**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 |  | Охрименко Д.И. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Изучить применение объектно-ориентированного программирования на языке Python.

## **Зада**ние.

**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:

1. Figure
   1. Polygon
   2. Circle
2. list
   1. PolygonList
   2. CircleList

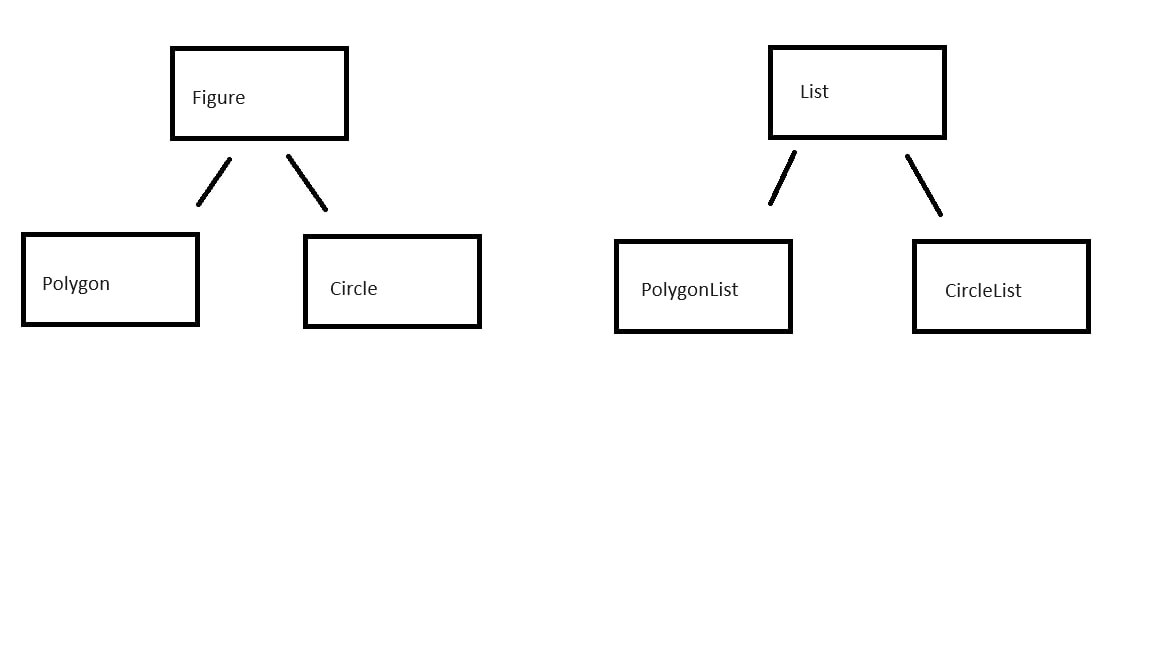


Рисунок 1 - Иерархия классов

2. Переопределенные методы:

\_\_init\_\_() - метод для инициализации класса, который был переопределен для всех классов.

\_\_add\_\_() - метод, переопределенный в классе Figure, который выполняется при попытке сложить один объект с другим.

\_\_str\_\_() - метод, переопределенный в классе Figure, который выполняется при попытке сложить один объект с другим.

\_\_eq\_\_() - метод, переопределенный в классе Figure, который выполняется при попытке сложить один объект с другим.

3. В каких случаях будут использованы методы \_\_str\_\_() и \_\_add\_\_():

Метод \_\_str\_\_() будет вызван при попытке преобразовать объект в строку, чтобы получить его текстовое представление.

Метод \_\_add\_\_() будет вызван при использовании оператора + с двумя объектами, позволяя определить, как объекты должны быть сложены вместе.

4. Будут ли работать переопределенные методы класса list для PolygonList и CircleList:

Переопределенные методы класса list будут работать, т. к. для нахождения метода по имени программа поднимается снизу-вверх по иерархии классов, то есть сначала ищет переопределённые методы в самом экземпляре, не найдя - переходит к родительским классам. Например, методы append и extend будут работать аналогично методам родительского класса list, но при этом будут проверять тип объектов, которые добавляются в список.

## Выводы

Приобретены навыки работы с объектно-ориентированным программированием и изучены особенности переопределения методов классов и обработки ошибок в Python.

# Приложение А Исходный код программы

Название файла: lb1.py

class Figure:

'''Поля объекта класс Figure:

perimeter - периметр фигуры (в сантиметрах, целое положительное число)

area - площадь фигуры (в квадратных сантиметрах, целое положительное число)

color - 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'.

