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

Студентка гр. 3344 _____ Коняева М.В.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2024

Цель работы

Целью работы является изучение основ парадигм ООП в языке программирования Python.

Задание

Вариант 1. Даны фигуры в двумерном пространстве.

Базовый класс - фигура *Figure*:

class Figure:

Поля объекта класса *Figure*:

1) периметр фигуры (в сантиметрах, целое положительное число)
2) площадь фигуры (в квадратных сантиметрах, целое положительное число)
3) 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*:

1) периметр фигуры (в сантиметрах, целое положительное число)
2) площадь фигуры (в квадратных сантиметрах, целое положительное число)
3) цвет фигуры (значение может быть одной из строк: '*r'*, '*b'*, *'g'*)
4) количество углов (неотрицательное значение, больше 2)
5) равносторонний (значениями могут быть или *True*, или *False*)
6) 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*:

1) периметр фигуры (в сантиметрах, целое положительное число)
2) площадь фигуры (в квадратных сантиметрах, целое положительное число)
3) цвет фигуры (значение может быть одной из строк: '*r*', '*b*', '*g*').
4) радиус (целое положительное число)
5) 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:

1) Вызвать конструктор базового класса.
2) 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:

1) Вызвать конструктор базового класса.
2) 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. Иерархия классов представлена на рисунке 1.

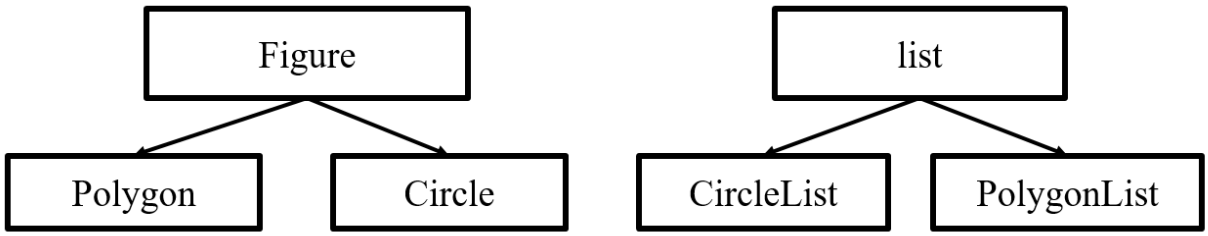

Рисунок 1 – Иерархия классов

2. Переопределенные методы:

   __init__() – Метод, который переопределен для всех классов. Инициализирует поля класса.

   add() – Метод, который переопределен у классов *Figure*. Возвращает число, полученное при сложении двух полученных на вход чисел.

   __str__() – Метод, который используется для строчного представления объекта.

   __eq__() – Метод для сравнения двух фигур класса.

   append()– Метод, который переопределен у класса *PolygonList*. Если объект класса *Polygon*, то элемент добавляется в список.

   extend() – Метод, который переопределен у класса *CircleList*. Если объект списка класса *Circle*, то элемент добавляется в список.

3. Метод *__str__()* используется для отображения информации об объекте в удобочитаемом формате.

   Метод *add()* используется для получения суммы двух объектов.

4. Переопределенные методы класса *list* для *PolygonList* и *CircleList* будут работать, потому что эти классы являются наследниками класса *list*. То есть при переопределении методов свойства сохранятся, изменится только нужная нам логика.

Разработанный программный код см. в приложении А. Результаты тестирования см. в приложении Б.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Тест	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, circle.radius, circle.diametr)
	print(circle.__str__())
	printcircle.__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_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
			Данные обработаны корректно

2.	try: #неправильные данные для фигуры
	fig = Figure(-10,25,'g')
	except (TypeError, ValueError):
	    print('OK')
	
	try:
	    fig = Figure(10,-25,'g')
		OK
		OK
		OK
		OK
		OK
		OK
		OK
		OK
		OK
			Данные обработаны корректно

```
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):
    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

Были получены базовые навыки работы с объектно-ориентированным программированием. Была написана программа, с помощью которой были изучены наследование классов и переопределение методов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb1.py

```
class Figure:
    def __init__(self, perimeter, area, color):
        if not (
            isinstance(perimeter, int)
            and isinstance(area, int)
            and isinstance(color, str)
            and area > 0
            and perimeter > 0
            and color in "rgb"
        ):
            raise ValueError("Invalid value")
        self.perimeter = perimeter
        self.area = area
        self.color = color

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

class Polygon(Figure):  # Наследуется от класса Figure
    def __init__(self, perimeter, area, color, angle_count, equilateral,
biggest_angle):
        super().__init__(perimeter, area, color)
        if not (
            isinstance(angle_count, int)
            and isinstance(equilateral, bool)
            and isinstance(biggest_angle, int)
            and biggest_angle > 0
            and angle_count > 2
        ):
            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 __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):  # Наследуется от класса Figure
    def __init__(self, perimeter, area, color, radius, diametr):
        super().__init__(perimeter, area, color)
        if not (
                isinstance(radius, int)
```

```
                and isinstance(diametr, int)
                and radius > 0
                and diametr > 0
                and 2 * radius == diametr
        ):
            raise ValueError("Invalid value")
        self.radius = radius
        self.diametr = diametr

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

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

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

    def append(self, p_object):
        if not isinstance(p_object, Polygon):
            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} 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):
        for i in iterable:
            if isinstance(i, Circle):
                super().append(i)

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

    def total_area(self):
        result = 0
        for i in self:
            result+= i.area
        print(result)
```