**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Наследование

|  |  |  |
| --- | --- | --- |
| Студент гр. 7381 |  | Ильясов А. В. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2019

**Цель работы**

Изучение принципов наследования классов в С++.

**Задание**

Необходимо спроектировать систему классов для моделирования геометрических фигур (в соответствии с полученным индивидуальным заданием). Задание предполагает использование виртуальных функций в иерархии наследования, проектирование и использование абстрактного базового класса. Разработанные классы должны быть наследниками абстрактного класса Shape, содержащего методы для перемещения в указанные координаты, поворота на заданный угол, масштабирования на заданный коэффициент, установки и получения цвета, а также оператор вывода в поток.

﻿﻿Необходимо также обеспечить однозначную идентификацию каждого объекта.

Решение должно содержать:

* условие задания;
* UML диаграмму разработанных классов;
* текстовое обоснование проектных решений;
* реализацию классов на языке С++.

**Условие задания**

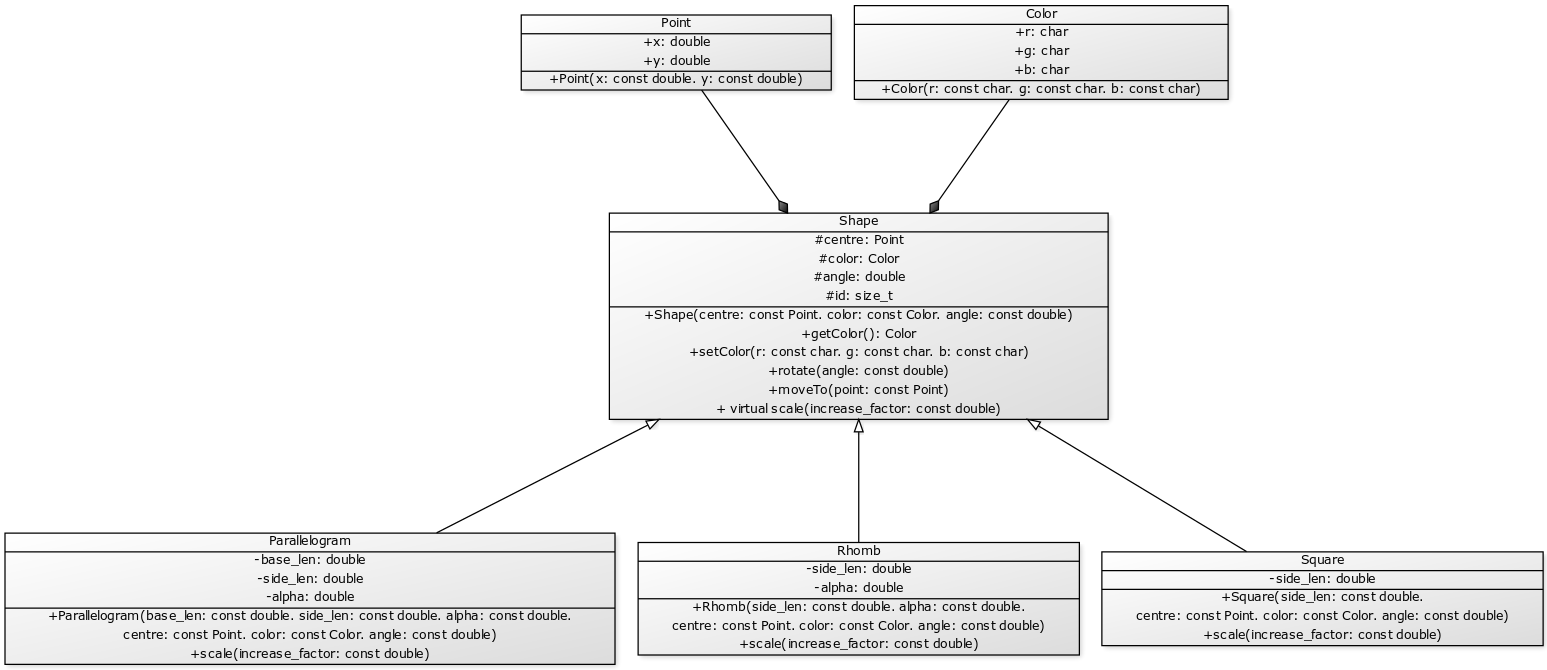
Вариант 7: фигуры, которые необходимо реализовать: квадрат, параллелограмм, ромб.

