**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.](https://e.moevm.info/mod/quiz/view.php?id=2173" \o "Тест)**

|  |  |  |
| --- | --- | --- |
| 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))

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

power (мощность)

wheels (количество колес, от 1 до 10)

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

load\_capacity (грузоподъемность)

wingspan (размах крыльев)

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

length (длина)

side\_height (высота борта)

1. 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)

1. 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.

1. 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.

1. 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.

1. 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}, 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):

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}, 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):

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')