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 _____ Ягудин Д. Р.

Преподаватель _____ Иванов Д. В.

Санкт-Петербург

2024

Цель работы

Цель этой работы была создание иерархии классов для представления различных геометрических фигур (многоугольников, окружностей) и их списков. Определили основные атрибуты и методы для каждого класса, а также переопределили методы 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.

Выполнение работы

1. Создаем базовый класс Figure, который содержит атрибуты perimeter, area, color. При инициализации объекта проверяем на корректность введенные данные.

2. Создаем класс Polygon, который наследуется от класса Figure и добавляет атрибуты angle_count, equilateral, biggest_angle. При инициализации объекта вызываем init родительского класса, затем проверяем biggest_angle, angle_count и equilateral на корректность.

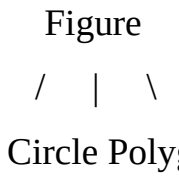
3. Создаем класс Circle, который также наследуется от класса Figure и добавляет атрибуты radius, diametr. При инициализации объекта вызываем init родительского класса, затем проверяем radius, diametr на корректность.

4. Создаем класс PolygonList, который наследуется от списка и добавляет методы init, append, print_count, print_color. Метод append позволяет добавлять только объекты класса Polygon в список.

6. Создаем класс CircleList, аналогично PolygonList, добавляет методы init, extend, total_area, print_colors. Метод extend позволяет добавлять в список только объекты класса Circle.

Этот код реализует иерархию классов персонажей (воин, маг, лучник) и списков для хранения персонажей каждого класса. Каждый класс персонажа имеет свои уникальные атрибуты и методы.

1. Изображение иерархии классов:



PolygonList <-- list

CircleList <-- list

2. В переопределении методов класса объекта object или других методов:

- Метод `__init__`: переопределен в каждом классе для инициализации атрибутов.

- Метод `__str__`: переопределен для возвращения строкового представления объекта.

3. Метод `__add__` в классе Polygon и в классе Circle переопределен таким образом, что значение area прибавляется к значению perimeter. Метод `__eq__` проверяет равенство атрибута area у двух объектов данного класса Polygon и атрибута radius у двух объектов класса Circle.

4. Переопределенные методы класса list для созданных списков не будут работать, т.к. созданные классы WarriorList, MagicianList и ArcherList являются подклассами списка (list), но они не переопределяют поведение всех методов класса list. Например, методы append, extend, print_count могут быть вызваны нормально, так как они определены в наших классах, но такие методы, как clear или pop, которые не переопределены, будут работать как обычно для списка без дополнительной логики, определенной в наших классах.

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
		Получено
		
		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))
			Проверка работы с корректными данными

	circle2 = Circle(13, 13,'g', 2, 4)
	print(circle.perimeter, circle.area, circle.color, circle.radius, circle.diametr)
	print(circle.__str__())
	print(circle.__add__())
	print(circle.__eq__(circle2))
	
	polygon_list = PolygonList(Polygon) #список многоугольников
	polygon_list.append(polygon)
	polygon_list.append(polygon2)
	polygon_list.print_colors()
	polygon_list.print_count()
	
	circle_list = CircleList(Circle) #список окружностей
	circle_list.extend([circle, circle2])
	circle_list.print_colors()
		circle = Circle(13, 13,'r', 2, 4) #окружность
		circle2 = Circle(13, 13,'g', 2, 4)
		print(circle.perimeter, circle.area, circle.color, circle.radius, circle.diametr)
		print(circle.__str__())
		print(circle.__add__())
		print(circle.__eq__(circle2))
		
		polygon_list = PolygonList(Polygon) #список многоугольников
		polygon_list.append(polygon)
		polygon_list.append(polygon2)
		polygon_list.print_colors()
		polygon_list.print_count()
		
		circle_list = CircleList(Circle) #список окружностей	

	circle_list.total_area()	circle_list.extend([circle, circle2]) circle_list.print_colors() circle_list.total_area() 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	

2.	try: #неправильные данные для фигуры
	fig = Figure(-10,25,'g')
	except (TypeError, ValueError):
	print('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')
		OK
		OK
		OK
		OK
		OK
		OK
		OK
		OK
		OK
			Проверка работы с некорректными данными

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

Изучив иерархию классов, поняли, как использовать наследование для создания классов с общими характеристиками, поддерживая при этом уникальные особенности и методы для каждого класса. Также рассмотрели, как переопределить методы базового класса object для более удобной работы с объектами и их строковым представлением.

Также установили, что переопределенные методы класса list для созданных подклассов не будут работать для всех методов списка, поскольку классы CircleList, PolygonList наследуют методы класса list, но не переопределяют все их поведения.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb1.py

```
class Figure:
    def __init__(self, perimeter, area, color):
        try:
            color_type = ['r', 'g', 'b']
              if (perimeter <= 0) or (area <= 0) or (color not in
color_type):
                  raise ValueError
            else:
                self.perimeter = perimeter
                self.area = area
                self.color = color
        except ValueError:
            raise ValueError

class Polygon(Figure):
        def __init__(self, perimeter, area, color, angle_count,
equilateral, biggest_angle):
        try:
            color_type = ['r', 'g', 'b']
              if (perimeter <= 0) or (area <= 0) or (color not in
color_type) or (angle_count <= 2) or (type(equilateral) != bool) or
(biggest_angle <= 0):
                  raise ValueError
            else:
                self.perimeter = perimeter
                self.area = area
                self.color = color
                self.angle_count = angle_count
                self.equilateral = equilateral
                self.biggest_angle = biggest_angle
        except ValueError:
            raise ValueError

    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.area + self.perimeter

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

class Circle(Figure):
    def __init__(self, perimeter, area, color, radius, diametr):
```

```
            try:
                color_type = ['r', 'g', 'b']
                  if (perimeter <= 0) or (area <= 0) or (color not in
color_type) or (radius <= 0) or (diametr <= 0 or diametr != (radius *
2)):
                    raise ValueError
                else:
                    self.perimeter = perimeter
                    self.area = area
                    self.color = color
                    self.radius = radius
                    self.diametr = diametr
            except ValueError:
                raise ValueError

        def __str__(self):
                 return f"Circle: Периметр {self.perimeter}, площадь
{self.area}, цвет фигуры {self.color}, радиус {self.radius}, диаметр
{self.diametr}."

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

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

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

        def append(self, p_object):
            try:
                if type(p_object) != Polygon:
                    raise TypeError
                else:
                    super().append(p_object)

            except TypeError:
                raise TypeError(f"Invalid type {type(p_object)}")

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

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

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

        def extend(self, iterable):
            if hasattr(iterable, "__iter__"):
                for i in range(len(iterable)):
```

```
            if type(iterable[i]) == Circle:
                super().append(iterable[i])
    else:
        raise TypeError

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

def total_area(self):
    sum = 0
    for i in range(len(self)):
        sum += self[i].area
    print(sum)
```