**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: cредняя 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: cредняя 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: cредняя 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:

cредняя 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.

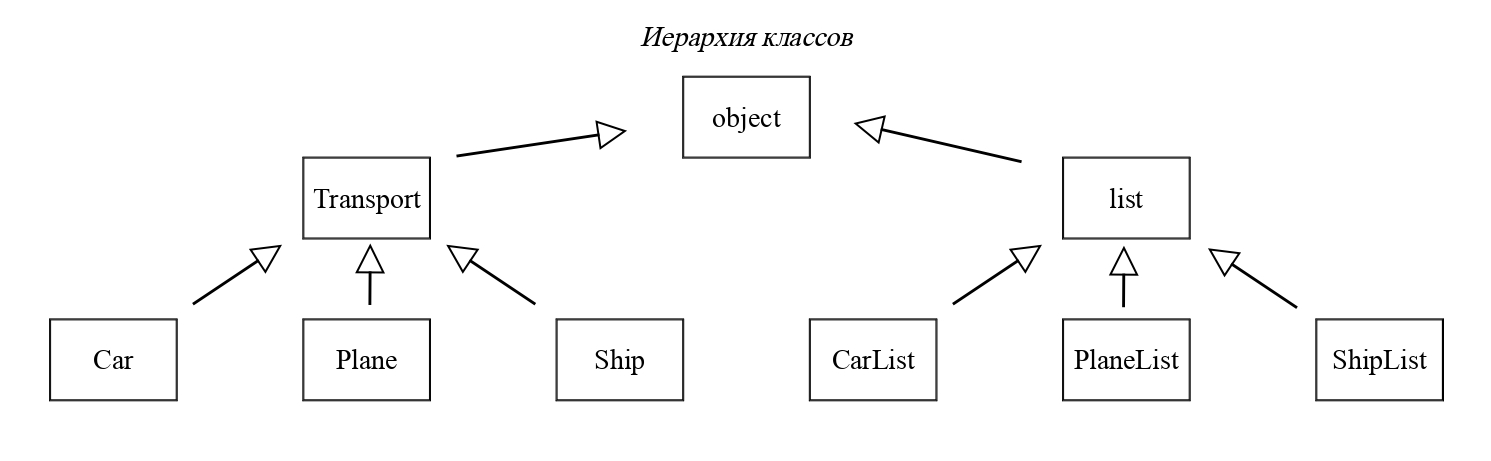
### Класс PlaneList

* 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**

**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):  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') | 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  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")