'''

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

if not(isinstance(perimeter, int) and perimeter > 0):

raise ValueError('Invalid value')

if not(isinstance(area, int) and area > 0):

raise ValueError('Invalid value')

if color not in ['r', 'b', 'g']:

raise ValueError('Invalid value')

self.perimeter = perimeter

self.area = area

self.color = color

class Polygon(Figure): # Наследуется от класса Figure

'''Поля объекта класс Polygon:

perimeter - периметр фигуры (в сантиметрах, целое положительное число)

area - площадь фигуры (в квадратных сантиметрах, целое положительное число)

color - цвет фигуры (значение может быть одной из строк: 'r', 'b', 'g')

angle\_count - количество углов (целое положительное значение, больше 2)

equilateral - равносторонний (значениями могут быть или True, или False)

biggest\_angle - самый большой угол (или любой угол, если многоугольник равносторонний) (в градусах, 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'.

'''

def \_\_init\_\_(self, perimeter, area, color, angle\_count, equilateral, biggest\_angle):

if not(isinstance(perimeter, int) and perimeter > 0):

raise ValueError('Invalid value')

if not(isinstance(area, int) and area > 0):

raise ValueError('Invalid value')

if not(isinstance(angle\_count, int) and angle\_count > 2):

raise ValueError('Invalid value')

if not(isinstance(biggest\_angle, int) and biggest\_angle > 0):

raise ValueError('Invalid value')

if not isinstance(equilateral, bool):

raise ValueError('Invalid value')

if color not in ['r', 'b', 'g']:

raise ValueError('Invalid value')

self.perimeter = perimeter

self.area = area

self.angle\_count = angle\_count

self.color = color

self.equilateral = equilateral

self.biggest\_angle = biggest\_angle

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

'''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>.'''

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):

'''Сложение площади и периметра многоугольника. Возвращает число, полученное при сложении площади и периметра многоугольника.'''

return self.perimeter + self.area

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

'''Метод возвращает True, если два объекта класса равны и False иначе. Два объекта типа Polygon равны, если равны их периметр, площадь и количество углов.'''

if isinstance(other, Polygon):

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

return False

class Circle(Figure):

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

if not(isinstance(perimeter, int) and perimeter > 0):

raise ValueError('Invalid value')

if not(isinstance(area, int) and area > 0):

raise ValueError('Invalid value')

if not(isinstance(radius, int) and radius > 0):

raise ValueError('Invalid value')

if not(isinstance(diametr, int) and diametr == 2 \* radius):

raise ValueError('Invalid value')

if color not in ['r', 'b', 'g']:

raise ValueError('Invalid value')

self.perimeter = perimeter

self.area = area

self.color = color

self.radius = radius

self.diametr = diametr

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

'''Преобразование к строке вида: Circle: Периметр <периметр>, площадь <площадь>, цвет фигуры <цвет фигуры>, радиус <радиус>, диаметр <диаметр>.'''

return f"Circle: Периметр {self.perimeter}, площадь {self.area}, цвет фигуры {self.color}, радиус {self.radius}, диаметр {self.diametr}."

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

'''Сложение площади и периметра окружности. Возвращает число, полученное при сложении площади и периметра окружности.'''

return self.area + self.perimeter

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

'''Метод возвращает True, если два объекта класса равны и False иначе. Два объекта типа Circle равны, если равны их радиусы.'''

if isinstance(other, Circle):

return self.radius == other.radius

return False

class PolygonList(list):

''' 1. Вызвать конструктор базового класса.

2. Передать в конструктор строку name и присвоить её полю name созданного объекта'''

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

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

self.name = name

def append(self, 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>'''

if isinstance(p\_object, Polygon):

super().append(p\_object)

else:

raise TypeError(f'Invalid type {type(p\_object).\_\_name\_\_}')

def print\_colors(self):

'''Вывести цвета всех многоугольников.'''

colors = [polygon.color for polygon in self if isinstance(polygon, Polygon)]

for i in range(len(colors)):

print(f'{i + 1} многоугольник: {colors[i]}')

def print\_count(self):

'''Вывести количество многоугольников. в списке'''

count = len([polygon for polygon in self if isinstance(polygon, Polygon)])

print(count)

class CircleList(list):

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

''' 1. Вызвать конструктор базового класса.

2. Передать в конструктор строку name и присвоить её полю name созданного объекта'''

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

self.name = name

def extend(self, 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.'''

for item in iterable:

if isinstance(item, Circle):

self.append(item)

def print\_colors(self):

'''Вывести цвета всех изогнутых фигур.'''

colors = [circle.color for circle in self if isinstance(circle, Circle)]

for i in range(len(colors)):

print(f'{i + 1} окружность: {colors[i]}')

def total\_area(self):

'''Посчитать и вывести общую площадь всех окружностей.'''

total\_area = sum(circle.area for circle in self if isinstance(circle, Circle))

print(total\_area)