МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Информационные технологии»

Тема: Лабораторная работа №1. Парадигмы программирования

Студент гр. 3343 _____ Синицкая Д.В.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2024

Цель работы

Изучение парадигм программирования, создание программы на языке программирования Python, в которой реализуются несколько классов фигур и списков для них.

Задание

Вариант 1. Даны фигуры в двумерном пространстве. Базовый класс – фигура Figure, многоугольник – Polygon, окружность – Circle:

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'.

class Polygon

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.

class Circle:

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:

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.

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.

Выполнение работы

Иерархия описанных в задании классов.

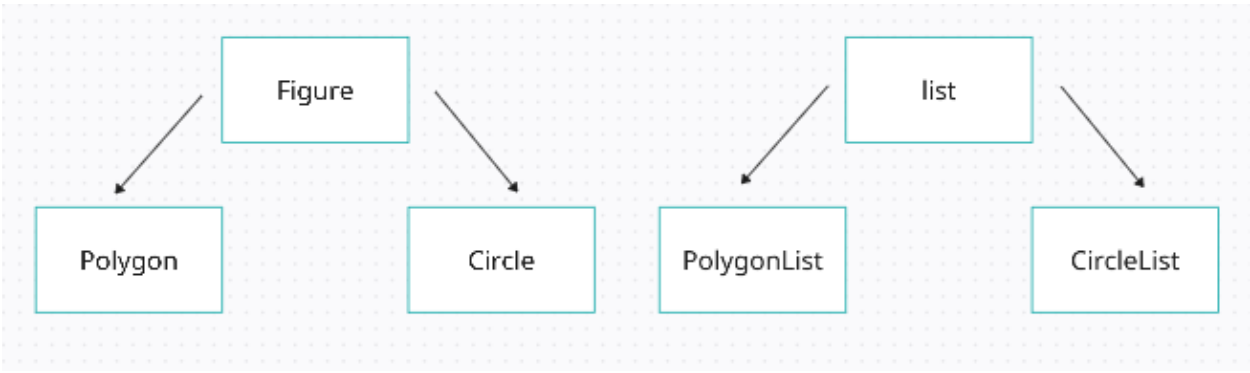

Рисунок 1 – Изображение иерархии классов

Были переопределены следующие методы:

класс Figure:

метод __init__(self, perimeter, area, color) – реализован конструктор по заданию.

класс Polygon(Figure):

метод __init__(self, perimeter, area, color, angle_count, equilateral, biggest_angle) – реализован конструктор по заданию.

метод __str__(self) – переопределено строковое представление объекта класса.

метод __add__(self) – переопределена операция сложения.

метод __eq__(self, other) – переопределена операция сравнения.

класс Circle(Figure):

метод __init__(self, perimeter, area, color, radius, diametr) – реализован конструктор по заданию.

метод __str__(self) – переопределено строковое представление объекта класса.

метод __add__(self) – переопределена операция сложения.

метод __eq__(self, other) – переопределена операция сравнения.

класс PolygonList(list):

метод __init__(self, name) – реализован конструктор по заданию.

метод append(self, p_object) – переопределена операция добавления элемента в список согласно условию задания.

класс CircleList(list):

метод __init__(self, name) – реализован конструктор по заданию.

метод extend(self, iterable) – переопределена операция добавления элементов коллекции в список согласно условию задания.

Метод __str__() будет использоваться для представления объекта в виде строки при его выводе, например, при вызове функции print() для объекта класса Polygon или Circle. Он возвращает строку, описывающую экземпляр объекта.

Метод __add__() будет вызываться при сложении объектов классов Polygon и Circle. В данном случае он определен для возвращения суммы значений площади и периметра объекта.

Переопределенные методы класса list, такие как append() для PolygonList и extend() для CircleList, будут работать как ожидается, так как они были переопределены в подклассах с учетом особенностей этих классов.

Например, в PolygonList метод append() переопределен таким образом, что позволяет добавлять только объекты типа Polygon в список. Если попытаться добавить объект другого типа, будет вызвано исключение TypeError.

Аналогично, метод extend() в CircleList позволяет только добавлять объекты типа Circle из итерируемого объекта в список.

Таким образом, переопределенные методы будут работать, следуя логике, заданной в соответствующих классах.

Разработанный программный код см. в приложении А.

Выводы

В ходе лабораторной работы были узучены виды парадигм программирования, создана программа на языке программирования Python, реализующая несколько класов фигур и списков для них.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import math

class Figure:
    def __init__(self, perimeter, area, color):
        if not isinstance(perimeter, int) or perimeter <= 0:
            raise ValueError('Invalid value')
        if not isinstance(area, int) or 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):
       def __init__(self, perimeter, area, color, angle_count,
equilateral, biggest_angle):
        super().__init__(perimeter, area, color)

        if not isinstance(angle_count, int) or angle_count <= 2:
            raise ValueError('Invalid value')

        if not isinstance(equilateral, bool):
            raise ValueError('Invalid value')

        if not isinstance(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.area + self.perimeter

    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 not isinstance(radius, int) or radius <= 0 or
diametr != 2 * radius:
                    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}, диаметр
{self.diametr}.'

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

            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).__name__}')

            def print_colors(self):
                output = ''
                for idx, polygon in enumerate(self, start=1):
                    output += f'{idx} многоугольник: {polygon.color}\n'
                output = output.rstrip()
                print(output)

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

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

            def extend(self, iterable):
                 circles = [item for item in iterable if isinstance(item,
Circle)]
                super().extend(circles)

            def print_colors(self):
                output = ''
```

```
            for idx, circle in enumerate(self, start=1):
                output += f'{idx} окружность: {circle.color}\n'
            output = output.rstrip()
            print(output)

        def total_area(self):
            total_area = sum([math.pi * circle.radius ** 2 for circle
in self])
            print(math.ceil(total_area))
```