**Обоснование проектных решений**

1. Для большего удобства чтения кода и работы с ним, были введены следующие структуры: Point и Color, хранящие координаты в плоскости и компоненты цветовой модели rgb соответственно. Также в этих структурах были объявлены конструкторы и перегружены операторы "<<" для удобства вывода.
2. В абстрактном классе Shape, который является базовым в данной л. р. Хранятся общие для всех фигур поля: координаты центра фигуры, цвет, угол, между основанием и осью абсцисс, а также идентификатор объектов. Был перегружен оператор вывода для вывода информации об объектах, которая хранится в классе Shape. В классе реализованы функции-методы получения и установки цвета, поворота на заданный угол, перемещения в заданную точку и виртуальная функция масштабирования, которая реализована в каждом наследуемом классе по-своему.
3. Для однозначного задания параллелограмма в пространстве, помимо координат центра и угла между осью и основанием, необходимы длина основания и боковой стороны, а также наименьший угол между ними. Следовательно, в класс Parallelogram, наследуемый от Shape, нужно добавить соответствующие поля.
4. Так как ромб является частным случаем параллелограмма с равными по длине сторонами, в класс Rhomb было добавлено только одно поле для длины боковой стороны и меньший угол между сторонами.
5. Квадрат – ромб, с прямым углом между ребрами, следовательно, нужно добавить только одно поле с длиной ребра.
6. Функция-метод масштабирования, исходя из задания всех фигур, просто производит умножение длин сторон этих фигур на заданный коэффициент.

**UML-диаграмма**

Ниже представлена UML-диаграмма, для отображения организации структур и классов.

Рисунок 1 - UML-диаграмма данного проекта.

**Выводы.**

В ходе выполнения данной лабораторной работы, был изучен один их механизмов ООП – наследование. Также были повышены навыки по работе с ЯВУ С++ и принципами ООП, в частности.

Приложение А

основной код программы main.cpp

#include "Parallelogram.hpp"

#include "Rhomb.hpp"

#include "Square.hpp"

int main() {

Parallelogram shape1(5, 3, 60);

Rhomb shape2(10, 45, Point(-2.5, 5));

Square shape3(20, Point(5, -5.8), Color(127, 127, 127), 30);

std::cout << shape1 << std::endl <<

shape2 << std::endl <<

shape3 << std::endl;

shape1.setColor(127, 0, 45);

std::cout << "New shape1's color: " << shape1.getColor() << std::endl << std::endl;

shape1.moveTo(Point(50, -10.5));

shape2.rotate(-90);

shape3.scale(1.9);

std::cout << shape1 << std::endl <<

shape2 << std::endl <<

shape3 << std::endl;

return 0;

}

Приложение Б

заголовочный файл класса shape

и его реализация

#ifndef SHAPE\_HPP

#define SHAPE\_HPP

#include <iostream>

size\_t ID = 0;

struct Point {

double x;

double y;

Point(const double x, const double y): x(x), y(y) {}

friend Point operator += (Point &P1, const Point &P2) {

return Point(P1.x = P1.x + P2.x, P1.y = P1.y + P2.y);

}

friend std::ostream& operator <<(std::ostream &out, const Point obj) {

out << "(" << obj.x << ", " << obj.y << ")";

return out;

}

};

struct Color {

char r;

char g;

char b;

