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 _____ Есиков О.И.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2019

Цель работы.

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

Задача.

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 5: прямоугольник, эллипс, сектор эллипса.

Обоснование проектных решений.

1)Для удобства работы с координатами была реализована структура Point, которая содержит координаты *X* и *Y,* а также перегруженные операции сложения, вычитания и присваивания.

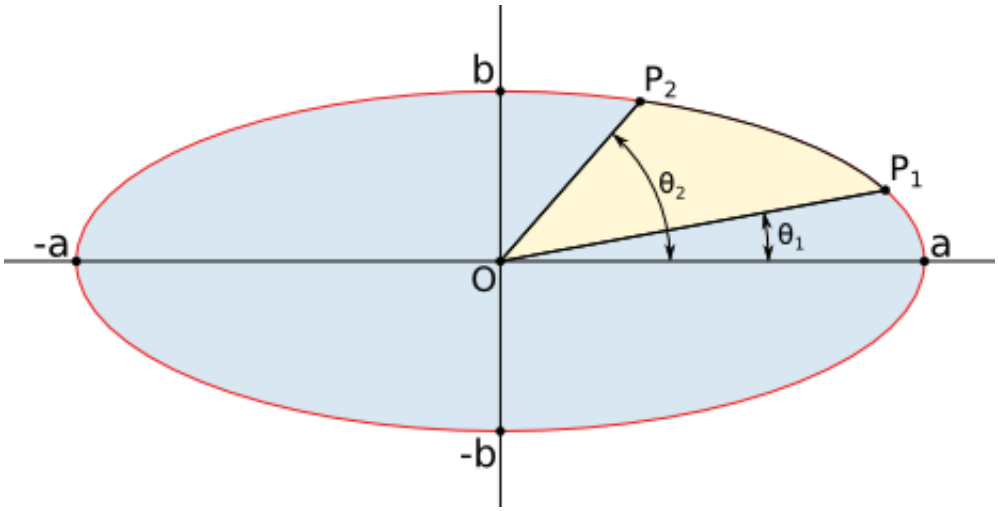
2)Для реализации цвета был разработан класс Color, который содержит информацию о цвете в формате rgb и содержит перегруженную операцию вывода этой информации.

3)Для реализации фигур был разработан абстрактный класс Shape, который содержит общие для всех фигур поля centre(центр фигуры), angle(количество углов), color(цвет фигуры), общие методы получения и задания цвета, а также виртуальные методы для масштабирования, перемещения, поворота на заданный угол и вывода информации, которые в каждом дочернем классе реализованы по-своему.

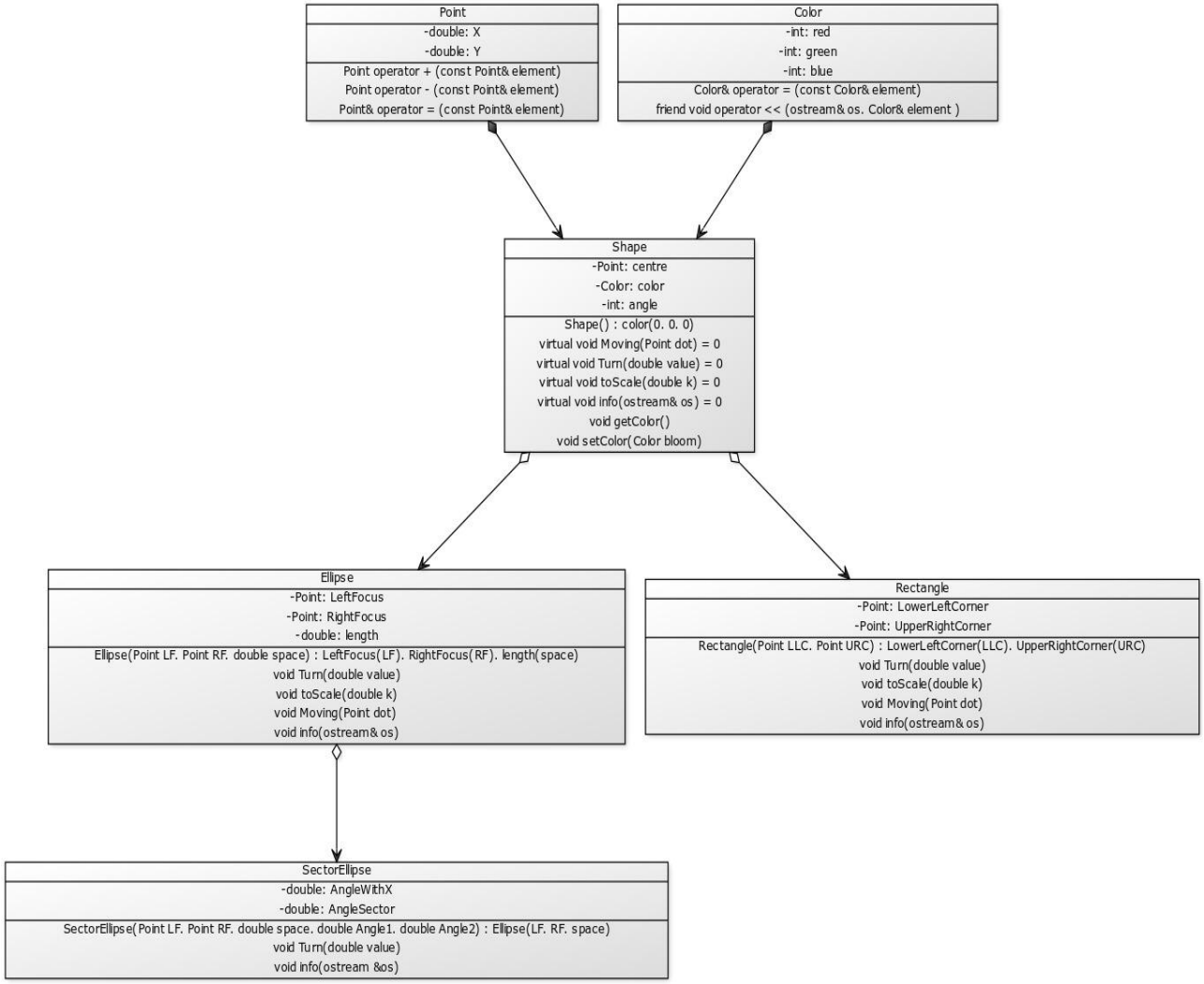
4)Для реализации прямоугольника был реализован дочерний класс от Shape, который содержит поля LowerLeftCorner и UpperRightCorner, которые содержат координаты левого нижнего угла и верхнего правого углов прямоугольника – минимальная информация, которая необходима для однозначного определения прямоугольника. В классе также реализованы методы для перемещения, масштабирования, поворота и вывода информации о прямоугольнике.

