**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 |  | Валиев Р.А. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Научиться использовать объекто-ориентированный подход программирования в языке Python.

## 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.

3. 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.

4. Переопределенные методы класса list будут работать, т. к. они являются частью класса list, от которого они наследуются, а значит, при переопределении у новых классов будет просто добавлена новая логика при работе с ними. Примером являются методы append и extend, которые работают точно также, как и методы родительского класса list, но к тому же, они проверяют, является ли объект, который добавляется в список, объектом нужного класса.

## Выводы

Был получен опыт работы с парадигмой объектно-ориентированного программирования, а также были изучены особенности переопределения методов классов в языке программирования Python.

# Приложение А Исходный код программы

Название файла: main.py

def check\_int(params):

if all(isinstance(obj, int) and obj > 0 for obj in params):

return 1

return 0

class Figure:

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

if check\_int([perimeter, area]) 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 check\_int([angle\_count, biggest\_angle]) and angle\_count>2 and isinstance(equilateral, bool):

self.angle\_count = angle\_count

self.biggest\_angle = biggest\_angle

self.equilateral = equilateral

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.perimeter + self.area

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

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

class Circle(Figure): # Наследуется от класса Figure

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

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

if check\_int([radius, diametr]) 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.perimeter + self.area

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

return self.radius == other.radius

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(f"Invalid type {type(p\_object)}")

def print\_colors(self):

print('\n'.join([f'{i+1} многоугольник: {self[i].color}' for i in range(len(self))]))

def print\_count(self):

print(len(self))

class CircleList(list):

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

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

self.name = name

def extend(self, iterable):

for i in iterable:

if isinstance(i, Circle):

super().append(i)

def print\_colors(self):

print('\n'.join([f'{i+1} окружность: {self[i].color}' for i in range(len(self))]))

def total\_area(self):

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