МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по практической работе №2

по дисциплине «Объектно-ориентированное программирование»

Тема: Наследование

Студент гр. 7381 _____ Павлов 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.

Постановка задачи.

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++.

Вариант №14. Фигуры: круг, трапеция и ромб.

Текстовое обоснование разработанных классов.

`struct Point2D` – структура для хранения координат точки.

`struct ColorRGBA` – структура для хранения значения цвета фигуры.

`class Shape` – базовый класс. Он содержит основные методы для геометрических фигур:

- `void moveFigure(Point newCenter)` – перемещение фигуры в указанные координаты.
- `void rotateFigure(int newAngle)` – поворот фигуры на заданный угол.
- `void changeColor(ColorRGBA)` – раскраска.
- `ColorRGBA getColot() const` – возвращается цвет фигуры.
- `void scalingPoints()` – изменяется координаты точек фигуры.
- `void scalingFigure(double factor)` – масштабирует фигуры по заданному коэффициенту.

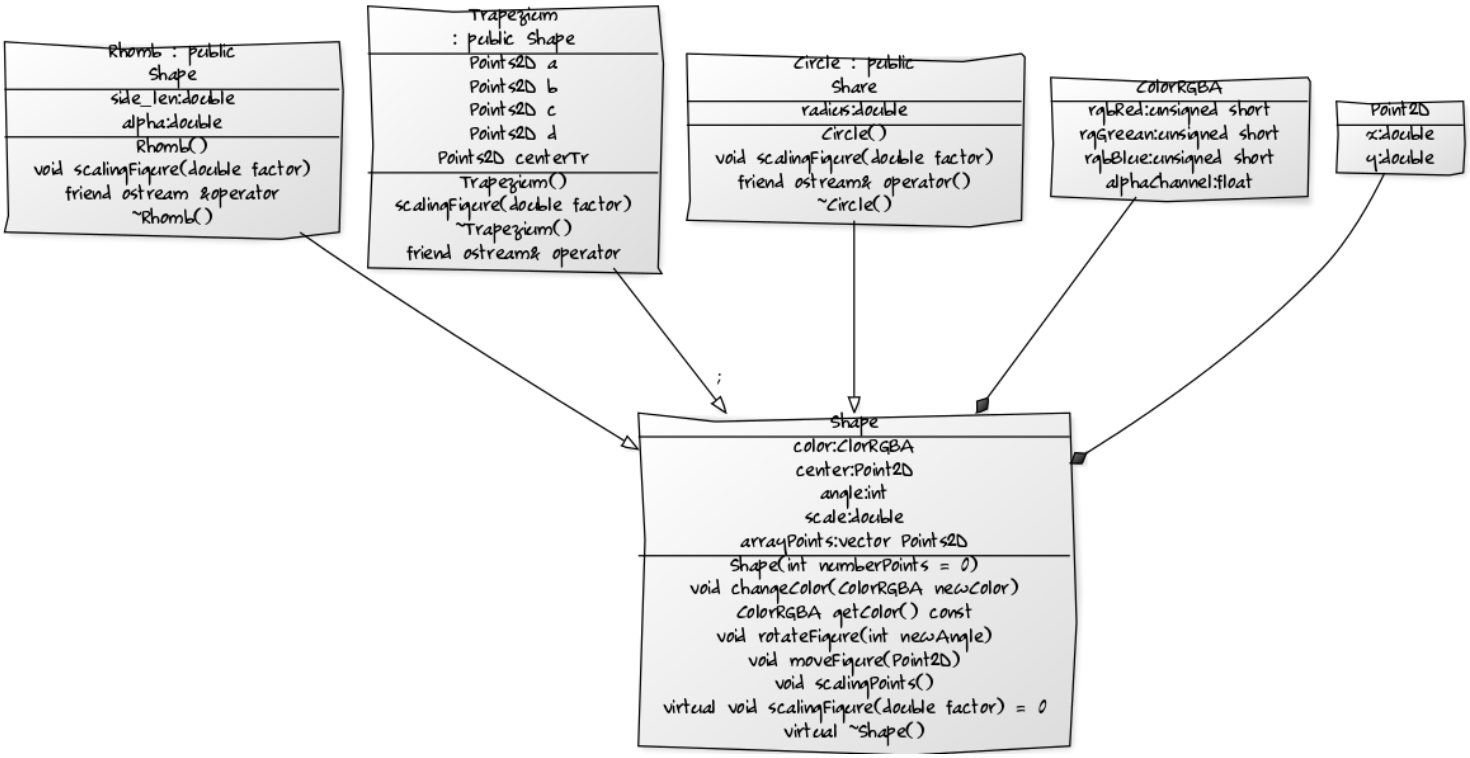
Классы `Circle`, `Trapezium` и `Rhomb` наследуются от класса `Shape`.

