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 _____ Анисимов Д.А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2024

Цель работы

Основной целью данного проекта было разработать иерархию классов, которая позволила бы представить различные фигуры (многоугольники, окружности) и их списки. Были определены основные атрибуты и методы для каждого класса, а также применены переопределения методов базового класса object для улучшения функциональности и взаимодействия с объектами.

Задание

Даны фигуры в двумерном пространстве.

Базовый класс - фигура Figure:

class Figure:

Поля объекта класса Figure:

периметр фигуры (в сантиметрах, целое положительное число)

площадь фигуры (в квадратных сантиметрах, целое положительное число)

цвет фигуры (значение может быть одной из строк: 'r', 'b', 'g').

При создании экземпляра класса Figure необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'.

Многоугольник - Polygon:

class Polygon: #Наследуется от класса Figure

Поля объекта класса Polygon:

периметр фигуры (в сантиметрах, целое положительное число)

площадь фигуры (в квадратных сантиметрах, целое положительное число)

цвет фигуры (значение может быть одной из строк: 'r', 'b', 'g')

количество углов (неотрицательное значение, больше 2)

равносторонний (значениями могут быть или True, или False)

самый большой угол (или любого угла, если многоугольник равносторонний) (целое положительное число)

При создании экземпляра класса Polygon 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: Polygon: Периметр <периметр>, площадь <площадь>, цвет фигуры <цвет фигуры>, количество углов <кол-во углов>, равносторонний <равносторонний>, самый большой угол <самый большой угол>.

Метод __add__():

Сложение площади и периметра многоугольника. Возвращает число, полученное при сложении площади и периметра многоугольника.

Метод __eq__():

Метод возвращает True, если два объекта класса равны и False иначе. Два объекта типа Polygon равны, если равны их периметры, площади и количество углов.

Окружность - Circle:

class Circle: #Наследуется от класса Figure

Поля объекта класса Circle:

периметр фигуры (в сантиметрах, целое положительное число)

площадь фигуры (в квадратных сантиметрах, целое положительное число)

цвет фигуры (значение может быть одной из строк: 'r', 'b', 'g').

радиус (целое положительное число)

диаметр (целое положительное число, равен двум радиусам)

При создании экземпляра класса Circle 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: Circle: Периметр <периметр>, площадь <площадь>, цвет фигуры <цвет фигуры>, радиус <радиус>, диаметр <диаметр>.

Метод __add__():

Сложение площади и периметра окружности. Возвращает число, полученное при сложении площади и периметра окружности.

Метод __eq__():

Метод возвращает True, если два объекта класса равны и False иначе. Два объекта типа Circle равны, если равны их радиусы.

Необходимо определить список list для работы с фигурами:

Многоугольники:

class PolygonList – список многоугольников - 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 - многоугольник (объект класса Polygon), элемент добавляется в список, иначе выбрасывается исключение TypeError с текстом: Invalid type <тип_объекта p_object>

Метод print_colors(): Вывести цвета всех многоугольников в виде строки (нумерация начинается с 1):

<i> многоугольник: <color[i]>

<j> многоугольник: <color[j]> ...

Метод print_count(): Вывести количество многоугольников в списке.

Окружности:

class CircleList – список окружностей - 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 качестве аргумента передается итерируемый объект iterable, в случае, если элемент iterable - объект класса Circle, этот элемент добавляется в список, иначе не добавляется.

Метод print_colors(): Вывести цвета всех окружностей в виде строки (нумерация начинается с 1):

<i> окружность: <color[i]>

<j> окружность: <color[j]> ...

Метод total_area(): Посчитать и вывести общую площадь всех окружностей.

В отчете укажите:

1. Изображение иерархии описанных вами классов.
2. Методы, которые вы переопределили (в том числе методы класса object).
3. В каких случаях будут использованы методы __str__() и __add__().

4. Будут ли работать переопределенные методы класса list для PolygonList и CircleList? Объясните почему и приведите примеры.

Основные теоретические положения

1. Python представляет данные и функции в виде объектов. Объекты могут быть созданы как экземпляры классов или динамически.

2. Классы в Python служат для создания объектов по заданным шаблонам. Они содержат как атрибуты (информацию), так и методы (действия).

3. В Python используется наследование, которое позволяет классу наследовать характеристики другого класса. Это упрощает создание иерархий и повторное использование кода.

4. Инкапсуляция в Python делает данные класса защищеными и доступ к ним осуществляется через методы класса, а не напрямую извне.

5. Python поддерживает полиморфизм, позволяя объектам с одинаковым интерфейсом иметь разные реализации. Это позволяет использовать различные классы объектов с одним интерфейсом в процессе выполнения программы.

6. Методы в Python бывают экземплярными (привязанными к экземпляру объекта) или статическими (привязанными к классу). Статические методы не имеют доступа к экземплярным данным и используются для управления классом в целом.

Выполнение работы

1. Создание класса "figure":

-Определение метода __init__(), который принимает параметры "perimeter", "area" и "color".

-Проверка на корректность переданных значений: периметр и площадь должны быть положительными целыми числами, цвет должен быть одним из ['r', 'b', 'g'].

