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.

Студент гр. 7304 _____ Субботин А.С.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2019

Цель работы:

Изучить механизм наследования в языке программирования C++, научиться проектировать иерархию классов

Формулировка 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++.

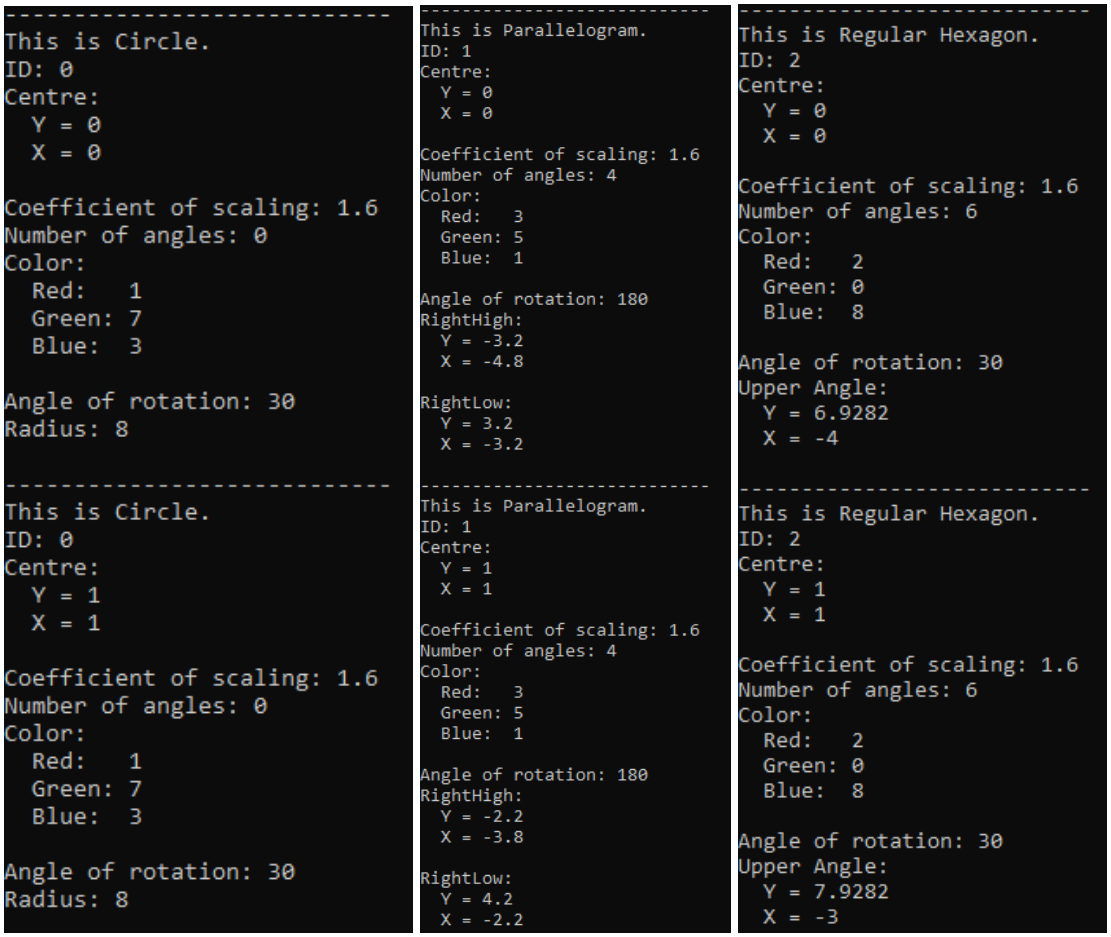
Условие задания:
Вариант 19: круг, параллелограмм, правильный шестиугольник.

