**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. 3341 |  | Шаповаленко Е.В |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2024

## Цель работы

Изучить объектно ориентированную парадигму, классы и их иерархию.

Создать иерархию классов для представления различных транспортов и списки для работы с ними. Определить основные атрибуты и методы классов.

## 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 ...

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

Создается класс *Transport*, с атрибутами средняя скорость, максимальная скорость, цена, грузовой, цвет. Проводится проверка, чтобы средняя и максимальная скорости и цена были целыми положительными значениям, параметр грузовой был *True* или *False*, а параметр цвет был *“w”*, *“g”* или *“b”*.

Создается класс *Car*, наследуемый от класса *Transport*. Вызывается конструктор базового класса, задаются атрибуты мощность, количество колес. Оба этих параметра проходят проверку, чтобы они были целыми положительными числами.

Создается класс *Plane*, наследуемый от класса *Transport*. Вызывается конструктор базового класса, задаются атрибуты грузоподъемность, размах крыльев. Оба этих параметра проходят проверку, чтобы они были целыми положительными числами.

Создается класс *Ship*, наследуемый от класса *Transport*. Вызывается конструктор базового класса, задаются атрибуты длина, высота борта. Оба этих параметра проходят проверку, чтобы они были целыми положительными числами.

Создается класс *CarList*, наследуемый от класса *list*, с методами *init*, *append* (можно добавить только объекты класса *Car*), *print\_colors* (выводит цвета всех элементов списка), *print\_count* (выводит количество элементов списка).

Создается класс *PlaneList*, наследуемый от класса *list*, с методами *init*, *extend* (добавятся только объекты класса *Plane*), *print\_colors*(выводит цвета всех элементов списка), *total\_speed* (выводит суммарную среднюю скорость всех элементов списка).

Создается класс *ShipList*, наследуемый от класса *list*, с методами *init*, *append* (можно добавить только объекты класса *Ship*), *print\_colors*(выводит цвета всех элементов списка), *print\_ship* (выводит номера только тех элементов, у которых длина больше 150).

Этот код реализует иерархию транспорта (*Car*, *Plane*, *Ship*), и списков для хранения объектов каждого из классов. Каждый класс транспорта имеет свои уникальные атрибуты.

Изображение иерархии классов:

Character

/ | \

Car Plane Ship

CarList <-- list

PlaneList <-- list

ShipList <-- list

Метод *\_\_init\_\_* переопределен в каждом классе для инициализации атрибутов.

Метод *\_\_str\_\_* переопределен для возвращения строкового представления объекта.

Метод *\_\_eq\_\_* переопределен для сравнения двух объектов одного класса.

Метод *\_\_add\_\_* переопределен для сложения двух объектов одного класса.

Метод *\_\_str\_\_* будет вызываться, когда объект класса будет использоваться в качестве аргумента метода *str()*, чтобы получить его строковое представление.

Метод *\_\_eq\_\_* будет вызываться, когда два объекта класса будут проверяться на равенство (неравенство).

Переопределенные методы из базового класса *list* будут работать для *CarList*, *PlaneList* и *ShipList*, а также и другие, так как все они являются подклассами *list* и наследуют их поведение. Однако было добавлено дополнительное поведение при добавлении элементов в список.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

## Выводы

Была изучена объектно ориентированная парадигма, классы и их иерархия. На практике создана иерархия классов и списки для работы с ними.

