**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 _____ Волох И.О.

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.

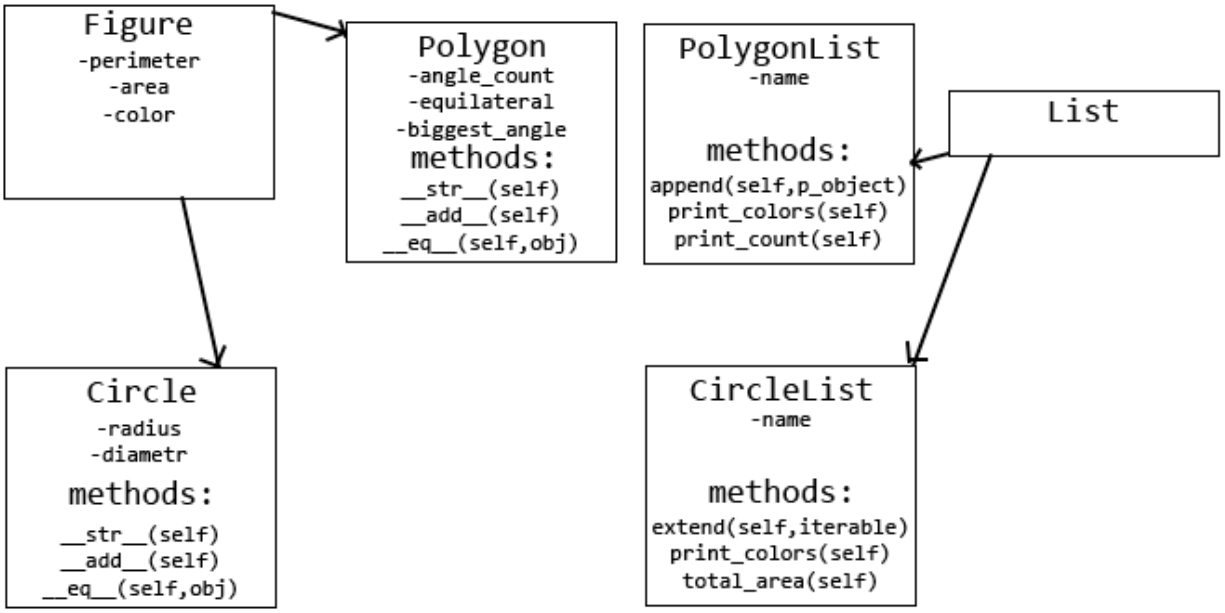


Рисунок 1 – Изображение иерархии классов

В рамках выполнения работы в классах Circle и Polygon были переопределены методы __str__(), __add__(), __eq__(). При вызове __str__(self) будет использоваться возвращаемое значение в переопределенном методе. Оператор == в вызывает метод __eq__(self,obj), который определяет поведение оператора равенства для объектов данного класса. Метод __add__(self) будет вызываться при сложении двух экземпляров класса. В классах CircleList и PolygonList унаследованных от класса List, были переопределены методы append(self,p_object), extend(self,iterable). Благодаря тому, что родительский метод вызывается с помощью функции super().

Разработанный программный код см. в приложении А.

**Выводы**

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

## Приложение А

## Исходный код программы

```
    class Figure:
        def __init__(self, perimeter, area, color):
            clr = "rbg"
            self.perimeter = perimeter
            self.area = area
            self.color = color
             if not(isinstance(perimeter, int)) or (self.area <= 0)
or (self.perimeter <= 0) or not(isinstance(area, int)) or (color not
in clr):
                raise ValueError('Invalid value')
    class Polygon(Figure): #Наследуется от класса 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(angle_count, int)) or (angle_count <=
2) or (self.biggest_angle <= 0) or (self.angle_count <= 0) or
not(isinstance(equilateral, bool)) or not(isinstance(biggest_angle,
int)) or (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):
            summ = self.perimeter + self.area
            return summ
        def __eq__(self, obj):
                if (self.area == obj.area) and (self.perimeter ==
obj.perimeter) and (self.angle_count == obj.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 (self.diametr <= 0) or (self.diametr
!= 2 * self.radius) or (self.radius <= 0):
                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):
            summ = self.perimeter + self.area
            return summ
```

```
        def __eq__(self, obj):
            if (self.radius == obj.radius) and (self.diametr ==
obj.diametr):
                return True
            else:
                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("Invalid type <тип_объекта
p_object>")
        def print_colors(self):
            for i in range(len(self)):
                print(f'{1 + i} 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):
            if isinstance(iterable, list):
                for el in iterable:
                    if isinstance(el, Circle):
                        self.append(el)
            else:
                        raise TypeError("Invalid type <тип_объекта
p_object>")
        def print_colors(self):
            for i in range(len(self)):
                print(f'{1 + i} окружность: {self[i].color}')
        def total_area(self):
            smm = 0
            for i in self:
                smm += i.area
            print(smm)
```