

**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

Гусева Е.А.

Преподаватель

Иванов Д.В.

Санкт-Петербург

2024

## **Цель работы**

Научиться использовать объекто-ориентированный подход программирования в языке Python, изучить исключения, а также императивную и декларативную подходы.

## **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:

- Figure
  - Polygon
  - Circle
- list
  - PolygonList
  - CircleList

### 2. Переопределенные методы:

`__init__()` - Метод, который был переопределен для всех классов.

Используется для инициализации класса.

`__add__()` - Метод, который был переопределен в классе Figure.

Используется при попытке сложить один объект вместе с другим.

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

`__eq__()` - Метод, который используется при сравнении объектов.

Метод `__str__()` будет использоваться, если попытаться обратиться к объекту как к строке. Он будет возвращать строковое представление объекта.

Метод `__add__()` будет использоваться при использовании двух объектов вместе с оператором `+`. Он позволяет определить, как объекты должны взаимодействовать с этим оператором.



## **Выводы**

Был получен опыт работы с парадигмой объектно-ориентированного программирования, а также были изучены исключения, а также императивная и декларативная парадигмы.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main\_for\_lb1.py

```
class Figure():
    def __init__(self, perimeter, area, color):
        self.perimeter = perimeter
        self.area = area
        self.color = color

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

class Polygon(Figure):
    def __init__(self, perimeter, area, color, angle_count, equilateral,
biggest_angle):
        super().__init__(perimeter, area, color)
        self.angle_count = angle_count
        self.equilateral = equilateral
        self.biggest_angle = biggest_angle

        if not isinstance(self.angle_count, int) or self.angle_count <= 2
or not isinstance(self.equilateral, bool) or not
isinstance(self.biggest_angle, int) or self.biggest_angle <= 0:
            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.perimeter + self.area

    def __eq__(self, other):
        if other.__class__.__name__ == self.__class__.__name__:
```

```

        if self.perimeter == other.perimeter and self.area ==
other.area and self.angle_count == other.angle_count:
            return True
        else:
            return False

class Circle(Figure):
    def __init__(self, perimeter, area, color, radius, diametr):
        super().__init__(perimeter, area, color)
        self.radius = radius
        self.diametr = diametr
        if not isinstance(radius, int) or not isinstance(diametr, int) or
not radius > 0 or not diametr == 2 * radius:
            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.perimeter + self.area

    def __eq__(self, other):
        if other.__class__.__name__ == self.__class__.__name__:
            if self.radius == other.radius:
                return True
            else:
                return False

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

    def append(self, p_object):
        if p_object.__class__.__name__ == "Polygon":
            super().append(p_object)
        else:
            raise TypeError(f"Invalid type {type(p_object)}")

```

```

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

def print_count(self):
    print(len(list(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):
                self.append(item)

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

    def total_area(self):
        total = 0
        for i in list(self):
            total += i.area
        print(total)

```