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 _____ Алясова А.Н.

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 C++.

Индивидуальное задание.

Вариант 3 – реализовать систему классов для фигур: треугольник, параллелограмм, равносторонний 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 `equilateral_triangle` является `public` наследником класса `Triangle`. его единственное отличие от класса `Triangle` состоит в том, что при инициализации объекта типа `equilateral_triangle`, длину третьей стороны треугольника указывать не нужно, она определяется сама по себе.
наконец, класс `Parallelogramm`, наследуемый от `Shape`, содержит в себе дополнительно две стороны и угол между ними, описывающих размеры самой фигуры. И в переопределенном масштабировании эти размеры умножаются 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 C++.

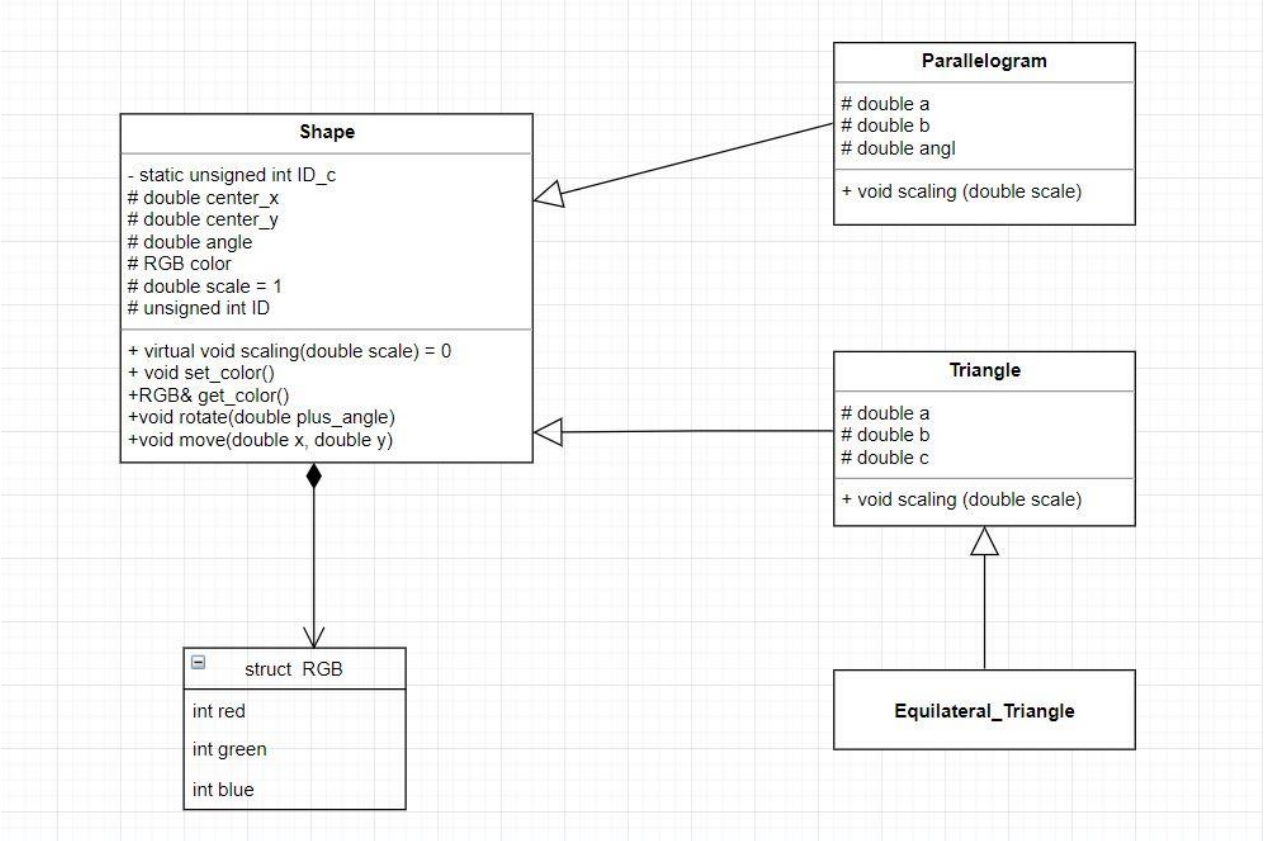
Реализация классов представлена в приложении Б.