5)Для реализации эллипса был реализован дочерний класс от Shape, который содержит поля LeftFocus, RightFocus и length, которые содержат координаты левого фокуса, правого фокуса и сумму расстояний от двух фокусов до точки на эллипсе. В классе также реализованы методы для перемещения, масштабирования, поворота и вывода информации об эллипсе.

6)Для реализации сектора эллипса был реализован дочерний класс от Ellipse, который дополнительно содержит 2 поля – AngleWithX (угол θ_1 на рисунке), AngleSector (угол θ_2 на рисунке), которые содержат значения углов, отсчитываемого от оси X до начала сектора и до конца сектора соответственно. Также был реализован метод для вывода информации.



UML-диаграмма.



Исходный код.

```
#include <iostream>
#include <cmath>

using namespace std;

struct Point
{
    double X;
    double Y;
    Point operator + (const Point& element)
    {
        Point result;
        result.X = X + element.X;
        result.Y = Y + element.Y;
        return result;
    }
    Point operator - (const Point& element)
    {
        Point result;
        result.X = X - element.X;
        result.Y = Y - element.Y;
        return result;
    }
    Point& operator = (const Point& element)
    {
        X = element.X;
        Y = element.Y;
        return *this;
    }
};

class Color
{
private:
    int red, green, blue;

public:
    Color(int r, int g, int b) : red(r), green(g), blue(b){}
    ~Color(){}
    Color& operator = (const Color& element)
    {
        red = element.red;
        green = element.green;
        blue = element.blue;
        return *this;
    }
    friend void operator << (ostream& os, Color& element)
    {
        os << "Red: " << element.red << " Green: " << element.green << " Blue: "
<< element.blue << endl;
    }
};

class Shape
{
protected:
    Point centre;
    Color color;
    int angle;
```

```
    friend ostream& operator << (ostream& os, Shape& element)
    {
        element.info(os);
        return os;
    }

public:
    Shape() : color(0, 0, 0){}
    ~Shape(){}

    virtual void Moving(Point dot) = 0;      //перемещение в указанные координаты
    virtual void Turn(double value) = 0;     //поворот на указанный угол против
часовой стрелки
    virtual void toScale(double k) = 0;      //масштабирование на заданный
коэффициент
    virtual void info(ostream& os) = 0;      //вывод информации

    void getColor()
    {
        cout << color;
    }

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

class Rectangle : public Shape
{
protected:
    Point LowerLeftCorner;
    Point UpperRightCorner;

public:
    Rectangle(Point LLC, Point URC) : LowerLeftCorner(LLC),
UpperRightCorner(URC)
    {
        centre.X = (LowerLeftCorner.X + UpperRightCorner.X) / 2;
        centre.Y = (LowerLeftCorner.Y + UpperRightCorner.Y) / 2;
        angle = 4;
    }
    ~Rectangle(){}

    void Turn(double value)
    {
        double TempX = LowerLeftCorner.X, TempY = LowerLeftCorner.Y;
        LowerLeftCorner.X = centre.X + (TempX - centre.X)*cos(value) - (TempY -
centre.Y)*sin(value);
        LowerLeftCorner.Y = centre.Y + (TempY - centre.Y)*cos(value) + (TempX -
centre.X)*sin(value);

        TempX = UpperRightCorner.X, TempY = UpperRightCorner.Y;
        UpperRightCorner.X = centre.X + (TempX - centre.X)*cos(value) - (TempY -
centre.Y)*sin(value);
        UpperRightCorner.Y = centre.Y + (TempY - centre.Y)*cos(value) + (TempX -
centre.X)*sin(value);
    }

    void toScale(double k)
    {
        double side1 = UpperRightCorner.X - LowerLeftCorner.X;
        side1 = fabs(side1);
```

```
        double side2 = UpperRightCorner.Y - LowerLeftCorner.Y;
        side2 = fabs(side2);

        if(k > 1)
