# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

# «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

**Кафедра МО ЭВМ**

# ОТЧЕТ

**по лабораторной работе №1**

# по дисциплине «Информационные технологии» Тема: Парадигмы программирования

|  |  |
| --- | --- |
| Студент гр. 2384 | Кузьминых Е.М |
| Преподаватель | Иванов Д.В. |

Санкт-Петербург 2023

# Цель работы.

Научиться применять ООП, написать программу на Python, удовлетворяющую парадигме ООП.

# Задачи.

Вариант №3

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:

Изображение иерархии описанных вами классов.

Методы, которые вы переопределили (в том числе методы класса object). В каких случаях будут использованы методы *str ()* и *eq ().*

Будут ли работать переопределенные методы класса *list* для *CarList*, *PlaneList* и *ShipList*? Объясните почему и приведите примеры.

# Выполнение работы.

В данном коде определены четыре класса: *Transport*, *Car*, *Plane*, и *Ship*.

Класс Transport имеет пять полей: *average\_speed* (средняя скорость), *max\_speed* (максимальная скорость), *price* (цена), *cargo* (грузовой или нет), и *color* (цвет), а также конструктор *init*, который принимает значения этих полей.

Класс *Car* наследуется от класса *Transport* и имеет два дополнительных поля: *power* (мощность) и *wheels* (количество колес), а также конструктор *init*, который также принимает значения этих полей. В классе *Car* также определены два метода: *str* и *eq*. Метод *str* возвращает строку, описывающую все поля класса *Car*. Метод *eq* сравнивает текущий объект с другим объектом класса Car и возвращает True, если значения всех полей равны у обоих объектов.

Класс *Plane* также наследуется от класса Transport и имеет два дополнительных поля: *load\_capacity* (грузоподъемность) и *wingspan* (размах крыльев), а также конструктор *init*, который принимает значения этих полей. В классе *Plane* также определены два метода: *str* и *eq*. Метод *str* возвращает строку, описывающую все поля класса *Plane*. Метод *eq* сравнивает текущий объект с другим объектом класса *Plane* и возвращает True, если значения поля wingspan равны у обоих объектов.

Класс *Ship* также наследуется от класса *Transport* и имеет два дополнительных поля: *length* (длина) и *side\_height* (высота борта), а также конструктор *init*, который принимает значения этих полей. В классе *Ship* также определены два метода: *str* и *eq*. Метод *str* возвращает строку, описывающую все поля класса *Ship*. Метод *eq* сравнивает текущий объект с другим объектом класса *Ship* и возвращает True, если значения всех полей равны у обоих объектов.

Затем мы определяем классы-списки для каждого типа транспортных средств, которые будут использоваться для хранения коллекции объектов каждого класса транспорта. Для каждого класса-списка мы переопределяем метод *append()*, чтобы он принимал только объекты транспорта соответствующего типа. Также мы определяем метод *print\_colors()*, который будет печатать цвет каждого объекта транспорта в списке.

*CarList*, *PlaneList*, и *ShipList* - это классы, которые наследуются от встроенного класса *list*. Они представляют собой список объектов соответствующего типа транспорта, т.е. *Car*, *Plane* и *Ship*. В этих классах переопределен метод *append()*, который проверяет, является ли добавляемый объект экземпляром соответствующего класса.

Иерархия классов можно представить в виде дерева (все классы наследуются от *object*) (см Рис.1 и Рис.2):

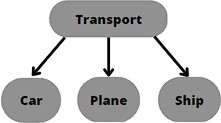


Рисунок 1 – Наследование от класса Transport

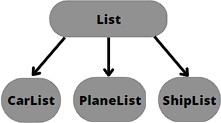


Рисунок 2 – наследование от класса List

Метод *str()* переопределен в классах *Car*, *Plane*, и *Ship*. Он используется для получения строкового представления объекта, которое может быть использовано для его вывода на экран или сохранения в файл. Метод *eq()* также переопределен в классах *Car*, *Plane*, и *Ship*. Он используется для сравнения объектов класса. В классах *Car*, *Plane*, и *Ship* он сравнивает значения полей объектов, а в классе Plane он сравнивает только поле wingspan.

Метод *str ()* используется для вывода информации о каждом объекте класса. Он возвращает строковое представление объекта, которое будет использоваться при вызове функции *print()*. Метод *eq ()* используется для сравнения двух объектов на равенство. Он должен возвращать True, если объекты равны, и False, если они не равны.

Переопределенные методы класса *list*, такие как *append()*, не будут работать для классов-списков *CarList*, *PlaneList* и *ShipList*, потому что они были переопределены и не принимают объекты, которые не являются экземплярами соответствующего класса. Например, если мы попытаемся добавить объект типа *Plane* в *CarList*, возникнет исключение *TypeError*.

# Тестирование.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Выходные данные | Комментарий |
| 1 | plane\_list =  PlaneList("my\_planes") plane1 = Plane(500, 800, 1000000, True, 'w', 2000, | Planes in the list: 1 самолет: w   1. самолет: b 2. самолет: g | Ответ верный |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 40)  plane2 = Plane(600, 900, 2000000, False, 'b', 2500,  45)  plane3 = Plane(700, 1000, 3000000, True, 'g', 3000,  50)  car1 = Car(80, 120, 10000,  False, 'w', 200, 4)  car2 = Car(90, 130, 20000,  True, 'b', 300, 4)  ship1 = Ship(50, 80, 500000, True, 'w', 50, 10)  plane\_list.append(plane1) plane\_list.append(plane2) plane\_list.append(plane3) plane\_list.append(car1) plane\_list.append(ship1) print("Planes in the list:") plane\_list.extend([plane2, plane3, car1, ship1]) plane\_list.print\_colors() print("Total speed of planes in the list:")  plane\_list.total\_speed() | Total speed of planes in the list: 1800 |  |

**Вывод.**

В ходе выполнения лабораторной работы была написана программа, соответствующая парадигме ООП, состоящая из нескольких классов, наследующихся от друг-друга.

# Приложение А.

**Исходный код программы.**

class Transport():

def init (self, average\_speed, max\_speed, price, cargo,

color): 0):

if not (isinstance(average\_speed, int) and average\_speed > raise ValueError("Invalid value")

if not (isinstance(max\_speed, int) and max\_speed > 0): raise ValueError("Invalid value")

if not (isinstance(price, int) and price > 0): raise ValueError("Invalid value")

if not isinstance(cargo, bool):

raise ValueError("Invalid value")

if not (color == 'w' or color == 'b' or color == 'g'): 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(wheels, int) and 10 >= wheels > 0): raise ValueError("Invalid value")

if not (isinstance(power, int) and power > 0): 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.max\_speed + self.average\_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):

def init (self, average\_speed, max\_speed, price, cargo, color, load\_capacity, wingspan):

super(). init (average\_speed, max\_speed, price, cargo,

color)

0):

if not (isinstance(wingspan, int) and wingspan > 0): raise ValueError("Invalid value")

if not (isinstance(load\_capacity, int) and load\_capacity >

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.max\_speed + self.average\_speed

def eq (self, other):

return self.wingspan == other.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) and length > 0): raise ValueError("Invalid value")

if not (isinstance(side\_height, int) and 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.max\_speed + self.average\_speed

def eq (self, other):

return self.length == other.length and self.side\_height == other.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):

super().extend(list(filter(lambda x: isinstance(x,Plane),iterable)))

def print\_colors(self):

for i in range(len(self)):

print(f"{i+1} самолет: {self[i].color}")

def total\_speed(self): count = 0

for i in range(len(self)):

count += self[i].average\_speed print(count)

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')