**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. 7382 Лящевская А. П.

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:

* Условие задания;
* UML диаграмму разработанных классов;
* Текстовое обоснование проектных решений;
* Реализацию классов на языке С++.

**Индивидуальное задание.**

Вариант 13 – реализовать систему классов для фигур:

1. Треугольник;
2. Эллипс;
3. Прямоугольный треугольник.

**Обоснование проектных решений.**

Для представления цвета написана структура *RGB* с байтовыми полями *red*, *green*, *blue*.

Базовым классом для представления всех фигур стал класс *Shape*. В нем определены такие параметры как: координаты центра фигуры, угол поворота, цвет, масштаб и идентификационный номер с его счетчиком.

Также для работы с этими параметрами были реализованы следующие методы:

* Перемещения. Собственно, это простая смена координат центра, потому этот метод не виртуальный.

*void move(double x, double y);*

* Поворота. Работает с параметром угла, добавляя к нему нужный угол поворота принятого в качестве параметра. Не виртуален.

*void rotate(double plus\_angle);*

* Масштабирования. Для реализации этого метода в этом классе недостаточно параметров. Потому он чисто виртуальный.

*virtual void scaling(double scale) = 0;*

* Установки цвета и получения цвета. Оба метода работают с уже определенным параметром и, следовательно, не виртуальны.

*void set\_color(const RGB& set\_color);*

*RGB& get\_color();*

Отдельно стоит добавить про идентификацию каждого объекта. Для этого определена приватная статическая переменная счетчика идентификаторов (по умолчанию 0) и при каждом создании следующего объекта этого класса или зависимого (при помощи конструктора *Shape*) статическая переменная увеличивается на единицу.

Класс *Triangle* является *public* наследником класса *Shape* и используется для представления простого, ничем не обусловленного, треугольника. Он содержит в себе защищенные поля для хранения длин сторон треугольника. Под центром данной фигуры понимается центр описанной окружности.

В классе *Triangle* переопределен метод масштабирования. Данный метод увеличивает каждую из длин сторон треугольника на определенную единицу масштаба.

Следующим класс *Rigth\_Triangle* является *public* наследником класса *Triangle*. Его единственное отличие от класса Triangle состоит в том, что при инициализации объекта типа *Rigth\_Triangle*, длину третьей стороны треугольника указывать не нужно, она определяется сама по себе.

Наконец, класс *Ellipse*,наследуемый от *Shape*, содержит в себе дополнительно два приватных поля длин полуосей, описывающих размеры само й фигуры. И в переопределенном масштабировании эти размеры домножаются на нужную единицу масштаба.

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

**UML диаграмма разработанных классов.**

UML диаграмма разработанных классов представлена в приложении А.

**Реализация классов на языке С++.**

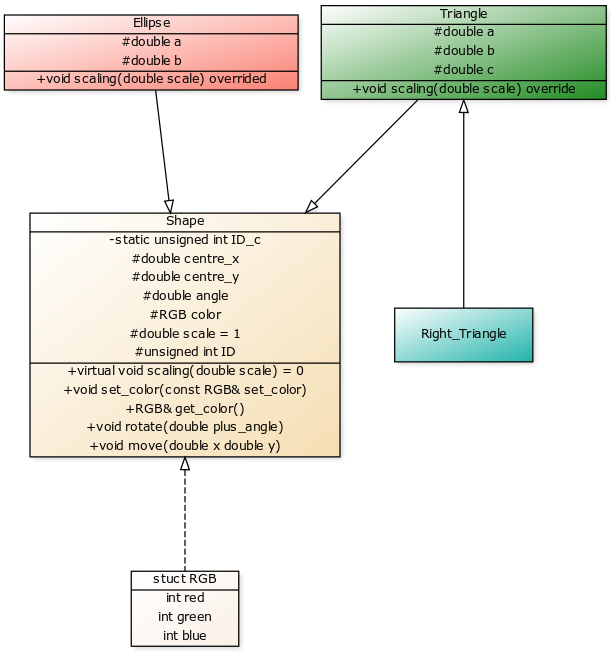
Реализация классов представлена в приложении Б.

