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.

Студентка гр. 3341 _____ Байрам Э.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2024

Цель работы

Цель этой работы была создание иерархии классов для представления различных персонажей (воинов, магов, лучников) и их списков. Определили основные атрибуты и методы для каждого класса, а также переопределили методы базового класса object для улучшения функциональности и взаимодействия с объектами.

Задание

Создать класс с названием Transport, определить его свойства и конструктор.

Создать класс Car, унаследованный от класса Transport, определить его свойства, конструктор и специальные методы (__str__(), __add__(), __eq__()).

Создать класс Plane, унаследованный от класса Transport, определить его свойства, конструктор и специальные методы (__str__(), __add__(), __eq__()).

Создать класс Ship, унаследованный от класса Transport, определить его свойства, конструктор и специальные методы (__str__(), __add__(), __eq__()).

Создать три отдельных класса списка: CarList, PlaneList и ShipList, определить их свойства и специальные методы (append(), extend(), print_colors(), print_count(), total_speed(), print_ship()).

Основные теоретические положения

Основные теоретические положения в задании описывают создание и использование классов в объектно-ориентированном программировании (ООП), а также работу с наследованием, методами и перегрузкой операторов. Вот некоторые из основных теоретических концепций, которые касаются данного задания:

Классы и объекты: Классы являются шаблонами для создания объектов в Python. Они содержат атрибуты (переменные) и методы (функции), которые определяют поведение объектов.

Наследование: Наследование позволяет одному классу (подклассу) наследовать атрибуты и методы другого класса (родительского класса). В данном случае, классы Car, Plane и Ship наследуют свойства и методы от класса Transport.

Конструктор и инициализация: Конструктор (__init__() метод) используется для инициализации объектов при их создании. В нем определяются начальные значения атрибутов объекта.

Строковое представление объекта (__str__() метод): Метод __str__() определяет строковое представление объекта, которое будет использоваться при вызове функции print() для объекта данного класса.

Перегрузка операторов: В Python можно перегрузить операторы, такие как + (сложение) и == (равенство), чтобы объекты класса могли использовать эти операторы по-разному в зависимости от контекста.

Списки и управление списками: В задании также упоминаются специальные списки (CarList, PlaneList, ShipList), которые являются расширениями встроенного класса list для управления объектами определенных типов (автомобили, самолеты, корабли).

Эти основные концепции являются фундаментальными для понимания работы с классами и объектами в Python, особенно в контексте ООП.

Выполнение работы

1. Класс Transport: Создайте класс Transport с следующими свойствами:

average_speed (средняя скорость)

max_speed (максимальная скорость)

price (цена)

cargo (грузовой, True или False)

color (цвет, w (white), g (gray), b (blue))

2. Класс Car: Унаследуйте класс Car от класса Transport и добавьте следующие свойства:

power (мощность)

wheels (количество колес, от 1 до 10)

3. Класс Plane: Унаследуйте класс Plane от класса Transport и добавьте следующие свойства:

load_capacity (грузоподъемность)

wingspan (размах крыльев)

4. Класс Ship: Унаследуйте класс Ship от класса Transport и добавьте следующие свойства:

length (длина)

side_height (высота борта)

5. Классы CarList, PlaneList и ShipList: Определите эти классы с методами, указанными в задании:

append(p_object) (добавление)

extend(iterable) (расширение)

print_colors() (вывод цветов)

print_count() (вывод количества)

total_speed() (общая скорость)

print_ship() (вывод кораблей)

6. Создайте экземпляры классов и вызовите соответствующие методы для проверки их работы.

Выводы

1. Основные классы:

Transport: Базовый класс для всех видов транспорта. Определяет общие свойства, такие как средняя и максимальная скорости, цена, грузовой и цвет.

Car, Plane, Ship: Классы для конкретных видов транспорта, унаследованные от Transport. Они добавляют уникальные свойства, такие как мощность и количество колес для автомобилей, грузоподъемность и размах крыльев для самолетов, длина и высота борта для кораблей.