# Приложение А Исходный код программы

Название файла: main.py

class Transport:

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

if (not isinstance(average\_speed, int)) or (not isinstance(max\_speed, int)) or (not isinstance(price, int)) or (not isinstance(cargo, bool)) or (not isinstance(color, str)):

raise ValueError("Invalid value")

if average\_speed <= 0 or max\_speed <= 0 or price <= 0:

raise ValueError("Invalid value")

if color not in "wgb":

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(power, int)) or (not isinstance(wheels, int)):

raise ValueError("Invalid value")

if wheels <= 0 or power <= 0 or wheels > 10:

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.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)

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

raise ValueError("Invalid value")

if load\_capacity <= 0 or wingspan <= 0:

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.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)

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

raise ValueError("Invalid value")

if length <= 0 or 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.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):

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):

for element in iterable:

if isinstance(element, Plane):

super().append(element)

def print\_colors(self):

for i in range(len(self)):

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

def total\_speed(self):

speed\_sum = 0

for i in range(len(self)):

speed\_sum += self[i].average\_speed

print(speed\_sum)

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")

# Приложение Б Тестирование

Таблица Б.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 | Тест с e.moefm.info. Проверка базового функционала. |
|  | 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') | OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK | Тест с e.moefm.info. Проверка класса Transport на исключения. |
|  | try: #неправильные данные для авто  car1 = Car(-70, 200, 50000, True, 'w', 100, 4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(70, -200, 50000, True, 'w', 100, 4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(70, 200, -50000, True, 'w', 100, 4)  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:  car1 = Car(70, 200, 50000, True, 'w', 100, -4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(0, 200, 50000, True, 'w', 100, 4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(70, 0, 50000, True, 'w', 100, 4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(70, 200, 0, True, 'w', 100, 4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(70, 200, 50000, 0, 'w', 100, 4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(70, 200, 50000, True, 0, 100, 4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(70, 200, 50000, True, 'w', 0, 4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(70, 200, 50000, True, 'w', 100, 0)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car('a', 200, 50000, True, 'w', 100, 4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(70, 'a', 50000, True, 'w', 100, 4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(70, 200, 'a', True, 'w', 100, 4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(70, 200, 50000, 'a', 'w', 100, 4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(70, 200, 50000, True, 'a', 100, 4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(70, 200, 50000, True, 'w', 'a', 4)  except (TypeError, ValueError):  print('OK')  try:  car1 = Car(70, 200, 50000, True, 'w', 100, 'a')  except (TypeError, ValueError):  print('OK') | OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK | Тест с e.moefm.info. Проверка класса Car на исключения. |
|  | 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:  plane1 = Plane(70, 200, 50000, -1, 'w', 1000, 150)  except (TypeError, ValueError):  print('OK')  try:  plane1 = Plane(70, 200, 50000, True, -1, 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:  plane1 = Plane(0, 200, 50000, True, 'w', 1000, 150)  except (TypeError, ValueError):  print('OK')  try:  plane1 = Plane(70, 0, 50000, True, 'w', 1000, 150)  except (TypeError, ValueError):  print('OK')  try:  plane1 = Plane(70, 200, 0, True, 'w', 1000, 150)  except (TypeError, ValueError):  print('OK')  try:  plane1 = Plane(70, 200, 50000, 0, 'w', 1000, 150)  except (TypeError, ValueError):  print('OK')  try:  plane1 = Plane(70, 200, 50000, True, 0, 1000, 150)  except (TypeError, ValueError):  print('OK')  try:  plane1 = Plane(70, 200, 50000, True, 'w', 0, 150)  except (TypeError, ValueError):  print('OK')  try:  plane1 = Plane(70, 200, 