**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

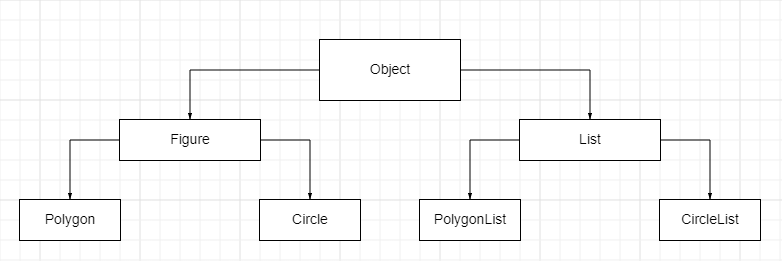
Для проверки типа подаваемых в конструктор класса данных используются вспомогательные функции:

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*  *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()* | *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 |

## Выводы

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)