МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Информационные технологии»

Тема: Парадигмы программирования

Студентка гр. 3342 _____ Антипина В.А.

Преподаватель _____ Иванов Д.И.

Санкт-Петербург

2024

Цель работы

Изучение парадигмы программирования на языке Python, основ объектно-ориентированного программирования.

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.

Основные теоретические положения

Объектно-ориентированное программирование - методология (парадигма) программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. В программе при этом в качестве основных логических конструктивных элементов используются объекты, а не алгоритмы.

В ООП центральными являются понятия класса и объекта:

- Класс (англ. Class): абстракция реального мира (обобщенный шаблон), специальный тип данных; класс описывает свойства и методы, которые могут быть доступны у подобных объектов;
- Объект (англ. Object) (экземпляр класса, англ. Class Instance): частный случай класса.

Каждый класс содержит и описывает поля и методы:

- Поле (англ. Data Member / Variable / Field): переменная, привязанная к классу;
- Метод (англ. Method): действие (функция), которую можно проводить над классом.

Объектно-ориентированная парадигма программирования включает 3 основных принципа (свойства):

- Инкапсуляция (англ. Encapsulation)

Концепция инкапсуляции вращается вокруг принципа, гласящего, что внутренние данные объекта не должны быть напрямую доступны через экземпляр объекта. Вместо этого данные класса определяются как закрытые. Если вызывающий код желает изменить состояние объекта, то он должен делать это непрямо через открытые методы.

- Наследование (англ. Inheritance)

Наследование — аспект ООП, облегчающий повторное использование кода. Принцип наследования в языке программирования позволяет строить новые определения классов на основе существующих. Наследование позволяет расширять поведение базового класса (родительского или суперкласса), наследуя его основную функциональность в производном подклассе (дочернем классе или подклассе).

- Полиморфизм (англ. Polymorphism)

Полиморфизм обозначает способность языка трактовать связанные объекты в сходной манере. В частности, этот принцип ООП позволяет базовому классу определять набор членов, которые доступны всем наследникам.

Можно выяснить, является ли объект представителем класса, используя функцию isinstance(obj, class).

Выполнение работы

Был описан базовый класс Figure. В нём был определён метод __init__(конструктор, на вход которому подаются периметр, площадь и цвет фигуры). Если периметр и площадь — целые положительные числа, а цвет — строка из предложеннных — „r“, „g“, „b“ — определялись соответствующие поля. В противном случае вызывалась ошибка значения.

Класс Polygon — наследник класса Figure. У него переопределён конструктор (он унаследован от родительского класса с помощью функции super(), но дополнен проверкой новых значений многоугольника — числа углов, наличия равных сторон и самого большого угла — на соответствие условию задания и определением полей angle_count, equilateral, biggest_angle, если данные корректны. В противном случае вызывается ValueError). В классе также переопределены методы __str__, __add__, __eq__. Первый метод возвращает строку нужного формата, второй возвращает сумму значений полей perimeter и area объекта класса, а третий получает на вход не только сам объект, но и другой объект, с которым нужно сравнить исходный. Тогда, если значения полей периметра и площади у них совпадают и количество углов одинаковое, возвращается True, в противном случае — False.

Класс Circle создаётся аналогичным образом, но у него в конструкторе проверяются радиус и диаметр на соответствие условиям задачи и, если такого нет, вызывается ошибка ValueError, иначе определяются поля radius, diametr. Методы __str__, __add__, __eq__ здесь также переопределяются (это происходит аналогично описанному выше случаю).

Класс PolygonList наследуется от класса list. Конструктор наследуется у родительского класса и дополняется полем name. Переопределён метод append: ему на вход подаётся объект, который нужно проверить на принадлежность к классу Polygon. Если объект — многоугольник, он добавляется в массив (для этого вызвается метод из родительского класса). В противном случае возникает сообщение об ошибке. В методе print_colors в цикле перебираются все

элементы списка и каждый раз возвращается форматная строка. Метод print_count возвращет длину списка, которая определяется с помощью функции len.