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 разработанных классов



ПРИЛОЖЕНИЕ Б

Исходный код программы

```
#include <iostream>
#include <vector>
#include <cstddef>
#include <cmath>

using namespace std;
#define M_PI 3.14159265358979323846

struct Point2D
{
	double x;
	double y;
};

struct ColorRGBA
{
	unsigned short rgbRed;
```

```
	unsigned short rgbGreen;
	unsigned short rgbBlue;
	float alphaChannel;
	ColorRGBA(unsigned short red, unsigned short green, unsigned
short blue, float alpha)
		: rgbRed(red), rgbGreen(green), rgbBlue(blue),
alphaChannel(alpha)
	{
	}
};

class Shape{
protected:
	ColorRGBA color;
	Point2D center;
	int angle;
	double scale;
	vector<Point2D> arrayPoints;
	static long int id;

public:
	Shape(int numberPoints = 0) : color(0, 0, 0, 0.0), center({0,
0}), angle(0), scale(1)
	{
		arrayPoints.resize(numberPoints);
		++id;
	}

	void changeColor(ColorRGBA newColor)
	{
		color.rgbRed = newColor.rgbRed;
		color.rgbGreen = newColor.rgbGreen;
		color.rgbBlue = newColor.rgbBlue;
		color.alphaChannel = newColor.alphaChannel;
	}

	ColorRGBA getColor() const
	{
		return color;
	}

	void rotateFigure(int newAngle)
	{
		angle += newAngle;
```

```
            angle %= 360;

            double rad = angle * M_PI / 180;
            double tmpX, tmpY;
            for (size_t i = 0; i < arrayPoints.size(); ++i)
            {
                  tmpX = (arrayPoints[i].x - center.x) * cos(rad) -
(arrayPoints[i].y - center.y) * sin(rad) + center.x;
                  tmpY = (arrayPoints[i].x - center.x) * cos(rad) +
(arrayPoints[i].y - center.y) * sin(rad) + center.y;
                  arrayPoints[i].x = tmpX;
                  arrayPoints[i].y = tmpY;
            }
      }

      void moveFigure(Point2D newCenter)
      {
            double dx = newCenter.x - center.x;
            double dy = newCenter.y - center.y;
            for (size_t i = 0; i < arrayPoints.size(); i++)
            {
                  arrayPoints[i].x = arrayPoints[i].x + dx;
                  arrayPoints[i].y = arrayPoints[i].y + dy;
            }
            center.x = newCenter.x;
            center.y = newCenter.y;
      }

      void scalingPoints()
      {
            for (size_t i = 0; i < arrayPoints.size(); ++i)
            {
                  arrayPoints[i].x *= scale;
                  arrayPoints[i].y *= scale;
            }
      }

      virtual void scalingFigure(double factor) = 0;
      virtual ~Shape() {}
};

class Circle : public Shape
{
private:
```

```
	double radius;

public:
	Circle(Point2D centerPoint, double r) : Shape(1), radius(r)
	{
		arrayPoints[0] = centerPoint;
		center = centerPoint;
	}
	void scalingFigure(double factor)
	{
		scale *= factor;
		radius *= abs(scale);
	}

	friend ostream &operator<<(std::ostream &out, const Circle &obj)
	{
		out << "Circle:" << endl;
		out << "ID: " << obj.id << endl;
		out << "Center: (" << obj.center.x << "," << obj.center.y <<
")" << endl;
		out << "Color RGBA: (" << obj.color.rgbRed << "," <<
obj.color.rgbGreen << "," << obj.color.rgbBlue << "," <<
obj.color.alphaChannel << ")" << endl;
		out << "Angle of rotation: " << obj.angle << endl;
		out << "Scale of figure: " << obj.scale << endl;
		return out;
	}
	~Circle() {}
};

class Trapezium : public Shape
{
private:
	Point2D a;
	Point2D b;
	Point2D c;
	Point2D d;
	Point2D centerTr;

public:
	Trapezium(Point2D _a, Point2D _b, Point2D _c, Point2D _d, Point2D
_o)
		: Shape(5), a(_a), b(_b), c(_c), d(_d), centerTr(_o)
