**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:** **Парадигмы программирования**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Епонишникова А.И |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Целью работы является изучение парадигм программирования и написание программы с использованием ООП

## Задание

Вариант 1.

Базовый класс — печатное издание *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'.

Класс многоугольник – *Polygon* наследуется от класса *Figure*.

Поля объекта класс 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.

Класс окружность – *Circle* наследуется от класса *Figure*.

Поля объекта класс *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:

Многоугольник:

*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 газет - 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 многоугольников в виде строки (нумерация начинается с 1):

<i> многоугольник: <color[i]>

<j> многоугольник: <color[j]> ...

* Метод total\_area(): Посчитать и вывести общую площадь всех окружностей.

## Выполнение работы

Иерархия описанных классов:

Сначала идет Figure (родительский класс), а от него Polygon и Circle.

List (родительский класс), от него PolygonList и CircleList.

Классы:

Figure принимает *perimeter, area, color* в качестве аргументов. Проводится проверка на соответствие типам параметров, если да, то *perimeter, area, color* присваиваются полям класса, иначе вызывается исключение ValueError с сообщением: “Invalid value”.

*Polygon* наследуется от класса *Figure.* Принимает *perimeter, area, color, angle\_count, equilateral, biggest\_angle* в качестве параметров. Поля *perimeter, area, color* передаются конструктору родительского класса. Проводится проверка на соответствие типам, если да, то параметры *angle\_count, equilateral, biggest\_angle* присваиваются полям класса. Иначе вызывается исключение ValueError с сообщением: “Invalid value”. Методы: \_\_str\_\_(приведение класса к типу string), \_\_add\_\_(сложение периметра и площади многоугольника), \_\_eq\_\_(сравнение с другим экземпляром по полям perimeter, area, angle\_count)

Circle наследуется от класса *Figure.* Принимает *perimeter, area, color, radius, diametr* в качестве параметров. Поля *perimeter, area, color* передаются конструктору родительского класса. Проводится проверка на соответствие типам, если да, то параметры *radius, diametr* присваиваются полям класса. Иначе вызывается исключение ValueError с сообщением: “Invalid value”. Методы: \_\_str\_\_(приведение класса к типу string), \_\_add\_\_(сложение периметра и площади окружности), \_\_eq\_\_(сравнение с другим экземпляром по полю redius)

*PolygonList* наследуется от класса *list*. В конструктор передается имя списка, в нем вызывается родительский конструктор, а затем присваивается параметр *name*. Переопределяется метод *append*, в котором проверяется тип добавляемого объекта, в случае несоответствия, вызывается TypeError, иначе вызывается *append* у родительского метода. Метод *print\_colors* выводит номер многоугольника и его цвет. Метод print\_count возвращает количество многоугольников в списке.

*CircleList* наследуется от класса *list*. В конструктор передается имя списка, в нем вызывается родительский конструктор, а затем присваивается параметр *name*. Переопределяется метод *extend*, в цикле проверяется все ли элементы *iterable* корректного типа, в случае несоответствия метод завершается, иначе вызывается родительский *extend*. Метод *print\_colors* выводит номер окружности и ее цвет. Метод *total\_area* печатает суммарную площадь окружностей в списке.

Разработанный программный код см. в приложении А.

## Выводы

Были изучены парадигмы программирования. Написана программа, содержащая классы, методы и исключения.

# **Приложение А Исходный код программы**

Название файла: lab.py

class Figure:

def \_\_init\_\_(self, perimeter, area, color):

if isinstance(perimeter, int) and (perimeter > 0) and isinstance(area, int) and (area > 0) and color in ['r', 'b', 'g']:

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 isinstance(angle\_count,int) and (angle\_count > 2) and isinstance(equilateral, bool) and isinstance(biggest\_angle, int) and (0 < biggest\_angle < 180):

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):

return self.perimeter == other.perimeter and self.area == other.area and self.angle\_count == other.angle\_count

class Circle(Figure):

def \_\_init\_\_(self, perimeter, area, color, radius, diametr):

super().\_\_init\_\_(perimeter, area, color)

if isinstance(radius,int) and (radius>0) and isinstance(diametr,int) 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):

return self.radius == other.radius

class PolygonList(list):

def \_\_init\_\_(self, name):

super().\_\_init\_\_(self)

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)}")

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)

self.name = name

def extend(self, iterable):

for i in iterable:

if isinstance(i, Circle):

super().append(i)

def print\_colors(self):

for i in range(len(self)):

print(f"{i + 1} окружность: {self[i].color}")

def total\_area(self):

sum\_areas = 0

for i in self:

sum\_areas += i.area

print(sum\_areas)