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 _____ Белаид Фарук

Преподаватель _____ Иванов Д. В.

Санкт-Петербург
2024

Цель работы

Рассмотреть принципы объектно-ориентированного программирования, такие как наследование, инкапсуляция и полиморфизм, на примере создания собственной иерархии классов с применением переопределения методов и наследования классов.

Задание

Даны фигуры в двумерном пространстве.

1. Базовый класс - фигура **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'.

2. Многоугольник – 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:

1. 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>.
2. Метод **__add__():** Сложение площади и периметра многоугольника. Возвращает число, полученное при сложении площади и периметра многоугольника.
3. 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.

3. Окружность – 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:

1. Метод **__str__():** Преобразование к строке вида: Circle: Периметр <периметр>, площадь <площадь>, цвет фигуры <цвет фигуры>, радиус <радиус>, диаметр <диаметр>.
2. Метод **__add__():** Сложение площади и периметра окружности. Возвращает число, полученное при сложении площади и периметра окружности.

3. 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:

1. 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:

1. 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>.
2. Метод print_colors(): Вывести цвета всех многоугольников в виде строки (нумерация начинается с 1)
3. Метод print_count(): Вывести количество многоугольников в списке.

2. 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:

1. 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.
2. Метод print_colors(): Вывести цвета всех окружностей в виде строки (нумерация начинается с 1)

3. Метод total_area(): Посчитать и вывести общую площадь всех окружностей.

Выполнение работы

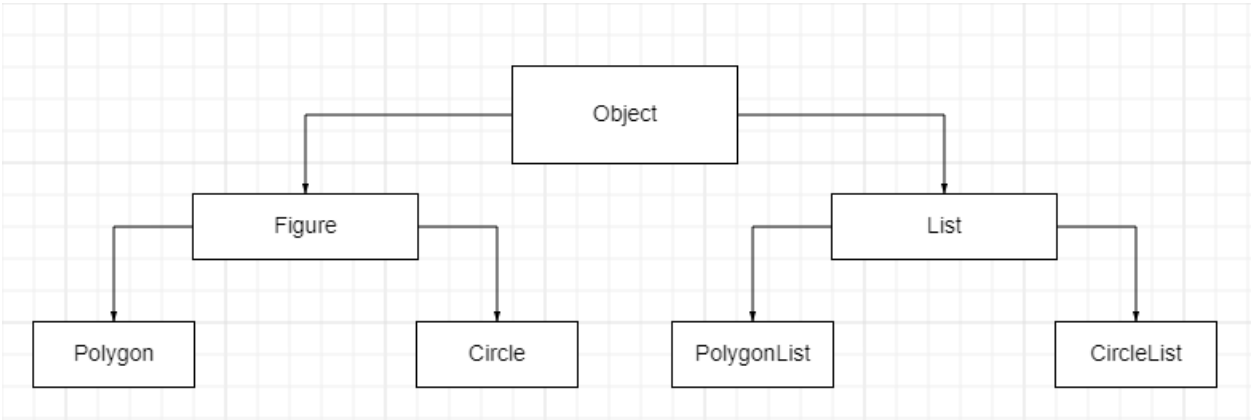
Для проверки типа подаваемых в конструктор класса данных используются вспомогательные функции:

1) *int_positive_check(item)* – возвращает *True*, если в качестве аргумента было передано целое положительное число, и *False* в противном случае.

2) *value_check(value, condition)* – возвращает *value*, если оно удовлетворяет условию *condition*. В противном случае вызывает исключение *ValueError* с сообщением *“Invalid value”*. В качестве условия могут передаваться любые функции, возвращающие *True* или *False*. Например, *int_positive_check()* или какие-либо лямбда-выражения.

Перед присваиванием полю объекта класса того или иного значения всегда выполняется проверка соответствия этого значения конкретным условиям.

Были реализованы классы *Figure*, *Polygon*, *Circle*, *PolygonList* и *CircleList*. Их иерархию можно представить в следующем виде:



В классе *Figure* был определен метод *__init__(),* позволяющий задавать значения полям *perimeter*, *area* и *color* при создания объекта класса.

В классе *Polygon*, наследуемым от *Figure*, переопределяется метод инициализации. В добавок к полям родительского класса, объекты-многоугольники также имеют поля *angle_count*, *equilateral* и *biggest_angle*. Для задания полей *perimeter*, *area* и *color* используется вызов конструктора родительского класса (с помощью *super().__init__()).*

Также переопределяются следующие методы:

1) *__str__()* – возвращает строку вида: *“Polygon: Периметр <периметр>, площадь <площадь>, цвет фигуры <цвет фигуры>, количество углов <кол-во углов>, равносторонний <равносторонний>, самый большой угол <самый большой угол>.”*
2) *__add__()* - возвращает число, полученное при сложении площади и периметра многоугольника.
3) *__eq__()* - возвращает True, если два объекта класса равны и False иначе. Два объекта типа Polygon равны, если равны их периметры, площади и количество углов.