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 <ostream>
#include <cmath>
#include <vector>
#include <string>

struct rgb {
	int red;
	int green;
	int blue;
};

class shape {
private:
	static unsigned int id_c;
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++;
      }

      //перемещение центра
      void move(double x, double y) {
            centre_x = x;
            centre_y = y;
      }

      //поворот угла
      void rotate(double plus_angle) {
            angle += plus_angle;
      }

      //масштаб
      virtual void scaling(double scale) = 0;

      //установ цвета
      void set_color(const rgb& set_color) {
            color = set_color;
      }

      //получение цвета
      rgb& get_color() {
            return color;
      }

      virtual std::ostream& print(std::ostream& stream, shape& shape) =
0;

      friend std::ostream& operator << (std::ostream& stream, shape&
shape)
      {
            return shape.print(stream, shape);
      }
};

unsigned int shape::id_c = 0;

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;
	}

	std::ostream& print(std::ostream& stream, shape& tri) override{
		stream << "figure : triangle" << std::endl;
		stream << "id : " << id << std::endl;
		stream << "centre coordinates: (" << centre_x << ", " <<
centre_y << ")" << std::endl;
		stream << "angle : " << angle << std::endl;
		stream << "color (rgb) : " << color.red << ":" <<
color.green << ":" << color.blue << std::endl;
		stream << "scale : " << scale << std::endl;
		stream << "side: : a - " << a << ", b - " << b << ", b - "
<< c << std::endl;
		return stream;
	}

class equilateral_triangle : public triangle {
public:
	equilateral_triangle() :
		triangle()
	{}

	equilateral_triangle(double x, double y, const rgb& color, double
a) :
		triangle(x, y, color, a, a, a)
	{}

	/*friend std::ostream& operator << (std::ostream& stream, const
equilateral_triangle& tri) {
		stream << "figure : equilateral triangle" << std::endl;
		stream << (shape&) tri;
```

```
            stream << "side: : a - " << tri.a << ", b - " << tri.a << ",
b - " << tri.a << std::endl;
            return stream;
      }*/

      std::ostream& print(std::ostream& stream, shape& tri) override {
            stream << "figure : equilateral triangle" << std::endl;
            stream << "id : " << id << std::endl;
            stream << "centre coordinates: (" << centre_x << ", " <<
centre_y << ")" << std::endl;
            stream << "angle : " << angle << std::endl;
            stream << "color (rgb) : " << color.red << ":" <<
color.green << ":" << color.blue << std::endl;
            stream << "scale : " << scale << std::endl;
            stream << "side: : a - " << a << ", b - " << a << ", b - "
<< a << std::endl;
            return stream;
      }
};

class parallelogram : public shape {
protected:
      double a;
      double b;
      double angle;
public:
      parallelogram()
            : shape(), a(0.0), b(0.0), angle(0.0)
      {}

      parallelogram(double x, double y, const rgb& color, double a,
double b, double angle)
            : shape(x, y, color), a(a), b(b), angle(angle)
      {}

      void scaling(double scale) {
            a *= scale;
            b *= scale;
      }

      std::ostream& print(std::ostream& stream, shape& par) override {
            stream << "figure : equilateral triangle" << std::endl;
            stream << "id : " << id << std::endl;
            stream << "centre coordinates: (" << centre_x << ", " <<
centre_y << ")" << std::endl;
```

```
            stream << "angle : " << angle << std::endl;
            stream << "color (rgb) : " << color.red << ":" <<
color.green << ":" << color.blue << std::endl;
            stream << "scale : " << scale << std::endl;
            stream << "side: : a - " << a << ", b - " << b << std::endl;
            return stream;
      }
};

int main() {
      triangle triangle(1, 2, { 67, 50, 20 }, 3, 4, 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;

      equilateral_triangle eq_triangle(13, 13, { 60, 60, 60 }, 5);
      std::cout << "\033[4;36mdemo equilateral triangle\033[0m" <<
std::endl;
      std::cout << eq_triangle << std::endl;

      parallelogram par(4, 10, { 40, 50, 60 }, 3, 8, 120);
      std::cout << "\033[4;31mdemo parallelogram\033[0m" << std::endl;
      std::cout << par << std::endl;

      std::cout << "\033[4;31mtask\033[0m";
      std::cout << std::endl;
      std::vector <shape*> objs;

      objs.push_back(new parallelogram(8, 17, { 40, 50, 60 }, 3, 8,
120));
      objs.push_back(new triangle(13, 13, { 60, 70, 60 }, 5, 10, 7));
```

```
	objs.push_back(new equilateral_triangle(17, 13, { 60, 60, 60 },
5));

	for (int i = 0; i < objs.size(); i++) {
		std::cout << *objs[i] << std::endl;
	}
	return 0;
}
```