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. 3344		**Сьомак Д.А.**
Преподаватель		**Иванов Д.В.**

Санкт-Петербург

2023

Цель работы

Получение навыков использования объектно-ориентированного подхода программирования в языке Python.

Задание

Вариант 1. 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. Иерархия описанных классов:

Figure list

Poligon Circle Poligonlist Circlelist

2.Переопределённые методы:

__init__() - метод инициализации класса.

__add__() - метод, возвращающий результат сложения двух переданных объекктов.

__str__() - метод, возвращающий строчное представление заданного объекта.

__eq__() - метод сравнения объектов разных классов.

append() - метод, добавляющий в список только объекты класса Poligon, при получении объектов других классов выводит TypeError.

extend() - метод, добавляющий в список только объекты класса Circle.

3.

Метод __str__ будет использован в случае обращению к объекту, как к строке. Метод вернёт строковое представление информации об объект.

Метод __add__ будет использован в случае попытки получения суммы двух объектов. Метод вернёт сумму объектов.

4.

Переопределённые методы класса list для классов Poligonlist и Circlelist будут работать, так как эти классы наследуются от класса list. При переопределении методы новых классов сохраняют функционал методов родительского класса, при это добавляется возможность добавить необходимую логику, такую как проверка объекта на принадлежность к определённому классу, как в данном случае (append, extend).

Исходный код см. в приложении А

Выводы

Были получены практические навыки использования объектно-ориентированного подхода программирования путём написания программы на языке Python. Были изучены процессы наследования классов и переопределения методов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Somak_Demid_lb1.py

```
class Figure:

    def __init__(self,perimeter,area,color):

        if not isinstance(perimeter, int) or perimeter < 1:
            raise ValueError('Invalid value')
        self.perimeter = perimeter

        if not isinstance(area, int) or area < 1:
            raise ValueError('Invalid value')
        self.area = area

        if color not in ['r', 'g', 'b']:
            raise ValueError('Invalid value')
        self.color = color


class Polygon(Figure):

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

        if not isinstance(angle_count, int) or angle_count < 3:
            raise ValueError('Invalid value')
        self.angle_count = angle_count

        if not isinstance(equilateral, bool):
            raise ValueError('Invalid value')
        self.equilateral = equilateral

        if not isinstance(biggest_angle, int) or biggest_angle < 1:
            raise ValueError('Invalid value')
        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,other):
```

```
        if self.perimeter == other.perimeter and self.area == other.area
and self.angle_count == other.angle_count:
            return True
        else:
            return False

class Circle(Figure):

    def __init__(self,perimeter,area,color,radius, diametr):
        super().__init__(perimeter, area,color)

        if not isinstance(radius, int) or radius < 1:
            raise ValueError('Invalid value')
        self.radius = radius

        if not isinstance(diametr, int) or diametr != 2 * radius:
            raise ValueError('Invalid value')
        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.perimeter + self.area

    def __eq__(self,other):
        if self.radius == other.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:
            raise TypeError(f"invalid type {type(p_object)}")

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



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

```
class CircleList(list):

    def __init__(self, name):
        super().__init__()
        self.name = name

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

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



    def total_area(self):
        summ = 0
        for item in list(self):
            summ += item.area

        print(summ)
```