Ход работы:

1) Для удобства работы с координатами был создан класс Point, содержащий двумерные координаты в формате double, а также операции сложения, вычитания, присваивания и вывода.
2) Для цвета разработан класс Color, содержащий информацию о цвете в формате rgb и операцию вывода этой информации.
3) Для реализации фигур был разработан абстрактный класс Shape, содержащий общие для всех фигур поля – centre, angles (количество углов), angle (угол поворота против часовой стрелки), color, ID, nextID, scaled (коэффициент увеличения фигуры), общие методы для получения и задания цвета, виртуальные методы для перемещения фигуры, её масштабирования, поворота и вывода информации. Также есть метод вывода общей для всех фигур информации.
4) Для реализации круга был создан дочерний класс Circle, содержащий поле radius и реализацию методов для перемещения, масштабирования, поворота и вывода информации.
5) Аналогично были созданы классы Parallelogram и RegularHexagon с полями RightHigh и RightLow для параллелограмма и radius, UpperA для шестиугольника, соответственно.

В приложении представлена UML-диаграмма по написанной программе. Линия с белой стрелкой – обозначение обобщения (показывает, какие классы наследуются). Линия с чёрным ребром – композиция (класс является частью другого).

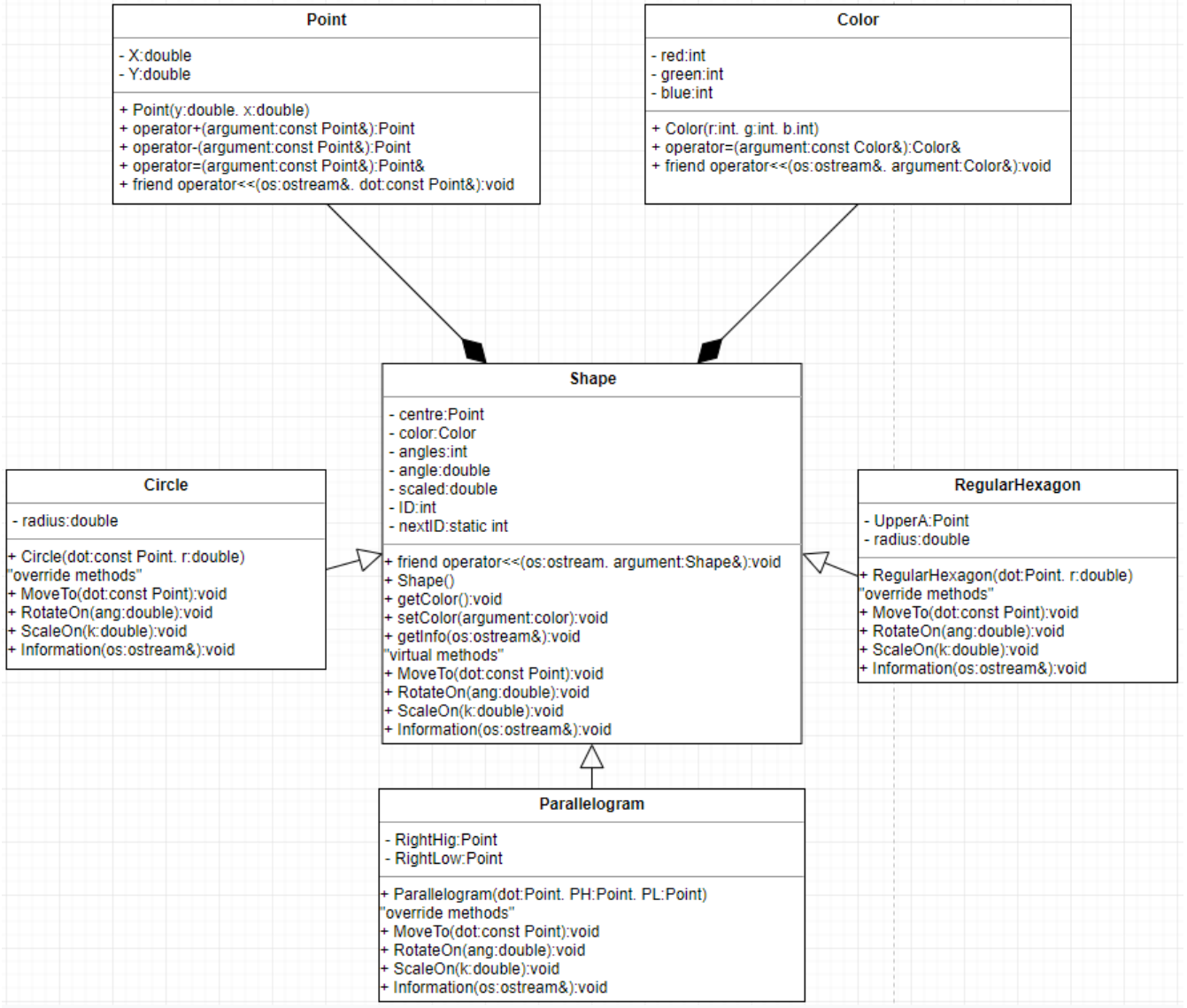
Результаты работы программы:



