**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

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

**Тема: «Наследование»**

|  |  |
| --- | --- |
| Студентка гр. 7381 | Машина Ю. Д. |
| Преподаватель | Жангиров Т. Р. |

Санкт-Петербург

2019

**Цель работы.**

Ознакомиться с понятиями наследование, полиморфизм, абстрактный класс, изучить виртуальные функции, принцип их работы, способ организации в памяти, раннее и позднее связывания в языке C++. В соответствии с индивидуальным заданием разработать систему классов для представления геометрических фигур.

**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:

1. Условие задания;
2. UML диаграмму разработанных классов;
3. Текстовое обоснование проектных решений;
4. Реализацию классов на языке С++.

**Индивидуализация.**

Вариант 12 – реализовать систему классов для фигур:

1. Квадрат;
2. Правильный пятиугольник;
3. Пятиконечная звезда.

**Обоснование проектных решений.**

Для представления цвета — отдельная структура Color, представляющую из себя структуру для представления 3-байтного цвета палитры RGB.

Для представления точки координаты — структура Point, представляющую из себя структуру c полями координат x и y.

Базовым классом для всех фигур является класс Shape, используемый для представления фигур. Хранит в себе координаты центра фигуры – абсциссу и ординату. Перемещение фигуры представляет из себя смену координат центра фигуры, поэтому метод перемещения фигуры объявлен в этом классе, необходимости переопределять этот метод в классах-наследниках нет, поэтому он не виртуальный.

Так же этот класс содержит в себе угол поворота (в радианах, по умолчанию ) от оси . Для поворота фигуры достаточно прибавить к этому углу необходимую величину поворота, поэтому поворот так же не виртуальный метод.

Здесь же хранится цвет фигуры, к которому так же есть доступ, потребность в виртуализации отсутствует.

Метод масштабирования сделан чисто виртуальным, поскольку его реализация зависит от способа организации фигуры в памяти (при ленивых вычислениях можно было бы использовать переменную для хранения масштаба объекта (по умолчанию ), но любые манипуляции связанные с вычислением каких-то метрик фигуры будут приводить к постоянному домножению на масштабный коэффициент, что может, вероятно, привести к ошибкам).

Для перегрузки оператора вывода фигуры в поток оператор объявлен во всех классах дружественной функцией, чтобы можно было вывести значения защищённых и приватных полей.

Класс Square, используемый для представления разностороннего треугольника, наследуется от Shape. Характеризуется длиной стороны и центром.

Класс Pentagon является наследником класса Shape, используется для представления правильного пятиугольника. Характеризуется радиусом окружности, в которую вписан.

Класс Fivepointed\_star используется для представления пятиконечной звезды, наследуется от Pentagon, характеризуется внутренним радиусом, помимо внешнего, предоставленного Pentagon.

**UML диаграмма разработанных классов.**

UML диаграмма разработанных классов представлена в приложении А и в соседнем документе (UML.png).

**Реализация классов на языке С++.**

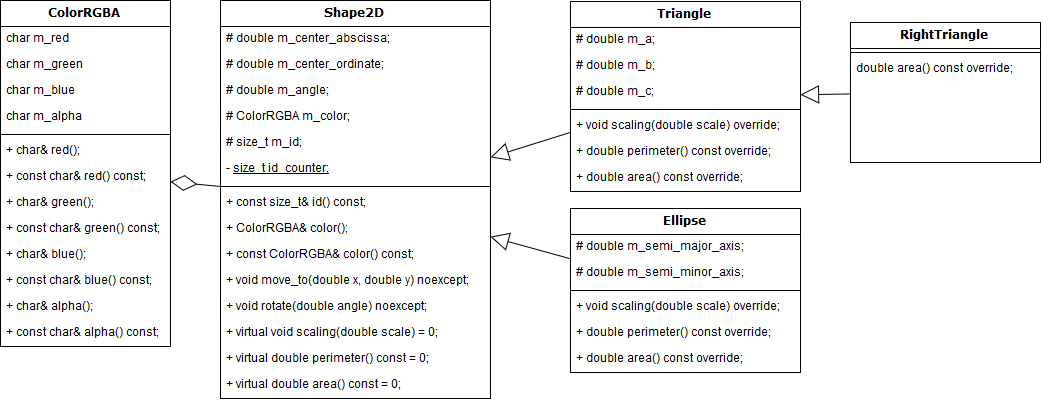
Реализация классов представлена в приложении Б.

