**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. 3342 |  | Роднов И.С. |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2024

## Цель работы

Ознакомится с работой с классами и исключениями в языке Python. При помощи полученных знаний написать программу для работы с ними, в соответствии с заданием.

## Задание

##### Базовый класс - фигура *Figure*:

class Figure:

* Поля объекта класс 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

Покажем наследование классов.

Figure

List

PolygonList

Polygon

Circle

CircleList

Рисунок 1 – Наследование классов

Опишем создание каждого класса.

Figure:

1) \_\_init\_\_ - добавление соответствующих полей в экземпляр класса. Вывод исключения, если данные неверные.

Polygon (наследование полей и функций от Figure):

1) \_\_init\_\_ - переопределение для добавления новых полей

2) \_\_str\_\_ - строковое представление экземпляра

3) \_\_eq\_\_ - сравнение с другим экземпляром (obj) по полям perimeter,

area, angle\_count

Circle (наследование полей и функций от Figure):

1) \_\_init\_\_ \_\_str\_\_ - по аналогии с Polygon

2) \_\_eq\_\_ - сравнение экземпляров по полю radius

PolygonList (наследование полей и функций от list):

1) \_\_init\_\_ - переопределение для добавления поля name

2) append – переопределение для добавления только элементов класса Figure, иначе вызывается исключение

3) print\_count – вывод количество многоугольников в списке.

CircleList (наследование полей и функций от list):

1) \_\_init\_\_ - переопределение для добавления поля name

2) extend - добавления из переданного аргумента только элементов класса Circle

3) print\_colors - вывод цвета всех окружностей в списке

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 1. | 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 | Верный вывод |

## Выводы

Реализована программа с использованием классов и методов, а также использованно наследование классов и работа с исключениями.

# Приложение А Исходный код программы

Название файла: main.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 isinstance(area, int) and color in ['r', 'g', 'b'] and perimeter > 0 and area > 0):

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):

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

if not(isinstance(angle\_count, int) and isinstance(equilateral, bool) and isinstance(biggest\_angle, int) and angle\_count > 2 and biggest\_angle > 0):

raise ValueError('Invalid value')

self.angle\_count = angle\_count

self.equilateral = equilateral

self.biggest\_angle = biggest\_angle

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

'''Преобразование к строке вида: Polygon: Периметр <периметр>, площадь <площадь>, цвет фигуры <цвет фигуры>, равносторонний <равносторонний>, прямоугольный <прямоугольный>.'''

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.'''

summ = 0

summ = self.area + self.perimeter

return summ

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

'''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(self.area == obj.area and self.perimeter == obj.perimeter and self.angle\_count == obj.angle\_count):

return True

else:

return False

class Circle(Figure): # Наследуется от класса Figure

'''Поля объекта класс Circle:

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')

radius - радиус (целое положительное число)

diametr - 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'.

'''

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

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

if not(isinstance(radius, int) and isinstance(diametr, int) and radius > 0 and diametr == radius \* 2):

raise ValueError("Invalid value")

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.'''

summ = 0

summ = self.area + self.perimeter

return summ

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

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

if(self.radius == obj.radius):

return True

else:

return False

class PolygonList(list): #– список многоугольников - наследуется от класса 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

'''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>'''

def append(self, p\_object):

if(isinstance(p\_object, Polygon)):

super().append(p\_object)

else:

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

'''Вывести цвета всех многоугольников.'''

def print\_colors(self):

for i in range(len(self)):

print(f'{i+1} многоугольник: {self[i].color}')

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

def print\_count(self):

print(len(self))

class CircleList(list): #– список изогнутых фигур - наследуется от класса 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

'''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.'''

def extend(self, iterable):

for i in iterable:

if(isinstance(i, Circle)):

super().append(i)

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

def print\_colors(self):

for i in range(len(self)):

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

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

def total\_area(self):

summ = 0

for i in range(len(self)):

summ += self[i].area

print(summ)