2. Списки транспорта:

CarList, PlaneList, ShipList: Эти классы-списки расширяют функциональность стандартного списка в Python для управления соответствующими типами транспорта. Они имеют специализированные методы, такие как добавление элементов, вывод цветов, подсчет количества, вычисление общей скорости и вывод информации о кораблях.

3. Методы и перегрузка операторов:

Каждый класс имеет перегруженный метод __str__() для создания строкового представления объекта, который используется при вызове print().

Также есть метод __eq__(), который определяет равенство между двумя объектами класса на основе определенных критериев (например, равенство количества колес и скоростей для автомобилей).

Для определенных классов (Car, Plane, Ship) перегружен оператор +, который возвращает сумму средней и максимальной скоростей.

4. Выводы:

Создание иерархии классов позволяет эффективно управлять различными типами данных и добавлять новые функциональности при необходимости.

Перегрузка методов и операторов обеспечивает удобное использование объектов классов и их сравнение.

Специализированные классы-списки (CarList, PlaneList, ShipList) дополняют стандартный список и упрощают работу с коллекциями объектов определенного типа.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
    class Transport:
        def __init__(self, average_speed, max_speed, price, cargo,
color):
            if (type(average_speed) == int and average_speed > 0 and
type(max_speed) == int and max_speed > 0 and type(price) == int and
price > 0 and type(cargo) == bool and color in ['w', 'g', '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')


    class Car(Transport): #Наследуется от класса Transport
        def __init__(self, average_speed, max_speed, price, cargo,
color, power, wheels):
            if (type(average_speed) == int and average_speed > 0 and
type(max_speed) == int and max_speed > 0 and type(price) == int and
price > 0 and type(cargo) == bool and color in ['w', 'g', 'b'] and
type(power) == int and power > 0 and type(wheels) == int and 0 < wheels
<= 10):
                self.average_speed = average_speed
                self.max_speed = max_speed
                self.price = price
                self.cargo = cargo
                self.color = color
                self.power = power
                self.wheels = wheels
            else:
                raise ValueError('Invalid value')


        def __str__(self):
            return f'Car: средняя скорость {self.average_speed}, макси
мальная скорость {self.max_speed}, цена {self.price}, грузовой
{self.cargo}, цвет {self.color}, мощность {self.power}, количество коле
с {self.wheels}.'

        def __add__(self):
            return self.average_speed + self.max_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): #Наследуется от класса Transport
        def __init__(self, average_speed, max_speed, price, cargo, color, load_capacity, wingspan):
            if type(average_speed) == int and average_speed > 0 and type(max_speed) == int and max_speed > 0 and type(price) == int and price > 0 and type(cargo) == bool and color in ['w', 'g', 'b'] and type(load_capacity) == int and load_capacity > 0 and type(wingspan) == int and wingspan > 0:
                self.average_speed = average_speed
                self.max_speed = max_speed
                self.price = price
                self.cargo = cargo
                self.color = color
                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):
            return self.wingspan == other.wingspan


    class Ship(Transport): #Наследуется от класса Transport
        def __init__(self, average_speed, max_speed, price, cargo, color, length, side_height):
            if (type(average_speed) == int and average_speed > 0 and type(max_speed) == int and max_speed > 0 and type(price) == int and price > 0 and type(cargo) == bool and color in ['w', 'g', 'b'] and type(length) == int and length > 0 and type(side_height) == int and side_height > 0):
                self.average_speed = average_speed
                self.max_speed = max_speed
                self.price = price
                self.cargo = cargo
                self.color = color
                self.length = length
                self.side_height = side_height
```

```
            else:
                raise ValueError('Invalid value')


        def __str__(self):
            return f'Ship: средняя скорость {self.average_speed}, мак
симальная 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):
            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):
            self.name = name


        def extend(self, iterable):
            if type(iterable) == list:
                for el in iterable:
                    if type(el) == Plane:
                        super().extend([el])

            else:
                raise TypeError

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

```
        def total_speed(self):
            print(sum([el.average_speed for el in self]))


    class ShipList(list):
        def __init__(self, name):
            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
')
```