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. 3342 _____ Хайруллов Д.Л.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2024

Цель работы

Изучение парадигм программирования и основ работы с классами в языке программирования Python.

Задание

Вариант 4.

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

Описание созданных классов:

Класс Figure:

Поля:

perimeter - целое число, периметр фигуры

area - целое число, площадь фигуры

color - строка, цвет фигуры

Методы:

init(self, perimeter, area, color): конструктор класса, принимает значения периметра, площади и цвета фигуры и проверяет их на корректность

Класс Polygon (наследует от Figure):

Поля:

perimeter - целое число, периметр фигуры

area - целое число, площадь фигуры

color - строка, цвет фигуры

angle_count - целое число, количество углов в многоугольнике

equilateral - булево значение, является ли многоугольник равносторонним

biggest_angle - целое число, значение самого большого угла в многоугольнике

Методы:

init(self, perimeter, area, color, angle_count, equilateral, biggest_angle): конструктор класса, вызывает конструктор родительского класса и проверяет корректность значений переданных параметров

str(self): возвращает строку с описанием свойств многоугольника

add(self): возвращает сумму периметра и площади многоугольника

eq(self, other): сравнивает текущий многоугольник с другим по периметру, площади и количеству углов

Класс Circle (наследует от Figure):

Поля:

perimeter - целое число, периметр фигуры

area - целое число, площадь фигуры

color - строка, цвет фигуры

radius - целое число, радиус окружности

diametr - целое число, диаметр окружности

Методы:

init(self, perimeter, area, color, radius, diametr): конструктор класса, вызывает конструктор родительского класса и проверяет корректность значений переданных параметров

str(self): возвращает строку с описанием свойств окружности

add(self): возвращает сумму периметра и площади окружности

eq(self, other): сравнивает текущую окружность с другой по радиусу

Класс PolygonList (наследует от list):

Поля:

name - строка, имя списка многоугольников

Методы:

init(self, name): конструктор класса, вызывает конструктор списка и устанавливает имя списка

append(self, p_object): добавляет объект (многоугольник) в список, если он принадлежит классу Polygon

print_colors(self): выводит цвета всех многоугольников из списка

print_count(self): выводит количество многоугольников в списке

Класс CircleList (наследует от list):

Поля:

name - строка, имя списка окружностей

Методы:

init(self, name): конструктор класса, вызывает конструктор списка и устанавливает имя списка

extend(self, iterable): добавляет все объекты из переданного итерируемого объекта (список окружностей), если они принадлежат классу Circle

print_colors(self): выводит цвета всех окружностей из списка

total_area(self): выводит сумму площадей всех окружностей из списка

Иерархия описанных классов представлена на рисунке 1.

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные

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__())
	print(circle.__add__())
	print(circle.__eq__(circle2))
		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

	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()	

Выводы

Были изучены парадигмы программирования и освоены основы работы с классами в языке программирования Python. Была написана программа с реализацией необходимых классов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Figure:
    perimeter = 0
    area = 0
    color = ''

    def __init__(self, perimeter, area, color):
        if not(isinstance(perimeter, int) and isinstance(area, int) and
color in ['r', 'g', 'b'] and perimeter >0 and area > 0):
            raise ValueError('Invalid value')
        self.perimeter = perimeter
        self.area = area
        self.color = color


class Polygon(Figure):
    perimeter = 0
    area = 0
    color = ''
    angle_count = 0
    equilateral = True
    biggest_angle = 0

    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 angle_count > 2 and
biggest_angle > 0):
            raise ValueError('Invalid value')
        self.angle_count = angle_count
        self.equilateral = equilateral
        self.biggest_angle = biggest_angle

    def __str__(self):
        return "Polygon: Периметр " + str(self.perimeter) + ", площадь "
+ str(self.area) + ", цвет фигуры " + self.color + ", количество углов " +
str(self.angle_count) + ", равносторонний " + str(self.equilateral) + ", с
амый большой угол " + str(self.biggest_angle) + "."

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

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


class Circle(Figure):
    perimeter = 0
```

```
    area = 0
    color = ''
    radius = 0
    diametr = 0

    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 diametr == radius*2):
            raise ValueError('Invalid value')
        self.radius = radius
        self.diametr = diametr

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

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

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


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("Invalid type " + str(type(p_object)))


    def print_colors(self):
        for index in range(0, len(self)):
            print(str(index + 1) + " многоугольник: " + self[index].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 item in iterable:
```

```
        if isinstance(item, Circle):
            super().append(item)


def print_colors(self):
    for index in range(0, len(self)):
        print(str(index + 1) + " окружность: " + self[index].color)


def total_area(self):
    answer = 0
    for item in self:
        answer+=item.area
    print(answer)
```