В классе *Polygon*, наследуемым от *Figure*, переопределяется метод инициализации. В добавок к полям родительского класса, объекты-круги также имеют поля *radius* и *diametr.* Для задания полей *perimeter*, *area* и *color* используется вызов конструктора родительского класса (с помощью *super().__init__()).*

Также переопределяются следующие методы:

1) *__str__()* – возвращает строку вида: *“ Circle: Периметр <периметр>, площадь <площадь>, цвет фигуры <цвет фигуры>, радиус <радиус>, диаметр <диаметр>.”*
2) *__add__()* - возвращает число, полученное при сложении площади и периметра окружности.
3) *__eq__()* - возвращает *True*, если два объекта класса равны и *False* иначе. Два объекта типа *Circle* равны, если равны их радиусы.

Класс *PolygonList* является наследником стандартного класса list. Он переопределяет метод *append(),* чтобы проверять, является ли добавляемый объект экземпляром класса *Polygon*. Если это так, то вызывается метод *append()* родительского класса *list* для добавления объекта в список, иначе выбрасывается исключение *TypeError*.

Также класс *PolygonList* определяет методы *print_colors()* и *print_count(),* которые выводят цвета многоугольников из списка и общее количество многоугольников соответственно.

Аргумент name передается в конструктор списка с помощью вызова конструктора родительского класса *list*. Таким образом, объект класса *PolygonList* является списком экземпляров класса *Polygon* с дополнительным полем *name,* которое задается при создании объекта.

Класс *CircleList* наследуется от встроенного класса *list* и представляет собой список объектов класса *Circle*. Класс содержит следующие методы:

1) Метод *__init__* является конструктором и вызывается при создании нового объекта. Он инициализирует пустой список и сохраняет переданное имя в поле *name*.
2) Метод *extend* принимает итерируемый объект *iterable* и добавляет в список все элементы, являющиеся объектами класса *Circle*.
3) Метод *print_colors* выводит на экран цвет каждой окружности в списке.
4) Метод *total_area* вычисляет и выводит на экран суммарную площадь всех окружностей в списке.

Метод *__str__()* будет автоматически вызываться в случае передачи объекта Polygon или Circle в функцию print() либо же при прямом вызове функции str(). *Метод __add__()* можно будет вызвать только вручную, так как для того, чтобы он работал через операнд “+”, необходимо, чтобы он принимал в качестве аргумента два объекта (*self* и *other*). В нашем же случае, он принимает лишь один аргумент, а значит не будет работать через операнд.

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.	*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*	*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*	Ответ верный

	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()*		

Вывод

В ходе выполнения лабораторной работы при реализации классов были применены основные принципы объектно-ориентированного программирования, а именно наследование, полиморфизм и инкапсуляция. Были изучены инструменты переопределения методов, использования внутри них одноименных методов родительского класса (с помощью super()), а также создания некоторой защиты от ввода пользователем неправильных данных, способных нарушить ход выполнения программы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
def int_positive_check(item):
    if isinstance(item, int) and item > 0:
        return True

def value_check(value, condition):
    if condition(value):
        return value
    else:
        raise ValueError("Invalid value")


class Figure:
    def __init__(self, perimeter, area, color):
        self.perimeter = value_check(perimeter, int_positive_check)
        self.area = value_check(area, int_positive_check)
        self.color = value_check(color, lambda x: x in ('r', 'b',
'g'))


class Polygon(Figure):  # Наследуется от класса Figure
    def __init__(self, perimeter, area, color, angle_count,
equilateral, biggest_angle):
        super().__init__(perimeter, area, color)
        self.angle_count = value_check(angle_count, lambda x:
int_positive_check(x) and x > 2)
        self.equilateral = value_check(equilateral, lambda x:
isinstance(x, bool))
        self.biggest_angle = value_check(biggest_angle,
int_positive_check)

    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):
        return self.perimeter == other.perimeter and self.area ==
other.area and self.angle_count == other.angle_count


class Circle(Figure):  # Наследуется от класса Figure
    def __init__(self, perimeter, area, color, radius, diametr):
        super().__init__(perimeter, area, color)
        self.radius = value_check(radius, int_positive_check)
        self.diametr = value_check(diametr, lambda x:
int_positive_check(x) and x == self.radius*2)

    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):
        return self.radius == other.radius


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 {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):
        for i in iterable:
            if isinstance(i, Circle):
                super().append(i)

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

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