**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. 3343 |  | Гребнев Е. Д |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2024

## Цель работы

Научится работать с классами, создавать методы и функции для классов, понять принцип наследования и переопределения, понять, как работает super().

Необходимо создать программу, которая может на основе различных классов создавать объекты фигур и работать с ними. Также программа должна уметь определять верный тип данных, а также уметь добавлять в определённую группу объектов.

## Задание

Даны фигуры в двумерном пространстве.

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(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(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(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(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

В рамках задания по лабораторной работе требуется разработать классы, содержащие определённые методы, представляющие собой геометрические фигуры с заданными параметрами, а также списки для их хранения.

Родительский класс `Figure` содержит информацию о периметре, площади и цвете фигуры. При создании экземпляра класса происходит проверка типа входных данных и их корректности: периметр и площадь должны быть положительными числами, а цвет фигуры должен быть одним из 'r', 'g' или 'b'.

Класс `Polygon` описывает многоугольник и содержит информацию о количестве углов, равносторонности и наибольшем угле. Добавлены методы для вычисления суммы периметра и площади фигуры, вывода информации об объекте и сравнения объектов по периметру, площади и количеству углов. Проверяется, что количество углов больше 2 и наибольший угол больше нуля.

Класс `Circle` описывает окружность с заданным радиусом и диаметром. Добавлены методы для вывода информации об объекте, сравнения объектов по радиусам и вычисления суммы периметра и площади окружности. Проверяется, что радиус и диаметр являются положительными числами, а диаметр равен удвоенному радиусу.

Метод `\_\_str\_\_()` отвечает за строковое представление объекта класса, а метод `\_\_add\_\_()` позволяет складывать два объекта класса (в данном случае происходит сложение площади и периметра фигур).

Класс `PolygonList` наследуется от класса `list` и переопределяет метод `append`, чтобы проверить, что добавляемый элемент является объектом класса `Polygon`. Метод `print\_colors` выводит информацию о цвете каждого многоугольника, а `print\_counts` - количество элементов в списке.

Класс `CircleList` также наследуется от класса `list` и переопределяет метод `extend`, чтобы добавить только объекты класса `Circle`. Метод `print\_colors` выводит информацию о цвете каждой окружности, а `total\_area` - общую площадь всех окружностей.

Переопределённые методы `append` и `extend` в классах `PolygonList` и `CircleList` вызывают соответствующие методы родительского класса `list` с помощью `super()`, обеспечивая корректное добавление объектов.

## Выводы

В процессе работы были изучены принципы наследования от различных классов, переопределение их методов, а также применение функции `super()` для доступа к методам родительского класса.

Результатом этой работы стала программа, способная создавать экземпляры различных классов фигур, добавлять их в соответствующие группы и выполнять с ними операции.

# Приложение А Исходный код программы

Название файла: main.py

class Figure:

def \_\_init\_\_(self, perimeter: int, area: int, color: str) -> None:

if not all(isinstance(x, int) and x > 0 for x in (perimeter, area)) or color not in {'r', 'b', 'g'}:

raise ValueError('Invalid parameters')

self.perimeter, self.area, self.color = perimeter, area, color

def \_\_str\_\_(self):

return f'{self.\_\_class\_\_.\_\_name\_\_}: Периметр {self.perimeter}, площадь {self.area}, цвет фигуры {self.color}'

class Polygon(Figure):

def \_\_init\_\_(self, perimeter: int, area: int, color: str, angle\_count: int, equilateral: bool, biggest\_angle: int) -> None:

if not all(isinstance(x, int) and x > 0 for x in (perimeter, angle\_count, area, biggest\_angle)) or angle\_count < 2 or not isinstance(equilateral, bool):

raise ValueError('Invalid parameters')

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

self.angle\_count, self.equilateral, self.biggest\_angle = angle\_count, equilateral, biggest\_angle

def \_\_str\_\_(self):

return f'{super().\_\_str\_\_()}, 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 isinstance(other, Polygon) and super().\_\_eq\_\_(other) and self.angle\_count == other.angle\_count

class Circle(Figure):

def \_\_init\_\_(self, perimeter: int, area: int, color: str, radius: int, diametr: int) -> None:

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

if not all(isinstance(x, int) and x > 0 for x in (radius, diametr)) or diametr != 2 \* radius:

raise ValueError('Invalid parameters')

self.radius, self.diametr = radius, diametr

def \_\_str\_\_(self):

return f'{super().\_\_str\_\_()}, радиус {self.radius}, диаметр {self.diametr}.'

def \_\_add\_\_(self):

return self.area + self.perimeter

def \_\_eq\_\_(self, other):

return isinstance(other, Circle) and super().\_\_eq\_\_(other) and self.radius == other.radius

class PolygonList(list):

def \_\_init\_\_(self, name: str) -> None:

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)}")

def print\_colors(self):

for i, polygon in enumerate(self, 1):

print(f"{i} многоугольник: {polygon.color}")

def print\_count(self):

print(len(self))

class CircleList(list):

def \_\_init\_\_(self, name: str) -> None:

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

self.name = name

def extend(self, iterable):

super().extend(filter(lambda x: isinstance(x, Circle), iterable))

def print\_colors(self):

for i, circle in enumerate(self, 1):

print(f"{i} окружность: {circle.color}")

def total\_area(self):

print(sum(circle.area for circle in self))