-Если значения некорректны, вызывается исключение ValueError.

-Запись переданных значений в атрибуты объекта perimeter, area и color.

2. Создание класса "polygon", который наследуется от "figure":

-Определение метода __init__(), который принимает дополнительные параметры "angle_count", "equilateral" и "biggest_angle".

-Вызов метода __init__() родительского класса с передачей параметров "perimeter", "area" и "color".

-Проверка на корректность дополнительных переданных значений: angle_count должен быть положительным целым числом больше 2, equilateral - булевым значением, biggest_angle - положительным целым числом.

-Если значения некорректны, вызывается исключение ValueError.

-Запись переданных значений в атрибуты объекта angle_count, equilateral и biggest_angle.

3. Определение метода __str__() для класса "polygon":

-Определение возвращаемого форматированного строки, содержащей значения атрибутов объекта.

4. Определение метода __add__() для класса "polygon":

-Определение возвращаемого значения, равного сумме периметра и площади объекта.

5. Определение метода __eq__() для класса "polygon":

-Проверка, является ли переданный объект экземпляром класса "polygon".

-Если да, сравнение значений периметра, площади и angle_count текущего объекта с переданным объектом.

-Если значения совпадают, возвращается True, иначе - False.

6. Создание класса "circle", который наследуется от "figure":

-Определение метода __init__(), который принимает дополнительные параметры "radius" и "diametr".

-Вызов метода __init__() родительского класса с передачей параметров "perimeter", "area" и "color".

-Проверка на корректность дополнительных переданных значений: radius должен быть положительным целым числом, diametr - положительным целым числом, равным удвоенному значению radius.

-Если значения некорректны, вызывается исключение ValueError.

-Запись переданных значений в атрибуты объекта radius и diametr.

7. Определение метода __str__() для класса "circle":

-Определение возвращаемого форматированного строки, содержащей значения атрибутов объекта.

8. Определение метода __add__() для класса "circle":

-Определение возвращаемого значения, равного сумме периметра и площади объекта.

9. Определение метода __eq__() для класса "circle":

-Проверка, является ли переданный объект экземпляром класса "circle".

-Если да, сравнение значений радиуса текущего объекта с переданным объектом.

-Если значения совпадают, возвращается True, иначе - False.

10. Создание класса "polygonlist", который наследуется от встроенного класса "list":

-Определение метода __init__(), который принимает параметр "name".

-Вызов метода __init__() родительского класса.

-Запись переданного значения в атрибут объекта name.

11. Определение метода append() для класса "polygonlist":

-Проверка, является ли переданный объект экземпляром класса "polygon".

-Если да, вызов метода append() родительского класса с переданной подходящей фигурой, иначе - вызов исключения TypeError.

12. Определение метода print_colors() для класса "polygonlist":

-Создание переменной result, в которую будут добавляться строки с информацией о цвете каждого объекта "polygon".

-Итерация по объектам "polygon" в списке и добавление строки с номером многоугольника и его цветом в переменную result.

-Если номер многоугольника равен 1, добавление символа новой строки в конец строки result.

-Вывод на экран значения переменной result.

13. Определение метода print_count() для класса "polygonlist":

-Вывод на экран значения, равного количеству элементов в списке.

14. Создание класса "circlelist", который наследуется от встроенного класса "list":

-Определение метода __init__(), который принимает параметр "name".

-Вызов метода __init__() родительского класса.

-Запись переданного значения в атрибут объекта name.

15. Определение метода extend() для класса "circlelist":

-Проверка, является ли каждый элемент переданного итерируемого объекта экземпляром класса "circle".

-Если да, добавление элемента в список с помощью метода append() родительского класса.

-Если нет, элемент игнорируется.

16. Определение метода print_colors() для класса "circlelist":

-Создание переменной result, в которую будут добавляться строки с информацией о цвете каждой окружности.

-Итерация по объектам "circle" в списке и добавление строки с номером окружности и ее цветом в переменную result.

-Если номер окружности равен 1, добавление символа новой строки в конец строки result.

-Вывод на экран значения переменной result.

17. Определение метода total_area() для класса "circlelist":

-Создание переменной total_area, в которую будут суммироваться значения площадей каждой окружности.

-Итерация по объектам "circle" в списке и добавление значения площади каждой окружности к переменной total_area.

-Вывод на экран значения переменной total_area.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии

