**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. 3343 |  | Пухов А. Д. |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2024

## Цель работы

Изучение и применение на практике основ объектно-ориентированного программирования, классов и исключений.

## Задание

​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 ...

## Выполнение работы

В данной лабораторной работе были реализованы базовый класс Transport, подклассы Car, Plane, Ship которые наследуют класс Transport. Так же были реализованы подклассы CarList, PlaneList, ShipList которые наследую родительский класс list. В подклассах Сar, Plane, Ship были переопределены методы: \_\_str\_\_ - выводит всю информацию объекта, \_\_add\_\_ - складывает два атрибута внутри текущего объекта, \_\_eq\_\_ - сравнивает атрибуты двух объектов, если они равны возвращает True иначе Else. Переопределённые методы list для CarList, PlaneList и ShipList будут работать, потому что родительский метод вызывается с помощью функции super().

Код программы смотрите в Приложении А

Базовый класс

Transport

Подклассы

Car

Shape

Plane

Базовый класс

list

Подклассы

CarList

PlaneList

ShapeList

Рисунок 1 – Наследование классов

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | transport = Transport(70, 200, 50000, True, 'w') #транспорт  print(transport.average\_speed, transport.max\_speed, transport.price, transport.cargo, transport.color)    car1 = Car(70, 200, 50000, True, 'w', 100, 4) #авто  car2 = Car(70, 200, 50000, True, 'w', 100, 4)  print(car1.average\_speed, car1.max\_speed, car1.price, car1.cargo, car1.color, car1.power, car1.wheels)  print(car1.\_\_str\_\_())  print(car1.\_\_add\_\_())  print(car1.\_\_eq\_\_(car2))    plane1 = Plane(70, 200, 50000, True, 'w', 1000, 150) #самолет  plane2 = Plane(70, 200, 50000, True, 'w', 1000, 150)  print(plane1.average\_speed, plane1.max\_speed, plane1.price, plane1.cargo, plane1.color, plane1.load\_capacity, plane1.wingspan)  print(plane1.\_\_str\_\_())  print(plane1.\_\_add\_\_())  print(plane1.\_\_eq\_\_(plane2))    ship1 = Ship(70, 200, 50000, True, 'w', 200, 100) #корабль  ship2 = Ship(70, 200, 50000, True, 'w', 200, 100)  print(ship1.average\_speed, ship1.max\_speed, ship1.price, ship1.cargo, ship1.color, ship1.length, ship1.side\_height)  print(ship1.\_\_str\_\_())  print(ship1.\_\_add\_\_())  print(ship1.\_\_eq\_\_(ship2))    car\_list = CarList(Car) #список авто  car\_list.append(car1)  car\_list.append(car2)  car\_list.print\_colors()  car\_list.print\_count()    plane\_list = PlaneList(Plane) #список самолетов  plane\_list.extend([plane1, plane2])  plane\_list.print\_colors()  plane\_list.total\_speed()    ship\_list = ShipList(Ship) #список кораблей  ship\_list.append(ship1)  ship\_list.append(ship2)  ship\_list.print\_colors()  ship\_list.print\_ship() | 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 | ОК |
|  | try: #неправильные данные для самолета      plane1 = Plane(-70, 200, 50000, True, 'w', 1000, 150)  except (TypeError, ValueError):      print('OK')    try:      plane1 = Plane(70, -200, 50000, True, 'w', 1000, 150)  except (TypeError, ValueError):      print('OK')    try:      plane1 = Plane(70, 200, -50000, True, 'w', 1000, 150)  except (TypeError, ValueError):      print('OK')  try:      car1 = Car(70, 200, 50000, -1, 'w', 100, 4)  except (TypeError, ValueError):      print('OK')    try:      car1 = Car(70, 200, 50000, True, -1, 100, 4)  except (TypeError, ValueError):      print('OK')    try:      car1 = Car(70, 200, 50000, True, 'w', -100, 4)  except (TypeError, ValueError):      print('OK')  try:      ship1 = Ship('a', 200, 50000, True, 'w', 200, 100)  except (TypeError, ValueError):      print('OK')    try:      ship1 = Ship(70, 'a', 50000, True, 'w', 200, 100)  except (TypeError, ValueError):      print('OK')    try:      ship1 = Ship(70, 200, 'a', True, 'w', 200, 100)  except (TypeError, ValueError):  print('OK') | OK  OK  OK  OK  OK  OK  OK  OK  OK | ОК |

## Выводы

В данной лабораторной работе были изучены и применены на практике основы объектно-ориентированного программирования.