	{
```

```
            arrayPoints[0] = a;
            arrayPoints[1] = b;
            arrayPoints[2] = c;
            arrayPoints[3] = d;
            arrayPoints[4] = centerTr;
            center = centerTr;
      }
      void scalingFigure(double factor)
      {
            scale *= factor;
            Point2D oldCenter = {centerTr.x, centerTr.y};
            scalingPoints();
            moveFigure(oldCenter);
      }

      ~Trapezium() {}

      friend ostream &operator<<(std::ostream &out, const Trapezium
&obj)
      {
            out << "Trapezium:" << endl;
            out << "ID: " << obj.id << endl;
            out << "Center: (" << obj.center.x << "," << obj.center.y <<
")" << endl;
            out << "Color RGBA: (" << obj.color.rgbRed << "," <<
obj.color.rgbGreen << "," << obj.color.rgbBlue << "," <<
obj.color.alphaChannel << ")" << endl;
            out << "Angle of rotation: " << obj.angle << endl;
            out << "Scale of figure: " << obj.scale << endl;
            return out;
      }
};

class Rhomb : public Shape {
      double side_len;
      double alpha;
public:
      Rhomb(double side_len,double alpha)
            : Shape(4), side_len(side_len), alpha(alpha) {
            arrayPoints[0] = {center.x - side_len / 2 * cos(alpha * M_PI
/ 180) - side_len / 2, center.y - side_len / 2 * sin(alpha * M_PI /
180)};
```

```
            arrayPoints[1] = {center.x - side_len / 2 * cos(alpha * M_PI
/ 180) + side_len / 2, center.y - side_len / 2 * sin(alpha * M_PI /
180)};
            arrayPoints[2] = {center.x + side_len / 2 * cos(alpha * M_PI
/ 180) + side_len / 2, center.y + side_len / 2 * sin(alpha * M_PI /
180)};
            arrayPoints[3] = {center.x + side_len / 2 * cos(alpha * M_PI
/ 180) - side_len / 2, center.y + side_len / 2 * sin(alpha * M_PI /
180)};
      };
      ~Rhomb() {};


      void scalingFigure(double factor) {
            scale *= factor;
            side_len *= scale;
            scalingPoints();
      }


      friend ostream &operator<<(std::ostream &out, const Rhomb &obj)
      {
            out << "Rhomb:" << endl;
            out << "ID: " << obj.id << endl;
            out << "Center: (" << obj.center.x << "," << obj.center.y <<
")" << endl;
            out << "Color RGBA: (" << obj.color.rgbRed << "," <<
obj.color.rgbGreen << "," << obj.color.rgbBlue << "," <<
obj.color.alphaChannel << ")" << endl;
            out << "Side linght: " << obj.side_len << endl;
            out << "alpha" << obj.alpha << endl;
            out << "Angle of rotation: " << obj.angle << endl;
            out << "Scale of figure: " << obj.scale << endl;
            return out;
      }
};

long int Shape::id = 0;
int main()
{
      ColorRGBA Red(255, 0, 0, 1.0);
      ColorRGBA Green(0, 255, 0, 0.5);

      // ------------- Trapezium tests -------------
```

```
    Trapezium tr({2.0, 2.0}, {4.0, 7.0}, {8.0, 7.0}, {14.0, 2.0},
{6.5, 5.8});
    cout << tr;
    tr.changeColor(Red);
    tr.moveFigure({1.5, 2.8});
    tr.rotateFigure(90);
    cout << tr;

    // ------------- Circle tests -------------
    Circle cir({99.0, 88.0}, 2.0);
    cout << cir;
    cir.changeColor(tr.getColor());
    cir.moveFigure({1, 12});
    cir.scalingFigure(4.0);
    cout << cir;

    // ------------- Rhomb tests -------------
    Rhomb rh(5.0, 20.0);
    cout << rh;
    rh.changeColor(Green);
    rh.moveFigure({0, -10});
    rh.rotateFigure(180);
    rh.scalingFigure(5.0);
    cout << rh;

    return 0;
}
```