Color(const char r, const char g, const char b): r(r), g(g), b(b) {}

friend std::ostream& operator <<(std::ostream &out, const Color obj) {

out << "(" << int(obj.r) << ", " << int(obj.g) << ", " << int(obj.b) << ")";

return out;

}

};

class Shape {

protected:

Point centre;

Color color;

double angle;

size\_t id;

public:

Shape(const Point centre = Point(0, 0),

const Color color = Color(0, 0, 0),

const double angle = 0)

: centre(centre), color(color), angle(angle), id(ID++) {};

~Shape() {};

Color getColor() { return color; }

void setColor(const char r, const char g, const char b) {

this->color = Color(r, g, b);

}

void rotate(const double angle) {

this->angle = angle;

}

void moveTo(const Point point) {

centre += point;

}

virtual void scale(const double increase\_factor) = 0;

friend std::ostream & operator << (std::ostream &out, const Shape &obj) {

out << "ID: " << obj.id << std::endl <<

"Color: " << obj.color << std::endl <<

"Center coordinates: " << obj.centre << std::endl <<

"Angle between base and X axis: " << obj.angle << std::endl;

return out;

}

};

#endif

Приложение В

заголовочный файл класса parallelogram

и его реализация

#ifndef PARALLELOGRAM\_HPP

#define PARALLELOGRAM\_HPP

#include "Shape.hpp"

class Parallelogram : public Shape {

double base\_len;

double side\_len;

double alpha;

public:

Parallelogram(const double base\_len, const double side\_len, const double alpha,

const Point centre = Point(0, 0), const Color color = Color(0, 0, 0), const double angle = 0)

: Shape(centre, color, angle), base\_len(base\_len), side\_len(side\_len), alpha(alpha) {};

~Parallelogram() {};

void scale(const double increase\_factor) override {

base\_len \*= increase\_factor;

side\_len \*= increase\_factor;

}

friend std::ostream & operator << (std::ostream &out, const Parallelogram &obj) {

out << dynamic\_cast<const Shape &>(obj) <<

"Shape type: Parallelogram" << std::endl <<

"Base length: " << obj.base\_len << std::endl <<

"Side length: " << obj.side\_len << std::endl <<

"Alpha: " << obj.alpha << std::endl;

return out;

}

};

#endif

Приложение Г

заголовочный файл класса Rhomb

и его реализация

#ifndef RHOMB\_HPP

#define RHOMB\_HPP

#include "Shape.hpp"

class Rhomb : public Shape {

double side\_len;

double alpha;

public:

Rhomb(const double side\_len, const double alpha,

const Point centre = Point(0, 0), const Color color = Color(0, 0, 0), const double angle = 0)

: Shape(centre, color, angle), side\_len(side\_len), alpha(alpha) {};

~Rhomb() {};

void scale(const double increase\_factor) override {

side\_len \*= increase\_factor;

}

friend std::ostream & operator << (std::ostream &out, const Rhomb &obj) {

out << dynamic\_cast<const Shape &>(obj) <<

"Shape type: Rhomb" << std::endl <<

"Side length: " << obj.side\_len << std::endl <<

"Alpha: " << obj.alpha << std::endl;

return out;

}

};

#endif

Приложение д

заголовочный файл класса square

и его реализация

#ifndef SQUARE\_HPP

#define SQUARE\_HPP

#include "Shape.hpp"

class Square : public Shape {

double side\_len;

public:

Square(const double side\_len,

const Point centre = Point(0, 0), const Color color = Color(0, 0, 0), const double angle = 0)

: Shape(centre, color, angle), side\_len(side\_len) {};

~Square() {};

void scale(const double increase\_factor) override {

side\_len \*= increase\_factor;

}

friend std::ostream & operator << (std::ostream &out, const Square &obj) {

out << dynamic\_cast<const Shape &>(obj) <<

"Shape type: Square" << std::endl <<

"Side length: " << obj.side\_len << std::endl;

return out;

}

};

#endif