**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 гр. 3341 |  | Бойцов В.А. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Целью данной работы заключается в изучении основных парадигм программирования – в особенности, объектно-ориентированного программирования – и написании программы, демонстрирующей иерархию классов и основные принципы ООП, на языке Python.

## 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 ShipL*ist – 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 ...

## Основные теоретические положения

Основные принципы объектно-ориентированного программирования:

* Инкапсуляция: данные, содержащиеся в классе (поля класса), защищены от непосредственного доступа к ним извне класса, и доступ к ним осуществляется через методы класса;
* Наследование: одни классы могут «наследовать» методы и поля других классов, что позволяет создавать иерархию классов;
* Полиморфизм: позволяет обрабатывать объекты разных классов.

Все эти принципы поддерживаются языком Python в той или иной степени.

## Выполнение работы

Для проверки верности переданных в конструктор класса данных используется функция *are\_positive\_ints(variables)*, которая принимает на вход список значений, которые необходимо проверить на отсутствие неположительных или нецелых значений. Если в цикле *for* такие были найдены, функция возвращает 0, иначе 1.

Далее создаётся класс *Transport*, который будет родительским для остальных классов, описывающих транспорт. В этом классе определены поля *average\_speed, max\_speed, price, cargo, color*. В конструкторе класса с помощью функции *are\_positive\_ints()* проверяется соответствие поданным в конструктор значениям необходимым требованиям к данным (цвет *color* и тип *cargo* обрабатываются отдельно). Если данные не соответствуют требованиям, выбрасывается исключение *ValueError*.

Создаётся класс *Car*, наследующийся от *Transport*. В конструкторе класса *Car* вызывается конструктор класса *Transport* для заполнения общих полей, а поля *power* и *wheels* проверяются и заполняются отдельно. В случае, если данные в них не соответствуют требованиям, выбрасывается исключение *ValueError*. К этом же классе созданы методы *\_\_str\_\_(self),* который возвращает форматную строку, содержащую описание всех полей объекта, *\_\_add\_\_(self),* который возвращает сумму полей *average\_speed* и *max\_speed*, и *\_\_eq\_\_(self, other)*, который сравнивает два объекта по полям *wheels, max\_speed, average\_speed* и *power* и возвращает *True*, если все поля одинаковы, иначе – *False*.

Создаётся класс *Plane*, наследующийся от *Transport*. В конструкторе класса *Plane* вызывается конструктор класса *Transport* для заполнения общих полей, а поля *load\_capacity* и *wingspan* проверяются и заполняются отдельно. В случае, если данные в них не соответствуют требованиям, выбрасывается исключение *ValueError*. К этом же классе созданы методы *\_\_str\_\_(self),* который возвращает форматную строку, содержащую описание всех полей объекта, *\_\_add\_\_(self),* который возвращает сумму полей *average\_speed* и *max\_speed*, и *\_\_eq\_\_(self, other),* который сравнивает два объекта по полям *wingspan* и возвращает *True*, если оба поля одинаковы, иначе – *False*.

Создаётся класс *Ship*, наследующийся от *Transport*. В конструкторе класса *Ship* вызывается конструктор класса *Transport* для заполнения общих полей, а поля *length* и *side\_height* проверяются и заполняются отдельно. В случае, если данные в них не соответствуют требованиям, выбрасывается исключение *ValueError*. К этом же классе созданы методы *\_\_str\_\_(self),* который возвращает форматную строку, содержащую описание всех полей объекта, *\_\_add\_\_(self),* который возвращает сумму полей *average\_speed* и *max\_speed*, и *\_\_eq\_\_(self, other),* который сравнивает два объекта по полям *length* и *side\_height* и возвращает *True*, если все поля одинаковы, иначе – *False*.

Создаётся класс *CarList*, наследующийся от *list*. Содержит поле *name*. в конструкторе сначала вызывается конструктор базового класса, а затем полю *name* присваивается переданное конструктору значение. В классе переопределен метод *append(self, p\_object),* который с помощью функции *isinstance()* проверяет *p\_object* на принадлежность классу *Car*, и если это так, то с помощью метода базового класса *append()* добавляет *p\_object* в конец списка, а иначе же выкидывает ошибку *TypeError*; созданы методы *print\_colors(self)*, который форматными строками выводит информацию о цветах всех элементов списка; *print\_count(self)*, который выводит количество элементов в списке.

Создаётся класс *PlaneList*, наследующийся от *list*. Содержит поле *name*. в конструкторе сначала вызывается конструктор базового класса, а затем полю *name* присваивается переданное конструктору значение. В классе переопределен метод *extend(self, iterable),* который с помощью функции *isinstance()* проверяет каждый элемент *iterable* на принадлежность классу *Plane*, и если это так, то с помощью метода базового класса *append()* добавляет элемент в конец списка; созданы методы *print\_colors(self),* который форматными строками выводит информацию о цветах всех элементов списка; *total\_speed(self),* который выводит сумму средних скоростей всех элементов списка.

Создаётся класс *ShipList*, наследующийся от *list*. Содержит поле *name*. в конструкторе сначала вызывается конструктор базового класса, а затем полю *name* присваивается переданное конструктору значение. В классе переопределен метод *append(self, p\_object),* который с помощью функции *isinstance()* проверяет *p\_object* на принадлежность классу *Ship*, и если это так, то с помощью метода базового класса *append()* добавляет *p\_object* в конец списка, а иначе же выкидывает ошибку *TypeError*; созданы методы *print\_colors(self)*, который форматными строками выводит информацию о цветах всех элементов списка; *print\_ship(self),* который для каждого корабля в списке выводит форматную строку с его номером в списке, если длина корабля больше 150 метров.