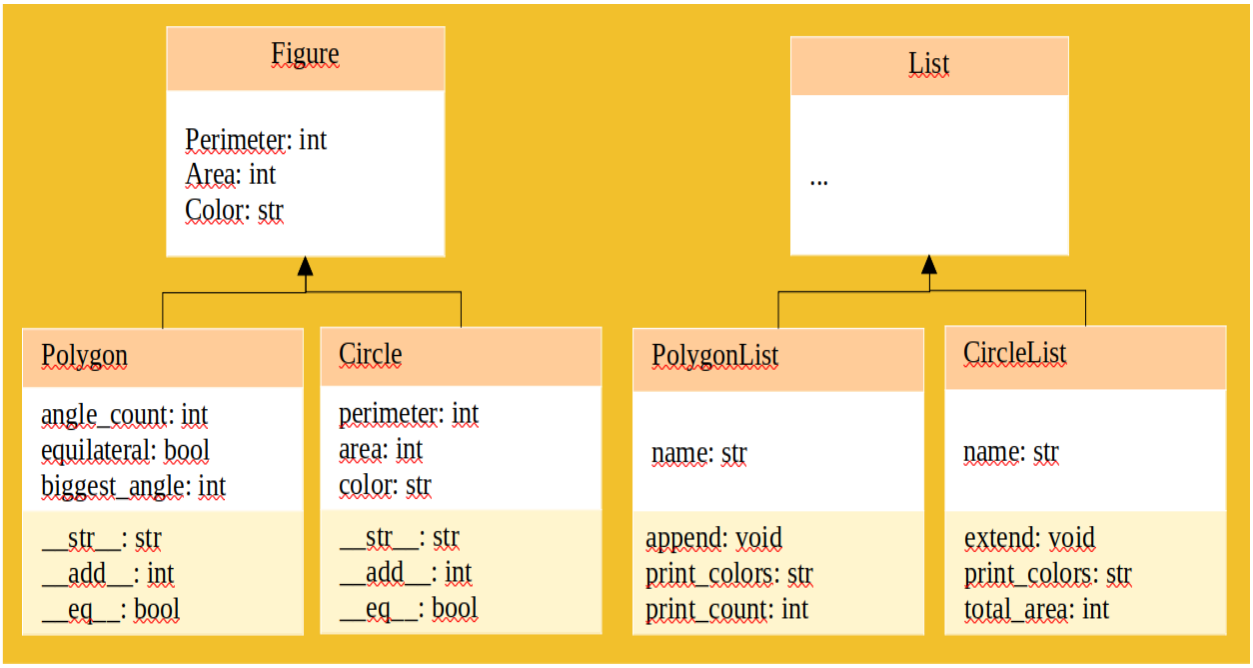

Рисунок 1 - Иерархия классов

Класс CircleList также наследуется от класса list и переопределение конструктора у него происходит аналогичным образом. В этом классе также переопределён метод extend, которому на вход подаётся список. Каждый элемент жтого списка проверяется на принадлежность к классу Circle. Если элемент — круг, он добавляется в массив (в виде списка из одного элемента, потому что этот метод добавляет список в список). Метод print_colors аналогичен описанному в предыдущем абзаце, а метод total_area перебирает все элементы списка и прибавляет значения полей area этих экземпляров класса Circle к сумматору, который функция и возвращает.

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	10,25,'g' 10,25,'g',4, True, 90 10,25,'g',4, True, 90 13, 13,'r', 2, 4 13, 13,'g', 2, 4 Polygon polygon polygon2 Circle) [circle, circle2]	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	Корректно
2.	-1,4,'g' 15,-5,'g' 10,25,-1 17,20,1 4,5,'a' 'a',25,'b' 10,'a','g' 0,25,'g'		OK OK OK OK OK OK OK OK

	10,0,'g'		OK

Выводы

Были изучены основы объектно-ориентированного программирования на языке Python. Была реализована программа, которая определяет класс фигур, его классы-наследники Circle, Rectangle, классы-наследники класса list CircleList и RectangleList и их методы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Antipina_Veronika_lb1.py

```
class Figure:

    def __init__(self,perimeter,area,color):
        if   (isinstance(perimeter,int)   and   perimeter>0   and
isinstance(area,int) and area>0 and color in ['r','g','b'])==False:
            raise ValueError('Invalid value')
        else:
            self.perimeter = perimeter
            self.area = area
            self.color = color

class Polygon(Figure):
    def
__init__(self,perimeter,area,color,angle_count,equilateral,biggest_ang
le):
        super().__init__(perimeter,area,color)

        if  (isinstance(angle_count,int)  and  angle_count>2  and
isinstance(equilateral,bool)   and   isinstance(biggest_angle,int)   and
biggest_angle>0)==False:
            raise ValueError("Invalid Value")
        else:
            self.angle_count = angle_count
            self.equilateral = equilateral
            self.biggest_angle = biggest_angle

    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,comp):
        if(self.perimeter  ==  comp.perimeter  and  self.area  ==
comp.area and self.angle_count == comp.angle_count):
            return True
        else:
            return False

class Circle(Figure):
    def __init__(self,perimeter,area,color,radius,diametr):
        super().__init__(perimeter,area,color)
        if     (isinstance(radius,int)     and     radius>0     and
isinstance(diametr,int) and diametr==2*radius)==False:
            raise ValueError("Invalid value")
        self.radius = radius
        self.diametr = diametr

    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,comp):
            if self.radius == comp.radius:
                return True
            else:
                return 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:
                t = type(p_object)
                raise TypeError(f'Invalid type <{t}>')

        def print_colors(self):
            for i in range(1,len(self)+1):
                print(f"{i} многоугольник: {self[i-1].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 new in iterable:
                if isinstance(new,Circle):
                    super().extend([new])


        def print_colors(self):
            for i in range(1,len(self)+1):
                print(f"{i} окружность: {self[i-1].color}")

        def total_area(self):
            s = 0
            for x in self:
                s+=x.area
            print(s)
```