**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. 3341 |  | Ягудин Д. Р. |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2024

## Цель работы

Цель этой работы была создание иерархии классов для представления различных геометрических фигур (многоугольников, окружностей) и их списков. Определили основные атрибуты и методы для каждого класса, а также переопределили методы базового класса object для улучшения функциональности и взаимодействия с объектами.

## 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: #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. Создаем базовый класс Figure, который содержит атрибуты perimeter, area, color. При инициализации объекта проверяем на корректность введенные данные.

2. Создаем класс Polygon, который наследуется от класса Figure и добавляет атрибуты angle\_count, equilateral, biggest\_angle. При инициализации объекта вызываем init родительского класса, затем проверяем biggest\_angle, angle\_count и equilateral на корректность.

3. Создаем класс Circle, который также наследуется от класса Figure и добавляет атрибуты radius, diametr. При инициализации объекта вызываем init родительского класса, затем проверяем radius, diametr на корректность.

4. Создаем класс PolygonList, который наследуется от списка и добавляет методы init, append, print\_count, print\_color. Метод append позволяет добавлять только объекты класса Polygon в список.

6. Создаем класс CircleList, аналогично PolygonList, добавляет методы init, extend, total\_area, print\_colors. Метод extend позволяет добавлять в список только объекты класса Circle.

Этот код реализует иерархию классов персонажей (воин, маг, лучник) и списков для хранения персонажей каждого класса. Каждый класс персонажа имеет свои уникальные атрибуты и методы.

1. Изображение иерархии классов:

Figure

/ | \

Circle Polygon

PolygonList <-- list

CircleList <-- list

2. В переопределении методов класса объекта object или других методов:

- Метод `\_\_init\_\_`: переопределен в каждом классе для инициализации атрибутов.

- Метод `\_\_str\_\_`: переопределен для возвращения строкового представления объекта.

3. Метод `\_\_add\_\_` в классе Polygon и в классе Circle переопределен таким образом, что значение area прибавляется к значению perimeter. Метод `\_\_eq\_\_` проверяет равенство атрибута area у двух объектов данного класса Polygon и атрибута radius у двух объектов класса Circle.

4. Переопределенные методы класса list для созданных списков не будут работать, т.к. созданные классы WarriorList, MagicianList и ArcherList являются подклассами списка (list), но они не переопределяют поведение всех методов класса list. Например, методы append, extend, print\_count могут быть вызваны нормально, так как они определены в наших классах, но такие методы, как clear или pop, которые не переопределены, будут работать как обычно для списка без дополнительной логики, определенной в наших классах.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | fig = Figure(10,25,'g') #фигура  print(fig.perimeter, fig.area, fig.color)  polygon = Polygon(10,25,'g',4, True, 90) #многоугольник  polygon2 = Polygon(10,25,'g',4, True, 90)  print(polygon.perimeter, polygon.area, polygon.color, polygon.angle\_count, polygon.equilateral, polygon.biggest\_angle)  print(polygon.\_\_str\_\_())  print(polygon.\_\_add\_\_())  print(polygon.\_\_eq\_\_(polygon2))  circle = Circle(13, 13,'r', 2, 4) #окружность  circle2 = Circle(13, 13,'g', 2, 4)  print(circle.perimeter, circle.area, circle.color, circle.radius, circle.diametr)  print(circle.\_\_str\_\_())  print(circle.\_\_add\_\_())  print(circle.\_\_eq\_\_(circle2))  polygon\_list = PolygonList(Polygon) #список многоугольников  polygon\_list.append(polygon)  polygon\_list.append(polygon2)  polygon\_list.print\_colors()  polygon\_list.print\_count()  circle\_list = CircleList(Circle) #список окружностей  circle\_list.extend([circle, circle2])  circle\_list.print\_colors()  circle\_list.total\_area() | Получено  fig = Figure(10,25,'g') #фигура  print(fig.perimeter, fig.area, fig.color)  polygon = Polygon(10,25,'g',4, True, 90) #многоугольник  polygon2 = Polygon(10,25,'g',4, True, 90)  print(polygon.perimeter, polygon.area, polygon.color, polygon.angle\_count, polygon.equilateral, polygon.biggest\_angle)  print(polygon.\_\_str\_\_())  print(polygon.\_\_add\_\_())  print(polygon.\_\_eq\_\_(polygon2))  circle = Circle(13, 13,'r', 2, 4) #окружность  circle2 = Circle(13, 13,'g', 2, 4)  print(circle.perimeter, circle.area, circle.color, circle.radius, circle.diametr)  print(circle.\_\_str\_\_())  print(circle.\_\_add\_\_())  print(circle.\_\_eq\_\_(circle2))  polygon\_list = PolygonList(Polygon) #список многоугольников  polygon\_list.append(polygon)  polygon\_list.append(polygon2)  polygon\_list.print\_colors()  polygon\_list.print\_count()  circle\_list = CircleList(Circle) #список окружностей  circle\_list.extend([circle, circle2])  circle\_list.print\_colors()  circle\_list.total\_area()  10 25 g  10 25 g 4 True 90  Polygon: Периметр 10, площадь 25, цвет фигуры g, количество углов 4, равносторонний True, самый большой угол 90.  35  True  13 13 r 2 4  Circle: Периметр 13, площадь 13, цвет фигуры r, радиус 2, диаметр 4.  26  True  1 многоугольник: g  2 многоугольник: g  2  1 окружность: r  2 окружность: g  26 | Проверка работы с корректными данными |
|  | try: #неправильные данные для фигуры  fig = Figure(-10,25,'g')  except (TypeError, ValueError):  print('OK')  try:  fig = Figure(10,-25,'g')  except (TypeError, ValueError):  print('OK')  try:  fig = Figure(10,25,-1)  except (TypeError, ValueError):  print('OK')  try:  fig = Figure(10,25,1)  except (TypeError, ValueError):  print('OK')  try:  fig = Figure(10,25,'a')  except (TypeError, ValueError):  print('OK')  try:  fig = Figure('a',25,'g')  except (TypeError, ValueError):  print('OK')  try:  fig = Figure(10,'a','g')  except (TypeError, ValueError):  print('OK')  try:  fig = Figure(0,25,'g')  except (TypeError, ValueError):  print('OK')  try:  fig = Figure(10,0,'g')  except (TypeError, ValueError):  print('OK') | OK  OK  OK  OK  OK  OK  OK  OK  OK | Проверка работы с некорректными данными |