Выстроенную в программе иерархию классов см на рис. 1.

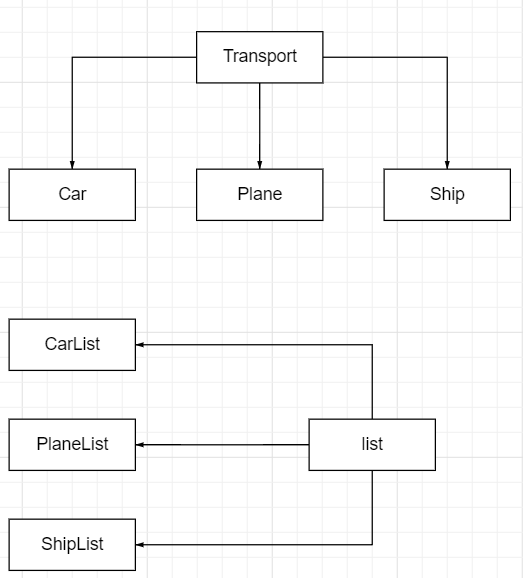


Рисунок 1 – Иерархия классов

Всего (с учётом класса *object*) были переопределены следующие методы: *\_\_init\_\_()* – в каждом классе; *\_\_str\_\_(), \_\_add\_\_(), \_\_eq\_\_()* - в наследниках класса *Transport*; *extend(), append()* – в наследниках *list*.

Метод *\_\_str\_\_()* будет использован тогда, когда потребуется строковое представление объекта класса, например, при вызове функции *print(my\_car),* где *my\_car* – объект класса *Car*. Метод *\_\_eq\_\_()* будет использоваться для операции сравнения (‘*==*’) объектов одного класса.

Для классов *CarList*, *PlaneList* и *ShipList* будут гарантированно корректно работать только те методы класса list, которые были переопределены в его классах-наследниках. Те же методы, которые не были переопределены в наследниках *list*, будут пытаться отработать как обычные методы списка.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

## Выводы

В ходе выполнения работы были изучены основные принципы объектно-ориентированного программирования. Были освоены основные понятия, которыми оперирует ООП. На языке Python была написана программа, реализующая иерархию классов, наследование, переопределение методов родительских классов.

# Приложение А Исходный код программы

Название файла: main.py

def are\_positive\_ints(variables):

for var in variables:

if var<=0 or var%1!=0:

return 0

return 1

class Transport:

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

if (color=='w' or color=='g' or color=='b') and are\_positive\_ints([average\_speed, max\_speed, price]) and (isinstance(cargo, bool)):

self.color = color

self.average\_speed = average\_speed

self.max\_speed = max\_speed

self.price = price

self.cargo=cargo

else:

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)

if(are\_positive\_ints([power, wheels]) and wheels<=10):

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):

if self.wheels==other.wheels and self.max\_speed==other.max\_speed and self.average\_speed==other.average\_speed and self.power==other.power:

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(are\_positive\_ints([load\_capacity, wingspan])):

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):

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 (are\_positive\_ints([length, side\_height])):

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):

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, 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):

super().\_\_init\_\_()

self.name=name

def extend(self, iterable):

for elem in iterable:

if isinstance(elem, Plane):

super().append(elem)

def print\_colors(self):

for i in range(len(self)):

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

def total\_speed(self):

total\_sum=sum(elem.average\_speed for elem in self)

print(total\_sum)

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):

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 - Примеры тестовых случаев

|  |  |  |  |
| --- | --- | --- | --- |
| № 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 | Тестирование созданных классов, их методов (созданных и переопределенных). |
|  | 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  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK | Тестирование при некорректных данных, подаваемых конструкторам классов. |
|  | transport = Transport(70, 200, 50000, True, 'w') #проверка наследства  car1 = Car(70, 200, 50000, True, 'w', 100, 4)  plane1 = Plane(70, 200, 50000, True, 'w', 1000, 150)  ship1 = Ship(70, 200, 50000, True, 'w', 200, 100)  try:  if(isinstance(transport, Transport)):  print('OK')  except:  pass  try:  if(isinstance(car1, Transport)):  print('OK')  except:  pass  try:  if(isinstance(car1, Car)):  print('OK')  except:  pass  try:  if(issubclass(Car, Transport)):  print('OK')  except:  pass  try:  if(isinstance(plane1, Transport)):  print('OK')  except:  pass  try:  if(isinstance(plane1, Plane)):  print('OK')  except:  pass  try:  if(issubclass(Plane, Transport)):  print('OK')  except:  pass  try:  if(isinstance(ship1, Transport)):  print('OK')  except:  pass  try:  if(isinstance(ship1, Ship)):  print('OK')  except:  pass  try:  if(issubclass(Ship, Transport)):  print('OK')  except:  pass  car\_list = CarList(Car)  plane\_list = PlaneList(Plane)  ship\_list = ShipList(Ship)  try:  if(isinstance(car\_list, list)):  print('OK')  except:  pass  try:  if(isinstance(plane\_list, list)):  print('OK')  except:  pass  try:  if(isinstance(ship\_list, list)):  print('OK')  except:  pass | OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK  OK | Тестирование наследования и иерархии классов. |