# Приложение А Исходный код программы

Название файла: lb1.py

**class** Transport:

**def** \_\_init\_\_(self, average\_speed, max\_speed, price, cargo, color):

        self.verify\_value\_transport(average\_speed, max\_speed, price, cargo, color)

        self.average\_speed = average\_speed

        self.max\_speed = max\_speed

        self.price = price

        self.cargo = cargo

        self.color = color

    @classmethod

**def** verify\_value\_transport(cls, average\_speed, max\_speed, price, cargo, color):

        if not isinstance(average\_speed, int) or average\_speed <= 0 or not isinstance(max\_speed, int) or max\_speed <= 0 or not isinstance(price, int) or price <= 0 or not isinstance(cargo, bool) or not color in ('w', 'g', 'b'):

            raise ValueError('Invalid value')

**class** Car(Transport):

**def** \_\_init\_\_(self, average\_speed, max\_speed, price, cargo, color , power , wheels):

        super().\_\_init\_\_(average\_speed, max\_speed, price, cargo, color)

        self.verify\_value\_car(wheels, power)

        self.power = power

        self.wheels = wheels

    @classmethod

**def** verify\_value\_car(cls, wheels, power):

        if not isinstance(power, int) or power <= 0 or not isinstance(wheels,int) or wheels <= 0 or wheels > 10:

            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):

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

        super().\_\_init\_\_(average\_speed, max\_speed, price, cargo, color)

        self.verify\_value\_plane(load\_capacity, wingspan)

        self.load\_capacity = load\_capacity

        self.wingspan = wingspan

    @classmethod

**def** verify\_value\_plane(cls, load\_capacity, wingspan):

        if not isinstance(load\_capacity, int) or load\_capacity <= 0 or not isinstance(wingspan, int) or wingspan <= 0:

            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):

**def** \_\_init\_\_(self, average\_speed, max\_speed, price, cargo, color , length, side\_height):

        super().\_\_init\_\_(average\_speed, max\_speed, price, cargo, color)

        self.verify\_value\_ship(length, side\_height)

        self.length = length

        self.side\_height = side\_height

    @classmethod

**def** verify\_value\_ship(cls, length, side\_height):

        if not isinstance(length, int) or length <= 0 or not isinstance(side\_height, int) or side\_height <= 0:

            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): *#– список автомобилей - наследуется от класса list.*

**def** \_\_init\_\_(self, name):

        super().\_\_init\_\_()

        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):

        print('\n'.join([**f**"{i+1} автомобиль: {self[i].color}" for i in range(len(self))]))

**def** print\_count(self):

        print(len(self))

**class** PlaneList(list):

**def** \_\_init\_\_(self, name):

        super().\_\_init\_\_()

        self.name = name

**def** extend(self, iterable):

        for object in iterable:

            if isinstance(object, Plane):

                super().append(object)

**def** print\_colors(self):

        print('\n'.join([**f**"{i+1} самолет: {self[i].color}" for i in range(len(self))]))

**def** total\_speed(self):

        print(sum([plane.average\_speed for plane in self]))

**class** ShipList(list):

**def** \_\_init\_\_(self, name):

        super().\_\_init\_\_()

        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):

        print('\n'.join([**f**"{i+1} корабль: {self[i].color}" for i in range(len(self))]))

**def** print\_ship(self):

        for i in range(len(self)):

            if self[i].length > 150:

                print(**f**"Длина корабля №{i + 1} больше 150 метров")