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. 3344 _____ Фоминых Е.Г.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2024

Цель работы

Изучить применение объектно-ориентированного программирования на языке Python.

Задание.

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. Иерархия классов:

1. Figure
   1.1. Polygon
   1.2. Circle
2. list
   2.1. PolygonList
   2.2. CircleList

2. Переопределенные методы:

__init__() - метод для инициализации класса, который был переопределен для всех классов.

__add__() - метод, переопределенный в классе Figure, который выполняется при попытке сложить один объект с другим.

__str__() - метод, который используется для строчного представления объекта.

__eq__() - метод, который используется при попытке сравнения объектов.

3. В каких случаях будут использованы методы __str__() и __add__():

Метод __str__() будет вызван при попытке преобразовать объект в строку, чтобы получить его текстовое представление.

Метод __add__() будет вызван при использовании оператора + с двумя объектами, позволяя определить, каков результат сложения обьектов.

4. Будут ли работать переопределенные методы класса list для PolygonList и CircleList:

Переопределенные методы класса list будут работать, т. к. для нахождения метода по имени программа поднимается снизу-вверх по иерархии

классов, то есть сначала ищет переопределённые методы в самом экземпляре, не найдя - переходит к родительским классам. Например, методы append и extend будут работать аналогично методам родительского класса list, но при этом будут проверять тип объектов, которые добавляются в список.

Выводы

В ходе работы были приобретены навыки работы с объектно-ориентированным программированием и изучены особенности переопределения методов классов и обработки ошибок в Python.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: src.py

```
class Figure:
    '''Поля объекта класс Figure:
    perimeter - периметр фигуры (в сантиметрах, целое положительное число)
    area - площадь фигуры (в квадратных сантиметрах, целое положительное число)
    color - 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'.
    '''
    def __init__(self,perimeter,area,color):
        if not(isinstance(perimeter,int) and perimeter>0):
            raise ValueError('Invalid value')
        if not(isinstance(area,int) and area>0):
            raise ValueError('Invalid value')
        if color not in ['r', 'b', 'g']:
            raise ValueError('Invalid value')

        self.perimeter = perimeter
        self.area = area
        self.color = color


class Polygon(Figure): #Наследуется от класса Figure
    '''Поля объекта класс Polygon:
    perimeter - периметр фигуры (в сантиметрах, целое положительное число)
    area - площадь фигуры (в квадратных сантиметрах, целое положительное число)
    color - цвет фигуры (значение может быть одной из строк: 'r', 'b', 'g')
    angle_count - количество углов (целое положительное значение, больше 2)
```

equilateral - равносторонний (значениями могут быть или True, или False)

biggest_angle - самый большой угол (или любой угол, если многоугольник равносторонний) (в градусах, 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'.

'''

```
    def __init__(self,perimeter,area,color,angle_count,equilateral,biggest_angle):
        if not(isinstance(perimeter,int) and perimeter>0):
            raise ValueError('Invalid value')
        if not(isinstance(area,int) and area>0):
            raise ValueError('Invalid value')
        if color not in ['r', 'b', 'g']:
            raise ValueError('Invalid value')
        if not(isinstance(angle_count,int) and angle_count>2):
            raise ValueError('Invalid value')
        if not isinstance(equilateral, bool):
            raise ValueError('Invalid value')
        if not(isinstance(biggest_angle,int) and biggest_angle>0):
            raise ValueError('Invalid value')

        self.perimeter = perimeter
        self.area = area
        self.angle_count = angle_count
        self.color = color
        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}."
```

'''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>.'''

    def __add__(self):
        return self.perimeter+self.area
    '''Сложение площади и периметра многоугольника. Возвращает число, полученное при сложении площади и периметра многоугольника.'''

    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
    '''Метод возвращает True, если два объекта класса равны и False иначе. Два объекта типа Polygon равны, если равны их периметр, площадь и количество углов.'''

class Circle(Figure): #Наследуется от класса Figure
    '''Поля объекта класс Circle:
    perimeter - периметр фигуры (в сантиметрах, целое положительное число)
    area - площадь фигуры (в квадратных сантиметрах, целое положительное число)
    color - цвет фигуры (значение может быть одной из строк: 'r', 'b', 'g')
    radius - радиус (целое положительное число)
    diametr - 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'.
    '''
    def __init__(self,perimeter,area,color,radius,diametr):
        if not(isinstance(perimeter,int) and perimeter>0):
            raise ValueError('Invalid value')
        if not(isinstance(area,int) and area>0):
            raise ValueError('Invalid value')
        if color not in ['r', 'b', 'g']:
            raise ValueError('Invalid value')
        if not(isinstance(radius,int) and radius>0):
            raise ValueError('Invalid value')
```

```
        if not(isinstance(diametr,int) and diametr>0 and diametr==2*radius):
            raise ValueError('Invalid value')

        self.perimeter = perimeter
        self.area = area
        self.color = color
        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}, диаметр {self.diametr}."
    '''Преобразование к строке вида: Circle: Периметр <периметр>, площадь <площадь>, цвет фигуры <цвет фигуры>, радиус <радиус>, диаметр <диаметр>.'''

    def __add__(self):
        return self.perimeter+self.area
    '''Сложение площади и периметра окружности. Возвращает число, полученное при сложении площади и периметра окружности.'''

    def __eq__(self,other):
        if self.radius == other.radius:
            return True
        return False
    '''Метод возвращает True, если два объекта класса равны и False иначе. Два объекта типа Circle равны, если равны их радиусы.'''

class PolygonList(list):

    def __init__(self, name):
        super().__init__()
        self.name = name

        '''1. Вызвать конструктор базового класса.
           2. Передать в конструктор строку name и присвоить её полю name созданного объекта'''
```

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

    def print_colors(self):
        colors = [p.color for p in self if isinstance(p, Polygon)]
        for i in range(len(colors)):
            print(f'{i + 1} многоугольник: {colors[i]}')
            '''Вывести цвета всех многоугольников.'''

    def print_count(self):
        count = len([p for p in self if isinstance(p, Polygon)])
        print(count)
        '''Вывести количество многоугольников. в списке'''

class CircleList(list):
    def __init__(self, name):
        ''' 1. Вызвать конструктор базового класса.
            2. Передать в конструктор строку name и присвоить её полю
name созданного объекта'''
        super().__init__()
        self.name = name

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

```
def print_colors(self):
    '''Вывести цвета всех изогнутых фигур.'''
    colors = [c.color for c in self if isinstance(c, Circle)]
    for i in range(len(colors)):
        print(f'{i + 1} окружность: {colors[i]}')

def total_area(self):
    '''Посчитать и вывести общую площадь всех окружностей.'''
    total_area = sum(c.area for c in self if isinstance(c, Circle))
    print(total_area)
```