

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

Пчелкин Н.И.

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*:

средняя 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*:

средняя 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*:

средняя 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*:

средняя 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 ...

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 *positive_int_check(variables)*. Возвращает *True*, если значения положительны и целые, иначе *False*.

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 этому же классу 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 этому же классу 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 этому же классу 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.

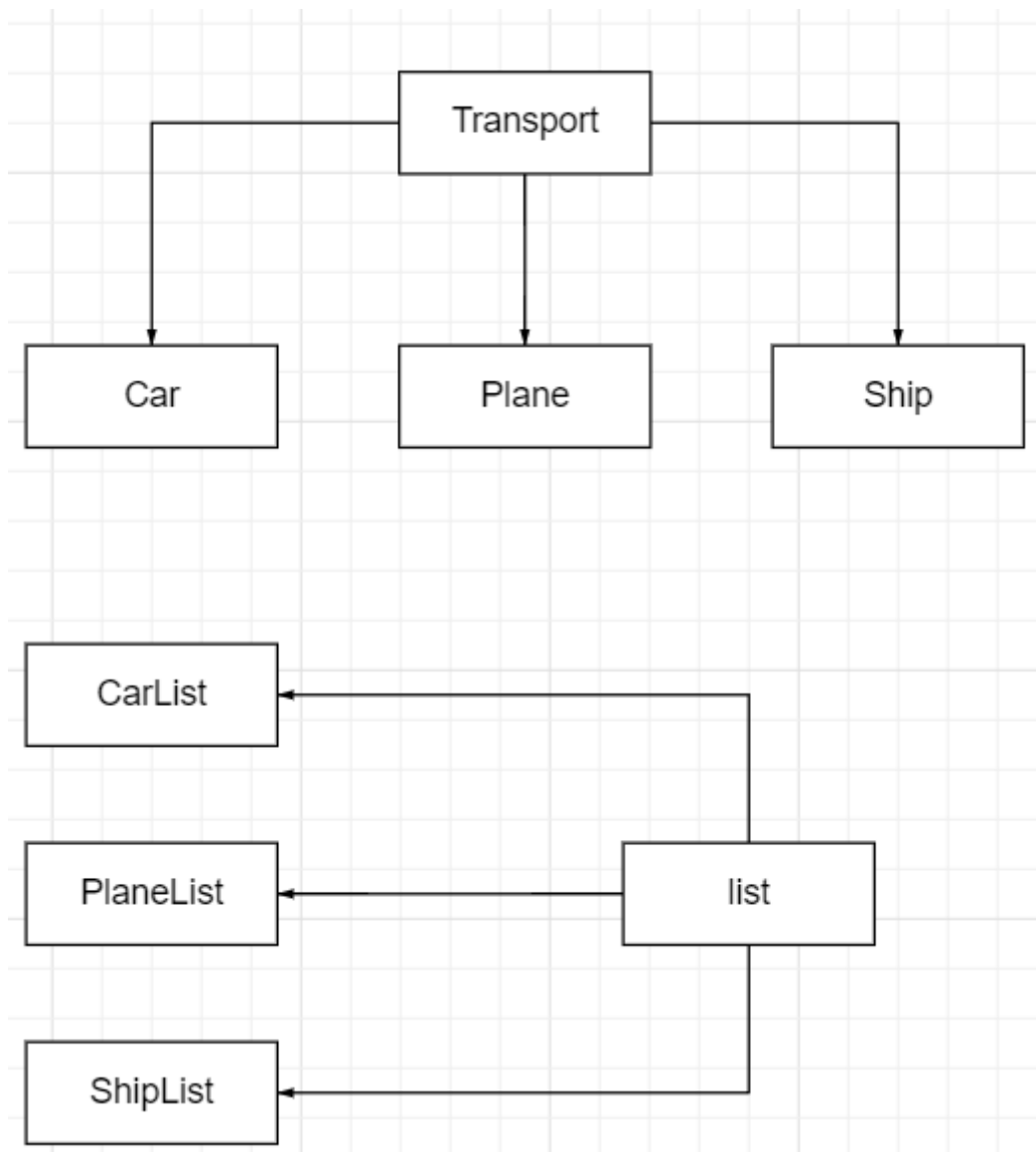


Рисунок 1 – Иерархия классов

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.

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 positive_int_check(variables):
    if all(isinstance(var, int) and (var > 0) for var in variables):
        return True
    return False

class Transport:
    def __init__(self, average_speed, max_speed, price, cargo, color):
        if positive_int_check([average_speed, max_speed, price]) and
type(cargo) == bool and any(color == x for x in ['w', 'g', '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')

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 positive_int_check([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):
        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 positive_int_check([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):
        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 positive_int_check([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):
        return self.side_height == other.side_height and self.length  
== other.length

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):
    result = []
    for element in iterable:
        if isinstance(element, Plane):
            result.append(element)
    super().extend(result)

def print_colors(self):
    for i in range(len(self)):
        print(f'{i+1} самолет: {self[i].color}')

def total_speed(self):
    summary = 0
    for element in self:
        summary += element.average_speed
    print(summary)

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
1.	<pre> 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__()) </pre>	<pre> 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 </pre>	Тестирование созданных классов, их методов (созданных и переопределенных).

<pre> 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() </pre>	<pre> 140 1 корабль: w 2 корабль: w Длина корабля №1 больше 150 метров </pre>
--	---

2.	<pre>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:</pre>	<pre>OK OK</pre>
----	--	---

```

    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):

```



```

    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

```