Выводы:

В ходе выполнения данной лабораторной работы был изучен механизм наследования в языке программирования C++, была спроектирована и реализована система классов геометрических фигур – круга, параллелограмма и правильного шестиугольника.

Приложение. UML-диаграмма



Приложение. Код программы (сборный)

```
#include <iostream>
using namespace std;

class Point
{
public:
    double X;
    double Y;

    Point() : X(0), Y(0){}
    Point(double y, double x) : X(x), Y(y){}
    ~Point(){}

    Point operator+(const Point& argument)
    {
        Point result;
        result.X = X + argument.X;
        result.Y = Y + argument.Y;
        return result;
    }

    Point operator-(const Point& argument)
    {
        Point result;
        result.X = X - argument.X;
        result.Y = Y - argument.Y;
        return result;
    }

    Point& operator=(const Point& argument)
    {
        X = argument.X;
        Y = argument.Y;
        return *this;
    }

    friend void operator<<(ostream& os, const Point& dot)
    {
        os << endl << "  Y = " << dot.Y << endl;
        os << "  X = " << dot.X << endl << endl;
    }
};

class Color
{
private:
    int red;
    int green;
    int blue;
    friend void operator<<(ostream& os, const Color& argument)
    {
        os << endl << "  Red:   " << argument.red << endl;
        os << "  Green: " << argument.green << endl;
        os << "  Blue:  " << argument.blue << endl << endl;
    }

public:
    Color(int r, int g, int b) : red(r), green(g), blue(b){}
    ~Color(){}

    Color& operator=(const Color& argument)
```

```
    {
        red = argument.red;
        green = argument.green;
        blue = argument.blue;
        return *this;
    }
};

class Shape
{
protected:
    Point centre;
    Color color;
    int angles;
    double angle;
    double scaled;
    int ID;
    static int nextID;
    friend void operator<<(ostream& os, Shape& argument)
    {
        argument.Information(os);
    }

public:
    Shape() : color(0, 0, 0)
    {
        ID = nextID++;
    }
    ~Shape(){}

    virtual void MoveTo(const Point) = 0;
    virtual void RotateOn(double) = 0;
    virtual void ScaleOn(double) = 0;
    virtual void Information(ostream&) = 0;

    void getColor()
    {
        cout << color;
    }

    void setColor(Color argument)
    {
        color = argument;
    }

    void getInfo(ostream& os)
    {
        os << "ID: " << ID << endl;
        os << "Centre: " << centre;
        os << "Coefficient of scaling: " << scaled << endl;
        os << "Number of angles: " << angles << endl;
        os << "Color: ";
        getColor();
        os << "Angle of rotation: " << angle << endl;
    }
};
int Shape::nextID = 0;
class Circle : public Shape
{
public:
    double radius;

    Circle(const Point dot, double r)
```

```
    {
        radius = r;
        centre = dot;
        angles = 0;
        angle = 0;
        scaled = 1;
    }
    ~Circle(){}

    void MoveTo(const Point dot)
    {
        centre = dot;
    }
    void RotateOn(double ang){
        angle = fmod(ang, 360.0);
    }
    void ScaleOn(double k)
    {
        scaled *= k;
        radius *= k;
    }
    void Information(ostream& os)
    {
        os << "----------------------------" << endl;
        os << "This is Circle." << endl;
        getInfo(os);
        os << "Radius: " << radius << endl << endl;
    }
};

class Parallelogram : public Shape
{
public:
    Point RightHigh;
    Point RightLow;

    Parallelogram(Point dot, Point RH, Point RL)