## Выводы

Изучив иерархию классов, поняли, как использовать наследование для создания классов с общими характеристиками, поддерживая при этом уникальные особенности и методы для каждого класса. Также рассмотрели, как переопределить методы базового класса object для более удобной работы с объектами и их строковым представлением.

Также установили, что переопределенные методы класса list для созданных подклассов не будут работать для всех методов списка, поскольку классы CircleList, PolygonList наследуют методы класса list, но не переопределяют все их поведения.

# Приложение А Исходный код программы

Название файла: lb1.py

class Figure:

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

try:

color\_type = ['r', 'g', 'b']

if (perimeter <= 0) or (area <= 0) or (color not in color\_type):

raise ValueError

else:

self.perimeter = perimeter

self.area = area

self.color = color

except ValueError:

raise ValueError

class Polygon(Figure):

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

try:

color\_type = ['r', 'g', 'b']

if (perimeter <= 0) or (area <= 0) or (color not in color\_type) or (angle\_count <= 2) or (type(equilateral) != bool) or (biggest\_angle <= 0):

raise ValueError

else:

self.perimeter = perimeter

self.area = area

self.color = color

self.angle\_count = angle\_count

self.equilateral = equilateral

self.biggest\_angle = biggest\_angle

except ValueError:

raise ValueError

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):

if (self.area == other.area) and (self.perimeter == other.perimeter) and (self.angle\_count == other.angle\_count):

return True

return False

class Circle(Figure):

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

try:

color\_type = ['r', 'g', 'b']

if (perimeter <= 0) or (area <= 0) or (color not in color\_type) or (radius <= 0) or (diametr <= 0 or diametr != (radius \* 2)):

raise ValueError

else:

self.perimeter = perimeter

self.area = area

self.color = color

self.radius = radius

self.diametr = diametr

except ValueError:

raise ValueError

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):

if self.radius == other.radius:

return True

return False

class PolygonList(list):

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

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

self.name = name

def append(self, p\_object):

try:

if type(p\_object) != Polygon:

raise TypeError

else:

super().append(p\_object)

except TypeError:

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.name = name

def extend(self, iterable):

if hasattr(iterable, "\_\_iter\_\_"):

for i in range(len(iterable)):

if type(iterable[i]) == Circle:

super().append(iterable[i])

else:

raise TypeError

def print\_colors(self):

for i in range(len(self)):

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

def total\_area(self):

sum = 0

for i in range(len(self)):

sum += self[i].area

print(sum)