**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:** Лабораторная работа №1. Парадигмы программирования

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Пименов П.В. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Изучить, какие бывают парадигмы программирования. Создать программу на языке Python, реализующую несколько классов фигур и списков для них.

## Задание

Вариант 1. Даны фигуры в двумерном пространстве. Базовый класс – фигура Figure, многоугольник – Polygon, окружность – Circle:

* 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'.
* class Polygon
  + 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.
* class Circle:
  + 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:

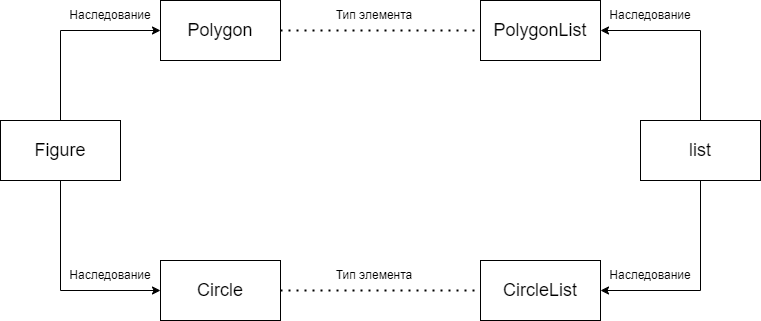
* class PolygonList – список многоугольников – наследуется от класса list.
  + Конструктор:
    - Вызвать конструктор базового класса.
    - Передать в конструктор строку name и присвоить её полю name созданного объекта
  + Необходимо реализовать следующие методы:
    - Метод append(p\_object): Переопределение 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.
* 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.

В отчете укажите:

1. Изображение иерархии описанных вами классов.
2. Методы, которые вы переопределили (в том числе методы класса object).
3. В каких случаях будут использованы методы \_\_str\_\_() и \_\_add\_\_().
4. Будут ли работать переопределенные методы класса list для PolygonList и CircleList? Объясните почему и приведите примеры.

## Выполнение работы

Указанные в задании классы, их методы и поля успешно реализованы.

Рисунок 1 – Изображение иерархии классов

Были переопределены следующие методы:

* класс Figure
  + метод \_\_init\_\_ – реализован конструктор по заданию
* класс Polygon
  + метод \_\_init\_\_ – реализован конструктор по заданию
  + метод \_\_str\_\_ – переопределено строковое представление объекта класса
  + метод \_\_add\_\_ – переопределена операция сложения
  + метод \_\_eq\_\_ – переопределена операция сравнения
* класс Circle
  + метод \_\_init\_\_ – реализован конструктор по заданию
  + метод \_\_str\_\_ – переопределено строковое представление объекта класса
  + метод \_\_add\_\_ – переопределена операция сложения
  + метод \_\_eq\_\_ – переопределена операция сравнения
* класс PolygonList
  + метод \_\_init\_\_ – реализован конструктор по заданию
  + метод append – переопределена операция добавления элемента в список согласно условию задания
* класс CircleList
  + метод \_\_init\_\_ – реализован конструктор по заданию
  + метод extend – переопределена операция добавления элементов коллекции в список согласно условию задания

Метод \_\_str\_\_ будет работать, когда будет когда будет нужно строковое представление объекта класса, например, при вызове print(circle), если circle – объект класса Circle. Метод \_\_add\_\_ будет работать, когда была вызвана операция сложения двух объектов. Однако в данном случае, будет корректен только вызов вида circle.\_\_add\_\_(), где circle – объект класса Circle, поскольку по условию задания метод \_\_add\_\_ не должен принимать второй операнд. Переопределенные методы классов PolygonList и CircleList будут работать, поскольку переопределенные методы классов работают «сверху-вниз»: сначала вызывается переопределенный метод класса-наследника, потом класса-родителя, потом класса-родителя-родителя и т. д. Примеры использования см. в п. 1, 2 части «Тестирование».

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. | circle = Circle(123, 123, 'r', 123, 246)  circle\_list = CircleList("list")  circle\_list.extend([circle, circle, 1, '1', True])  circle\_list.print\_colors() | 1 окружность: r  2 окружность: r | Программа работает корректно |
| 2. | polygon = Polygon(8, 4, 'g', 4, True, 90)  polygon\_list = PolygonList("list")  polygon\_list.append(polygon)  polygon\_list.print\_colors() | 1 многоугольник: g | Программа работает корректно |
| 3. | circle = Circle(123, 123, 'r', 123, 246)  circle\_list = CircleList("list")  circle\_list.extend([circle, circle, 1, '1', True])  circle\_list.total\_area() | 246 | Программа работает корректно |
| 4. | circle = Circle(123, 123, 'r', 123, 246)  print(circle) | Circle: Периметр 123, площадь 123, цвет фигуры r, радиус 123, диаметр 246. | Программа работает корректно |

## Выводы

Изучено, какие бывают парадигмы программирования. Создана программа на языке Python, реализующую несколько классов фигур и списков для них.

# Приложение А Исходный код программы

Название файла: main.py

class Figure:

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

if not isinstance(perimeter, int) or not isinstance(area, int) or not isinstance(color, str):

raise ValueError('Invalid value')

if perimeter > 0 and area > 0 and color in {'r', 'g', 'b'}:

self.perimeter = perimeter

self.area = area

self.color = color

else:

raise ValueError('Invalid value')

class Polygon(Figure):

def \_\_init\_\_(self, perimeter, area, color, angle\_count, equilateral, biggest\_angle):

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

if not isinstance(angle\_count, int) or not isinstance(equilateral, bool) or not isinstance(biggest\_angle, int):

raise ValueError('Invalid value')

if angle\_count > 2 and biggest\_angle > 0:

self.angle\_count = angle\_count

self.equilateral = equilateral

self.biggest\_angle = biggest\_angle

else:

raise ValueError('Invalid value')

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):

if isinstance(other, Polygon):

return self.perimeter == other.perimeter and self.area == other.area and self.angle\_count == other.angle\_count

else:

raise NotImplementedError(f'Cannot compare Polygon to {type(other)}')

class Circle(Figure):

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

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

if not isinstance(radius, int) or not isinstance(diametr, int):

raise ValueError('Invalid value')

if radius > 0 and diametr > 0 and diametr == 2 \* radius:

self.radius = radius

self.diametr = diametr

else:

raise ValueError('Invalid value')

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):

if isinstance(other, Circle):

return self.perimeter == other.perimeter and self.area == other.area and self.radius == other.radius

else:

raise NotImplementedError(f'Cannot compare Circle to {type(other)}')

class PolygonList(list):

def \_\_init\_\_(self, name):

super().\_\_init\_\_()

self.name = name

def append(self, \_\_object):

if isinstance(\_\_object, Polygon):

super().append(\_\_object)

else:

raise TypeError(f'Invalid type {type(\_\_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 item in \_\_iterable:

if isinstance(item, Circle):

super().append(item)

def print\_colors(self):

for i in range(len(self)):

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

def total\_area(self):

print(sum([x.area for x in self]))