**Выводы.**

В ходе выполнения лабораторной работы была спроектирована система классов для работы с геометрическими фигурами в соответствии с индивидуальным заданием. В иерархии наследования были использованы виртуальные функции, базовый класс при этом является виртуальным (класс называется виртуальным, если содержит хотя бы одну виртуальную функцию). Были реализованы методы перемещения фигуры в заданные координаты, поворота на заданный угол, масштабирования на заданный коэффициент, была реализована однозначная идентификация объекта.

**ПРИЛОЖЕНИЕ А**

**UML ДИАГРАММА КЛАССОВ**

****

**ПРИЛОЖЕНИЕ Б**

**РЕАЛИЗАЦИЯ КЛАССОВ НА ЯЗЫКЕ C++**

#include "pch.h""

#include <iostream>

#include <cmath>

#include <vector>

using namespace std;

struct Point {

double x;

double y;

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

};

struct Color {

unsigned char r, g, b;

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

};

class Shape {

protected:

Point center;

int angle;

Color color;

public:

Shape(const int angle = 0, const Point center = Point(0, 0), const Color color = Color(0, 0, 0)) :angle(angle), center(center), color(color) { }

~Shape() {}

Color get\_color() { return color; }

void change\_color(Color color) {

this->color = color;

}

virtual void scale(const double factor) = 0;

void move(const Point new\_center) {

this->center = new\_center;

}

void rotate(const int delta) {

this->angle += delta;

if (this->angle > 360)

this->angle %= 360;

if (this->angle < 0)

this->angle += 360;

}

};

class Square : public Shape {

//protected:

private:

double a;

public:

Square(const double a, const int angle = 0, const Point center = Point(0, 0), const Color color = Color(0, 0, 0))

: Shape(angle, center, color), a(a) {}

~Square() {}

void scale(const double factor) override {

if (factor <= 0) {

return;

}

a \*= factor;

}

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

out << "Square" << endl

<< "Color = (" << int(obj.color.r) << ", " << int(obj.color.g) << ", " << int(obj.color.b) << ")" << endl

<< "Center = (" << obj.center.x << ", " << obj.center.y << ")" << endl

<< "Angle = " << obj.angle << endl

<< "Edge of the square = " << obj.a << endl

<< "\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

return out;

}

};

class Pentagon : public Shape {

protected:

double outer\_radius;

public:

Pentagon(const double outer\_radius, const int angle = 0, const Point center = Point(0, 0), const Color color = Color(0, 0, 0))

: Shape(angle, center, color), outer\_radius(outer\_radius) {}

~Pentagon() {}

void scale(const double factor) override {

if (factor <= 0) {

return;

}

outer\_radius \*= factor;

}

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

out << "Pentagon" << endl

<< "Color = (" << int(obj.color.r) << ", " << int(obj.color.g) << ", " << int(obj.color.b) << ")" << endl

<< "Center = (" << obj.center.x << ", " << obj.center.y << ")" << endl

<< "Angle = " << obj.angle << endl

<< "Radius of the pentagon = " << obj.outer\_radius << endl

<< "\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

return out;

}

};

class Fivepointed\_star : public Pentagon {

protected:

double internal\_radius;

public:

Fivepointed\_star(const double internal\_radius, const double outer\_radius, const int angle, const Point center = Point(0, 0), const Color color = Color(0, 0, 0))

: Pentagon(outer\_radius, angle, center, color), internal\_radius(internal\_radius) {}

~Fivepointed\_star() {}

void scale(const double factor) override {

if (factor <= 0)

return;

internal\_radius \*= factor;

outer\_radius \*= factor;

}

friend std::ostream & operator << (std::ostream &out, const Fivepointed\_star &obj) {

out << "A star" << endl

<< "Color = (" << int(obj.color.r) << ", " << int(obj.color.g) << ", " << int(obj.color.b) << ")" << endl

<< "Center = (" << obj.center.x << ", " << obj.center.y << ")" << endl

<< "Angle = " << obj.angle << endl

<< "Internal radius of the star = " << obj.internal\_radius << endl

<< "Outer radius of the star = " << obj.outer\_radius << endl

<< "\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

return out;

}

};

int main()

{

cout << "Square: " << endl;

Square t(10);

cout << t;

cout << "Move Square to (30,20)..." << endl;

t.move(Point(30, 20));

cout << "Scale edges by 4..." << endl;

t.scale(4);

cout << "Rotate Square by 30dgr.." << endl;

t.rotate(30);

cout << "Change Square's color to (254,0,100)" << endl;

t.change\_color(Color(254, 0, 100));

cout << t;

Pentagon pent(40, 0, Point(100, 20), t.get\_color());

cout << pent;

cout << "Fivepointed Star: " << endl;

Fivepointed\_star fp\_star(70, 44, 90, Point(50, 200), Color(255, 255, 255));

cout << fp\_star;

return 0;

}