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 _____ Лучкин М. А.

Преподаватель _____ Иванов Д. В.

Санкт-Петербург
2024

Цель работы

Изучить такие инструменты программирования, как классы и исключения и их реализацию на языке Python. С их помощью написать программу, создающие элементы созданных классов.

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 на рисунке 1.

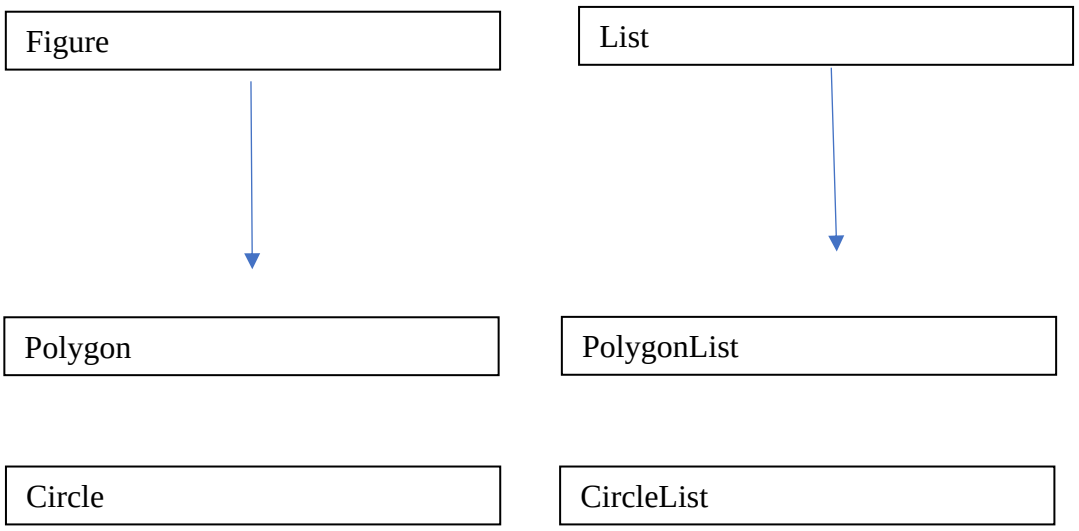

Рисунок 1 (пример наследования классов)

Опишем создание каждого класса.

Figure:

1) __init__ - добавление в экземпляр класса соответствующих полей. 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 (other) 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()
		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

	circle_list.total_area()		

Выводы

Была разработана программа, содержащая классы и их методы. Были добавлены также наследования классов, работы с экземплярами и исключения.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Figure:
    def __init__(self, perimeter, area, color):
        if not (isinstance(perimeter, int) and isinstance(area,
int)
                and isinstance(color, str) and perimeter > 0 and
area > 0 and
                color in ['r', 'g', 'b']):
            raise ValueError('Invalid value')

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


class Polygon(Figure):
    def __init__(self, perimeter, area, color, angle_count,
equilateral, biggest_angle):
        super().__init__(perimeter, area, color)
        if not ( angle_count > 2 and isinstance(biggest_angle, int)
and biggest_angle > 0\
                and isinstance(equilateral, bool)) is True:
            raise ValueError('Invalid value')

        self.angle_count = angle_count
        self.equilateral = equilateral
        self.biggest_angle = biggest_angle

    def __str__(self):
        return (f'Polygon: Периметр {self.perimeter}, площ
адь {self.area}, цвет фигуры {self.color}, '
                f'количество углов {self.angle_count},
равносторонний {self.equilateral}, самый большой уг
ол {self.biggest_angle}.')

    def __add__(self):
        return self.perimeter + self.area

    def __eq__(self, other):
        return True if self.perimeter == other.perimeter and
self.area == other.area \
                       and self.angle_count == other.angle_count
else False


class Circle(Figure):
    def __init__(self, perimeter, area, color, radius, diametr):
        super().__init__(perimeter, area, color)
        if not (isinstance(radius, int) and radius > 0 and
isinstance(diametr, int) and
                radius * 2 == diametr):
            raise ValueError('Invalid value')
```

```
            self.radius = radius
            self.diametr = diametr

        def __str__(self):
            return (f'Circle: П е р и м е т р {self.perimeter}, п л о щ
а д ь {self.area}, ц в е т  ф и г у р ы {self.color}, '
                    f' р а д и у с  {self.radius}, д и а м е т р
{self.diametr}.')

        def __add__(self):
            return self.perimeter + self.area

        def __eq__(self, other):
            return True if self.radius == other.radius else False


    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
{p_object.__class__.__name__}')

        def print_colors(self):
            for i, poly in enumerate(self, start=1):
                print(f"{i} м н о г о у г о л ь н и к : {poly.color}")

        def print_count(self):
            print(len(self))


    class CircleList(list):
        def __init__(self, name):
            super().__init__()
            self.name = name

        def extend(self, iterable):
            for item in iterable:
                if isinstance(item, Circle):
                    self.append(item)

        def print_colors(self):
            for i, el in enumerate(self, start=1):
                print(f"{i} о к р у ж н о с т ь : {el.color}")

        def total_area(self):
            total_area_sum = sum(el.area for el in self)
          print(total_area_sum)
```