50000, True, 'w', 1000, 0)  except (TypeError, ValueError):  print('OK')  try:  plane1 = Plane('a', 200, 50000, True, 'w', 1000, 150)  except (TypeError, ValueError):  print('OK')  try:  plane1 = Plane(70, 'a', 50000, True, 'w', 1000, 150)  except (TypeError, ValueError):  print('OK')  try:  plane1 = Plane(70, 200, 'a', True, 'w', 1000, 150)  except (TypeError, ValueError):  print('OK')  try:  plane1 = Plane(70, 200, 50000, 'a', 'w', 1000, 150)  except (TypeError, ValueError):  print('OK')  try:  plane1 = Plane(70, 200, 50000, True, 'a', 1000, 150)  except (TypeError, ValueError):  print('OK')  try:  plane1 = Plane(70, 200, 50000, True, 'w', 'a', 150)  except (TypeError, ValueError):  print('OK')  try:  plane1 = Plane(70, 200, 50000, True, 'w', 1000, 'a')  except (TypeError, ValueError):  print('OK') | OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK | Тест с e.moefm.info. Проверка класса Plane на исключения. |
|  | try: #неправильные данные для корабля  ship1 = Ship(-70, 200, 50000, True, 'w', 200, 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, -200, 50000, True, 'w', 200, 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, 200, -50000, True, 'w', 200, 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, 200, 50000, -1, 'w', 200, 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, 200, 50000, True, -1, 200, 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, 200, 50000, True, 'w', -1, 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, 200, 50000, True, 'w', 200, -100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(0, 200, 50000, True, 'w', 200, 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, 0, 50000, True, 'w', 200, 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, 200, 0, True, 'w', 200, 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, 200, 50000, 0, 'w', 200, 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, 200, 50000, True, 0, 200, 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, 200, 50000, True, 'w', 0, 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, 200, 50000, True, 'w', 200, 0)  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')  try:  ship1 = Ship(70, 200, 50000, 'a', 'w', 200, 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, 200, 50000, True, 'a', 200, 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, 200, 50000, True, 'w', 'a', 100)  except (TypeError, ValueError):  print('OK')  try:  ship1 = Ship(70, 200, 50000, True, 'w', 200, 'a')  except (TypeError, ValueError):  print('OK') | OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK | Тест с e.moefm.info. Проверка класса Ship на исключения. |
|  | car1 = Car(70, 200, 50000, True, 'w', 100, 4) #авто  car2 = Car(70, 200, 50000, True, 'w', 100, 4)  plane1 = Plane(70, 200, 50000, True, 'w', 1000, 150) #самолет  plane2 = Plane(70, 200, 50000, True, 'w', 1000, 150)  ship1 = Ship(70, 200, 50000, True, 'w', 200, 100) #корабль  ship2 = Ship(70, 200, 50000, True, 'w', 200, 100)  car\_list = CarList(Car)  plane\_list = PlaneList(Plane)  ship\_list = ShipList(Ship)  try: #проверка списка авто  car\_list.append(plane1)  except (TypeError):  print('OK')  try:  car\_list.append(ship1)  except (TypeError):  print('OK')  try:  car\_list.append([car1, car2])  except (TypeError):  print('OK')  try:  car\_list.append('car1')  except (TypeError):  print('OK')  try:  car\_list.append(1)  except (TypeError):  print('OK')  try: #проверка списка кораблей  plane\_list.extend(plane1)  except (TypeError):  print('OK')  try:  plane\_list.extend(ship1)  except (TypeError):  print('OK')  try:  plane\_list.extend(car1)  except (TypeError):  print('OK')  try:  plane\_list.extend([car1, car2])  except (TypeError):  print('OK')  finally:  if(len(plane\_list)==0):  print('OK2')  try:  plane\_list.extend([ship1, ship2])  except (TypeError):  print('OK')  finally:  if(len(plane\_list)==0):  print('OK2')  try:  plane\_list.extend(['plane1', 'plane2'])  except (TypeError):  print('OK')  finally:  if(len(plane\_list)==0):  print('OK2')  try:  plane\_list.extend([1, 2])  except (TypeError):  print('OK')  finally:  if(len(plane\_list)==0):  print('OK2')  try: #проверка списка кораблей  ship\_list.append(car1)  except (TypeError):  print('OK')  try:  ship\_list.append(plane1)  except (TypeError):  print('OK')  try:  ship\_list.append([ship1, ship2])  except (TypeError):  print('OK')  try:  ship\_list.append('ship1')  except (TypeError):  print('OK')  try:  ship\_list.append(1)  except (TypeError):  print('OK') | OK  OK  OK  OK  OK  OK  OK  OK  OK2  OK2  OK2  OK2  OK  OK  OK  OK  OK | Тест с e.moefm.info. Проверка списков для каждого из классов на исключения. |