):
    print('OK')

try:
     transport = Transport(70, 200, 'a',
True, 'w')
except (TypeError, ValueError):
    print('OK')

try:
    transport = Transport(70, 200, 50000,
'a', 'w')
except (TypeError, ValueError):
```

```
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
```

print('OK') try: transport = Transport(70, 200, 50000, True, 'a') except (TypeError, ValueError): print('OK') try: transport = Transport(0, 200, 50000, True, 'w') except (TypeError, ValueError): print('OK') try: transport = Transport(70, 0, 50000, True, 'w') except (TypeError, ValueError): print('OK') try: transport = Transport(70, 200, 0, True, 'w') except (TypeError, ValueError): print('OK') try: transport = Transport(70, 200, 50000, 0, 'w') except (TypeError, ValueError): print('OK') try: transport = Transport(70, 200, 50000, True, 0) except (TypeError, ValueError): print('OK')	2 корабль: w Длина корабля №1 больше 150 метров Длина корабля №2 больше 150 метров

Выводы.

В ходе выполнения практической работы были изучены и применены ключевые концепции объектно-ориентированного программирования на Python, включая наследование и полиморфизм. Наследование позволило создать иерархию классов транспортных средств, обеспечивая повторное использование кода и упрощение его поддержки. Полиморфизм, достигаемый через наследование, позволил объектам разных классов вести себя по-разному, но с использованием одного и того же интерфейса, что подчеркнуло гибкость и расширяемость кода.

ПРИЛОЖЕНИЕ А

Исходный код программы.

Название файла: main.py

```
class Transport:
      def __init__(self, average_speed, max_speed, price, cargo,
color):
        if (average_speed > 0 and max_speed > 0 and price > 0
                and isinstance(cargo, bool)
                   and (color == 'w' or color == 'g' or color ==
'b')):

            self.average_speed = average_speed
            self.max_speed = max_speed
            self.price = price
            self.cargo = cargo
            self.color = color

        else:
            raise ValueError("Invalid value")

    def __add__(self):
        return self.average_speed + self.max_speed


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 power > 0 and 0 < wheels <= 10:
            self.power = power
            self.wheels = wheels
        else:
            raise ValueError("Invalid value")

    def __str__(self):
          string = (f"Car: средняя скорость {self.average_speed},
максимальная скорость {self.max_speed}, "
                    f"цена {self.price}, грузовой {self.cargo}, цвет
{self.color}, "
                           f"мощность {self.power}, количество колес
{self.wheels}.")

        return string

    def __eq__(self, other):
         if self.wheels == other.wheels and self.average_speed ==
other.average_speed and self.max_speed == other.max_speed:
            return True
        else:
            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 load_capacity > 0 and wingspan > 0:
            self.load_capacity = load_capacity
            self.wingspan = wingspan
        else:
            raise ValueError("Invalid value")

    def __str__(self):
         string = (f"Plane: средняя скорость {self.average_speed},
максимальная скорость {self.max_speed}, "
                   f"цена {self.price}, грузовой {self.cargo}, цвет
{self.color}, "
                    f"грузоподъемность {self.load_capacity}, размах
крыльев {self.wingspan}.")
        return string

    def __eq__(self, other):
        if self.wingspan == other.wingspan:
            return True
        else:
            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 length > 0 and side_height > 0:
            self.length = length
            self.side_height = side_height
        else:
            raise ValueError("Invalid value")

    def __str__(self):
          string = (f"Ship: средняя скорость {self.average_speed},
максимальная скорость {self.max_speed}, "
                   f"цена {self.price}, грузовой {self.cargo}, цвет
{self.color}, "
                               f"длина {self.length}, высота борта
{self.side_height}.")
        return string

    def __eq__(self, other):
           if self.length == other.length and self.side_height ==
other.side_height:
            return True
```

```
        else:
            return False


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

    def append(self, obj):
        if isinstance(obj, Car):
            super().append(obj)
        else:
            raise TypeError(f"Invalid type {type(obj)}")

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

    def print_count(self):
        count = 0
        for car in self:
            count += 1
        print(count)



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

    def extend(self, obj):
        obj = list(filter(lambda x: isinstance(x, Plane), obj))
        super().extend(obj)

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

    def total_speed(self):
        all_speed = 0
        for plane in self:
            all_speed += plane.average_speed
        print(all_speed)


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

```
def append(self, obj):
    if isinstance(obj, Ship):
        super().append(obj)
    else:
        raise TypeError(f"Invalid type {type(obj)}")

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")
```