1.	fig = Figure(10,25,'g') #фигура
	print(fig.perimeter, fig.area, fig.color)
	
	polygon = Polygon(10,25,'g',4, True, 90) #многоугольник
	polygon2 = Polygon(10,25,'g',4, True, 90)
	print(polygon.perimeter, polygon.area, polygon.color, polygon.angle_count, polygon.equilateral, polygon.biggest_angle)
	print(polygon.__str__())
	print(polygon.__add__())
	print(polygon.__eq__(polygon2))
	
	circle = Circle(13, 13,'r', 2, 4) #окружность
	circle2 = Circle(13, 13,'g', 2, 4)
	print(circle.perimeter, circle.area, circle.color,
		10 25 g
		10 25 g 4 True 90
		Polygon: Периметр 10, площадь 25, цвет фигуры g, количество углов 4, равносторонний True, самый большой угол 90.
		35
		True
		13 13 r 2 4
		Circle: Периметр 13, площадь 13, цвет фигуры r, радиус 2, диаметр 4.
		26
		True
		1 многоугольник: g
		2 многоугольник: g
		2
		1 окружность: r
		2 окружность: g
		26	

	circle.radius, circle.diametr) print(circle.__str__()) print(circle.__add__()) print(circle.__eq__(ci rcle2)) polygon_list = PolygonList(Polygon) #список многоугольников polygon_list.append(polygon) polygon_list.append(polygon2) polygon_list.print_col ors() polygon_list.print_co unt() circle_list = CircleList(Circle) #список окружностей circle_list.extend([cir cle, circle2]) circle_list.print_color s() circle_list.total_area()		
2.	try: #неправильные данные для фигуры fig = Figure(- 10,25,'g') except (TypeError, ValueError): print('OK')	OK OK OK OK OK OK OK	

	try: fig = Figure(10,-25,'g') except (TypeError, ValueError): print('OK') try: fig = Figure(10,25,-1) except (TypeError, ValueError): print('OK') try: fig = Figure(10,25,1) except (TypeError, ValueError): print('OK') try: fig = Figure(10,25,'a') except (TypeError, ValueError): print('OK') try: fig = Figure('a',25,'g') except (TypeError, ValueError):	OK OK	

```
        print('OK')


            try:
             fig =
Figure(10,'a','g')
    except (TypeError,
  ValueError):
        print('OK')


            try:
             fig =
Figure(0,25,'g')
    except (TypeError,
  ValueError):
        print('OK')


            try:
             fig =
Figure(10,0,'g')
    except (TypeError,
  ValueError):
      print('OK')
```

Выводы

В ходе анализа иерархии классов мы исследовали способы использования наследования для создания классов с общими характеристиками, при этом обеспечивая уникальные особенности и методы для каждого класса. Мы также изучили процесс переопределения методов базового класса object с целью улучшения работы с объектами и их строковым отображением.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Figure:
    def __init__(self, perimeter, area, color):
        if type(perimeter) != int or type(area) != int or perimeter
<= 0 or area <= 0 or color not in ['r', 'b', 'g']:
            raise ValueError('Invalid value')
        self.perimeter = perimeter
        self.area = area
        self.color = color

class Polygon(Figure):
    def __init__(self, perimeter, area, color, angle_count,
equilateral, biggest_angle):
        super().__init__(perimeter, area, color)
        if type(angle_count) != int or angle_count <= 2 or
type(equilateral) != bool or type(biggest_angle) != int or biggest_angle
<= 0:
            raise ValueError('Invalid value')
        self.angle_count = angle_count
        self.equilateral = equilateral
        self.biggest_angle = biggest_angle

    def __str__(self):
        return f"Polygon: Периметр {self.perimeter}, площадь {self.area}, цвет фигуры {self.color}, количество углов {self.angle_count}, равносторонний {self.equilateral}, самый большой угол {self.biggest_angle}."

    def __add__(self):
        return self.perimeter + self.area

    def __eq__(self, other):
        if isinstance(other, Polygon):
            return self.perimeter == other.perimeter and self.area
== other.area and self.angle_count == other.angle_count
        return False


class Circle(Figure):
    def __init__(self, perimeter, area, color, radius, diametr):
        super().__init__(perimeter, area, color)
        if type(radius) != int or radius <= 0 or type(diametr) != int
or diametr <= 0 or diametr != 2 * radius:
            raise ValueError('Invalid value')
        self.radius = radius
        self.diametr = diametr

    def __str__(self):
```

```
            return f"Circle: П е р и м е т р {self.perimeter}, п л о щ а д ь {self.area}, ц в е т  ф и г у р ы {self.color}, р а д и у с {self.radius}, д и а м е т р {self.diametr}."

        def __add__(self):
            return self.perimeter + self.area

        def __eq__(self, other):
            if isinstance(other, Circle):
                return self.radius == other.radius
            return False


    class PolygonList(list):
        def __init__(self, name):
            super().__init__()
            self.name = name

        def append(self, p_object):
            if isinstance(p_object, Polygon):
                super().append(p_object)
            else:
                raise TypeError(f"Invalid type {type(p_object)}")

        def print_colors(self):
            result = ""
            for i, p in enumerate(self, start=1):
                result += f"{i} м н о г о у г о л ь н и к: {p.color}"
                if i==1:
                    result +="\n"
            print(result)

        def print_count(self):
            print(len(self))


    class CircleList(list):
        def __init__(self, name):
            super().__init__()
            self.name = name

        def extend(self, iterable):
            for item in iterable:
                if isinstance(item, Circle):
                    super().append(item)

        def print_colors(self):
            result = ""
            for i, c in enumerate(self, start=1):
                result += f"{i} о к р у ж н о с т ь: {c.color}"
                if i==1:
                    result +="\n"
            print(result)

        def total_area(self):
            total_area = 0
            for c in self:
```

```
    total_area += c.area
print(total_area)
```