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 _____ Пименов П.В.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2024

Цель работы

Изучить, какие бывают парадигмы программирования. Создать программу на языке Python, реализующую 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

Указанные в задании классы, их методы и поля успешно реализованы.

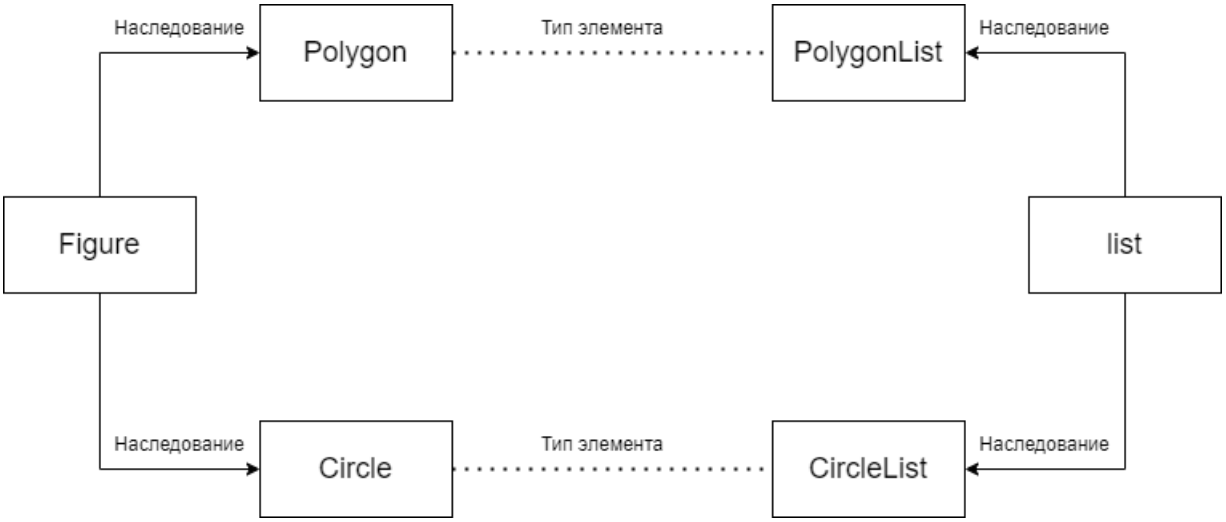

Рисунок 1 – Изображение иерархии классов

Были переопределены следующие методы:

- класс Figure
  - метод __init__ – реализован конструктор по заданию
- класс Polygon
  - метод __init__ – реализован конструктор по заданию
  - метод __str__ – переопределено строковое представление объекта класса
  - метод __add__ – переопределена операция сложения
  - метод __eq__ – переопределена операция сравнения
- класс Circle
  - метод __init__ – реализован конструктор по заданию
  - метод __str__ – переопределено строковое представление объекта класса
  - метод __add__ – переопределена операция сложения
  - метод __eq__ – переопределена операция сравнения
- класс PolygonList
  - метод __init__ – реализован конструктор по заданию

    - метод append – переопределена операция добавления элемента в список согласно условию задания
- класс CircleList
    - метод __init__ – реализован конструктор по заданию
    - метод extend – переопределена операция добавления элементов коллекции в список согласно условию задания

Метод __str__ будет работать, когда будет когда будет нужно строковое представление объекта класса, например, при вызове print(circle), если circle – объект класса Circle. Метод __add__ будет работать, когда была вызвана операция сложения двух объектов. Однако в данном случае, будет корректен только вызов вида circle.__add__(), где circle – объект класса Circle, поскольку по условию задания метод __add__ не должен принимать второй операнд. Переопределенные методы классов PolygonList и CircleList будут работать, поскольку переопределенные методы классов работают «сверху-вниз»: сначала вызывается переопределенный метод класса-наследника, потом класса-родителя, потом класса-родителя-родителя и т. д. Примеры использования см. в п. 1, 2 части «Тестирование».

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	circle = Circle(123, 123, 'r', 123, 246) circle_list = CircleList("list") circle_list.extend([circle, circle, 1, '1', True]) circle_list.print_colors()	1 окружность: r 2 окружность: r	Программа работает корректно
2.	polygon = Polygon(8, 4, 'g', 4, True, 90) polygon_list = PolygonList("list") polygon_list.append(polygon) polygon_list.print_colors()	1 многоугольник: g	Программа работает корректно
3.	circle = Circle(123, 123, 'r', 123, 246) circle_list = CircleList("list") circle_list.extend([circle, circle, 1, '1', True]) circle_list.total_area()	246	Программа работает корректно
4.	circle = Circle(123, 123, 'r', 123, 246) print(circle)	Circle: Периметр 123, площадь 123, цвет фигуры r, радиус 123, диаметр 246.	Программа работает корректно

Выводы

Изучено, какие бывают парадигмы программирования. Создана программа на языке Python, реализующую несколько классов фигур и списков для них.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Figure:
    def __init__(self, perimeter, area, color):
        if not isinstance(perimeter, int) or not isinstance(area,
int) or not isinstance(color, str):
            raise ValueError('Invalid value')
          if perimeter > 0 and area > 0 and color in {'r', 'g',
'b'}:
            self.perimeter = perimeter
            self.area = area
            self.color = color
        else:
            raise ValueError('Invalid value')


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 not
isinstance(equilateral, bool) or not isinstance(biggest_angle, int):
            raise ValueError('Invalid value')
        if angle_count > 2 and biggest_angle > 0:
            self.angle_count = angle_count
            self.equilateral = equilateral
            self.biggest_angle = biggest_angle
        else:
            raise ValueError('Invalid value')

    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
        else:
            raise NotImplementedError(f'Cannot compare Polygon to
{type(other)}')


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

```
            super().__init__(perimeter, area, color)
            if not isinstance(radius, int) or not isinstance(diametr,
int):
                raise ValueError('Invalid value')
            if radius > 0 and diametr > 0 and diametr == 2 * radius:
                self.radius = radius
                self.diametr = diametr
            else:
                raise ValueError('Invalid value')

        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.perimeter == other.perimeter and
self.area == other.area and self.radius == other.radius
            else:
                raise NotImplementedError(f'Cannot compare Circle to
{type(other)}')


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

        def append(self, __object):
            if isinstance(__object, Polygon):
                super().append(__object)
            else:
                raise TypeError(f'Invalid type {type(__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 item in __iterable:
                if isinstance(item, Circle):
                    super().append(item)
```

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

def total_area(self):
    print(sum([x.area for x in self]))
```