    {
        centre = dot;
        RightHigh = RH;
        RightLow = RL;
        angles = 4;
        angle = 0;
        scaled = 1;
    }
    ~Parallelogram(){}

    void MoveTo(const Point dot)
    {
        RightHigh = RightHigh - centre;
        RightLow = RightLow - centre;
        centre = dot;
        RightHigh = RightHigh + centre;
        RightLow = RightLow + centre;
    }
    void RotateOn(double ang)
    {
        ang = fmod(ang, 360.0);
        angle += ang;
        if(angle >= 360.0)
            angle -= 360.0;
        ang = ang * M_PI / 180;
        Point def = RightHigh - centre;
```

```
        RightHigh.X = centre.X + def.X * cos(ang) - def.Y * sin(ang);
        RightHigh.Y = centre.Y + def.Y * cos(ang) + def.X * sin(ang);

        def = RightLow - centre;
        RightLow.X = centre.X + def.X * cos(ang) - def.Y * sin(ang);
        RightLow.Y = centre.Y + def.Y * cos(ang) + def.X * sin(ang);
    }
    void ScaleOn(double k)
    {
        RightHigh = RightHigh - centre;
        RightHigh.X *= k;
        RightHigh.Y *= k;
        RightHigh = RightHigh + centre;
        RightLow = RightLow - centre;
        RightLow.X *= k;
        RightLow.Y *= k;
        RightLow = RightLow + centre;
        scaled *= k;
    }
    void Information(ostream& os)
    {
        os << "----------------------------" << endl;
        os << "This is Parallelogram." << endl;
        getInfo(os);
        os << "RightHigh: " << RightHigh;
        os << "RightLow: " << RightLow;
    }
};

class RegularHexagon : public Shape
{
public:
    Point UpperA;
    double radius;

    RegularHexagon(Point dot, double r)
    {
        centre = dot;
        radius = r;
        UpperA.X = centre.X;
        UpperA.Y = centre.Y + radius;
        angles = 6;
        angle = 0;
        scaled = 1;
    }
    ~RegularHexagon(){}

    void MoveTo(const Point dot)
    {
        UpperA = UpperA - centre;
        centre = dot;
        UpperA = UpperA + centre;
    }
    void RotateOn(double ang)
    {
        angle += ang;
        angle = fmod(angle, 360.0);
        double rad = angle * M_PI / 180;
        UpperA.X = centre.X - radius * sin(rad);
        UpperA.Y = centre.Y + radius * cos(rad);
    }
    void ScaleOn(double k)
    {
```

```
        UpperA = UpperA - centre;
        UpperA.X *= k;
        UpperA.Y *= k;
        UpperA = UpperA + centre;
        scaled *= k;
    }
    void Information(ostream& os)
    {
        os << "----------------------------" << endl;
        os << "This is Regular Hexagon." << endl;
        getInfo(os);
        os << "Upper Angle: " << UpperA;
    }
};




int main()
{
    Point centre(0.0,0.0);
    Point RH(2.0,3.0);
    Point RL(-2.0,2.0);
    double radius = 5.0;

    Circle circle(centre,radius);
    circle.setColor({1,7,3});
    circle.RotateOn(30.0);
    circle.ScaleOn(1.6);
    circle.Information(cout);
    circle.MoveTo({1,1});
    circle.Information(cout);

    Parallelogram parallelogram(centre, RH, RL);
    parallelogram.setColor({3,5,1});
    parallelogram.RotateOn(180);
    parallelogram.ScaleOn(1.6);
    parallelogram.Information(cout);
    parallelogram.MoveTo({1,1});
    parallelogram.Information(cout);

    RegularHexagon regularhexagon(centre, radius);
    regularhexagon.setColor({2,0,8});
    regularhexagon.RotateOn(30.0);
    regularhexagon.ScaleOn(1.6);
    regularhexagon.Information(cout);
    regularhexagon.MoveTo({1,1});
    regularhexagon.Information(cout);
    return 0;
}
```