**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 <iostream>

#include <cmath>

struct RGB

{

char red;

char green;

char blue;

};

class Shape

{

private:

static unsigned int ID\_c = 0;

protected:

double centre\_x;

double centre\_y;

double angle;

RGB color;

double scale = 1.0;

unsigned int ID;

public:

Shape()

: centre\_x(0.0), centre\_y(0.0), angle(0.0), color({0,0,0}), ID(ID\_c)

{

ID\_c++;

}

Shape(double x, double y, const RGB& set\_color)

: centre\_x(x), centre\_y(y), angle(0.0), color(set\_color), ID(ID\_c)

{

ID\_c++;

}

void move(double x, double y)

{

centre\_x = x;

centre\_y = y;

}

void rotate(double plus\_angle)

{

angle += plus\_angle;

}

virtual void scaling(double scale) = 0;

void set\_color(const RGB& set\_color)

{

color = set\_color;

}

RGB& get\_color()

{

return color;

}

};

class Triangle : public Shape

{

protected:

double a;

double b;

double c;

public:

Triangle()

: Shape(), a(0.0), b(0.0), c(0.0)

{}

Triangle(double x, double y, const RGB& color, double a, double b, double c)

: Shape(x, y, color), a(a), b(b), c(c)

{}

void scaling(double scale) override

{

a \*= scale;

b \*= scale;

c \*= scale;

}

friend std::ostream& operator<<(std::ostream& stream, const Triangle& tri)

{

stream << "Figure : Triangle" << std::endl;

stream << "ID : " << tri.ID << std::endl;

stream << "Centre coordinates: (" << tri.centre\_x << ", " << tri.centre\_y << ")" << std::endl;

stream << "Angle : " << tri.angle << std::endl;

stream << "Color (RGB) : " << tri.color.red << ":" << tri.color.green << ":" << tri.color.blue << std::endl;

stream << "Scale : " << tri.scale << std::endl;

stream << "Side: : a - " << tri.a << ", b - " << tri.b << ", c - "<< tri.c << std::endl;

return stream;

}

};

class Right\_Triangle : public Triangle

{

public:

Right\_Triangle()

: Triangle()

{}

Right\_Triangle(double x, double y, const RGB& color, double cat1, double cat2)

: Triangle(x, y, color, cat1, cat2, sqrt(pow(cat1, 2) + pow(cat2, 2)))

{}

friend std::ostream& operator<<(std::ostream& stream, const Right\_Triangle& tri)

{

stream << "Figure : Right Triangle" << std::endl;

stream << "ID : " << tri.ID << std::endl;

stream << "Centre coordinates: (" << tri.centre\_x << ", " << tri.centre\_y << ")" << std::endl;

stream << "Angle : " << tri.angle << std::endl;

stream << "Color (RGB) : " << tri.color.red << ":" << tri.color.green << ":" << tri.color.blue << std::endl;

stream << "Scale : " << tri.scale << std::endl;

stream << "Side: : cat1 - " << tri.a << ", cat2 - " << tri.b << ", hyp - "<< tri.c << std::endl;

return stream;

}

};

class Ellipse : public Shape

{

protected:

double a;

double b;

public:

Ellipse()

: Shape(), a(0.0), b(0.0)

{}

Ellipse(double x, double y, const RGB& color, double m\_a, double m\_b)

: Shape(x, y, color), a(m\_a), b(m\_b)

{}

void scaling(double scale)

{

a \*= scale;

b \*= scale;

}

friend std::ostream& operator<<(std::ostream& stream, const Ellipse& el)

{

stream << "Figure : Ellipse" << std::endl;

stream << "ID : " << el.ID << std::endl;

stream << "Centre coordinates: (" << el.centre\_x << ", " << el.centre\_y << ")" << std::endl;

stream << "Angle : " << el.angle << std::endl;

stream << "Color (RGB) : " << el.color.red << ":" << el.color.green << ":" << el.color.blue << std::endl;

stream << "Scale : " << el.scale << std::endl;

stream << "Side: : a - " << el.a << ", b - " << el.b << std::endl;

return stream;

}

};

int main(){

Triangle triangle(5, 10, {255, 0, 200}, 7, 2, 8);

std::cout << "\033[4;32mDEMO TRIANGLE\033[0m" << std::endl;

std::cout << triangle << std::endl;

std::cout << "\033[4;32mROTATE TRIANGLE +50\033[0m" << std::endl;

triangle.rotate(50);

std::cout << triangle << std::endl;

std::cout << "\033[4;32mSCALING TRIANGLE x25\033[0m" << std::endl;

triangle.scaling(25);

std::cout << triangle << std::endl;

std::cout << "\033[4;32mSET COLOR TRIANGLE 80:80:80\033[0m" << std::endl;

triangle.set\_color({80, 80, 80});

std::cout << triangle << std::endl;

std::cout << "\033[4;32mMOVE TRIANGLE (50, 60)\033[0m" << std::endl;

triangle.move(50, 60);

std::cout << triangle << std::endl;

Right\_Triangle r\_triangle(13, 13, {60, 60, 60}, 5 , 5);

std::cout << "\033[4;36mDEMO RIGHT TRIANGLE\033[0m" << std::endl;

std::cout << r\_triangle << std::endl;

Ellipse ellipse(4, 10, {40, 50, 60}, 3 , 8);

std::cout << "\033[4;31mDEMO ELLIPSE\033[0m" << std::endl;

std::cout << ellipse << std::endl;

return 0;

}