        {
            if(UpperRightCorner.X > LowerLeftCorner.X)
            {
                UpperRightCorner.X += side1/2 * (k-1);
                LowerLeftCorner.X -= side1/2 * (k-1);
                UpperRightCorner.Y += side2/2 * (k-1);
                LowerLeftCorner.Y -= side2/2 * (k-1);
            }
            else
            {
                UpperRightCorner.X -= side1/2 * (k-1);
                LowerLeftCorner.X += side1/2 * (k-1);
                UpperRightCorner.Y -= side2/2 * (k-1);
                LowerLeftCorner.Y += side2/2 * (k-1);
            }
        }
        else
        {
            double delta1 = (side1 - side1*k) / 2;
            double delta2 = (side2 - side2*k) / 2;
            if(UpperRightCorner.X > LowerLeftCorner.X)
            {
                UpperRightCorner.X -= delta1;
                LowerLeftCorner.X += delta1;
                UpperRightCorner.Y -= delta2;
                LowerLeftCorner.Y += delta2;
            }
            else
            {
                UpperRightCorner.X += delta1;
                LowerLeftCorner.X -= delta1;
                UpperRightCorner.Y += delta2;
                LowerLeftCorner.Y -= delta2;
            }
        }
    }

    void Moving(Point dot)
    {
        Point DeltaUp = UpperRightCorner - centre;
        Point DeltaLow = LowerLeftCorner - centre;

        centre = dot;

        UpperRightCorner = centre + DeltaUp;
        LowerLeftCorner = centre + DeltaLow;
    }

    void info(ostream& os)
    {
        os << "This is Rectangle!" << endl;
        os << "Count angle: " << angle << endl;
        os << "Centre: (" << centre.X << "; " << centre.Y << ")" << endl;
        os << "First point: (" << LowerLeftCorner.X << "; "  <<
LowerLeftCorner.Y << ")" << endl;
        os << "Second point: (" << UpperRightCorner.X << "; "  <<
UpperRightCorner.Y << ")" << endl;
        os << endl;
    }
```

```
};

class Ellipse : public Shape
{
protected:
    Point LeftFocus;
    Point RightFocus;
    double length; // |MF1| + |MF2| = const = 2a = length!

public:
    Ellipse(Point LF, Point RF, double space) : LeftFocus(LF), RightFocus(RF),
length(space)
    {
        centre.X = (LF.X + RF.X) / 2;
        centre.Y = (LF.Y + RF.Y) / 2;
        angle = 0;
    }
    ~Ellipse(){}

    void Turn(double value)
    {
        double TempX = LeftFocus.X, TempY = LeftFocus.Y;
        LeftFocus.X = centre.X + (TempX - centre.X)*cos(value) - (TempY -
centre.Y)*sin(value);
        LeftFocus.Y = centre.Y + (TempY - centre.Y)*cos(value) + (TempX -
centre.X)*sin(value);

        TempX = RightFocus.X, TempY = RightFocus.Y;
        RightFocus.X = centre.X + (TempX - centre.X)*cos(value) - (TempY -
centre.Y)*sin(value);
        RightFocus.Y = centre.Y + (TempY - centre.Y)*cos(value) + (TempX -
centre.X)*sin(value);
    }

    void toScale(double k)
    {
        if(k > 1)
        {
            if(RightFocus.X > LeftFocus.X)
            {
                RightFocus.X += length/2 * (k-1);
                LeftFocus.X -= length/2 * (k-1);
                length *= k;
            }
            else
            {
                RightFocus.X -= length/2 * (k-1);
                LeftFocus.X += length/2 * (k-1);
                length *= k;
            }
        }
        else
        {
            double delta = (length - length*k)/2;
            if(RightFocus.X > LeftFocus.X)
            {
                RightFocus.X -= delta;
                LeftFocus.X += delta;
                length *= k;
            }
            else
            {
                RightFocus.X += delta;
```

```
                LeftFocus.X -= delta;
                length *= k;
            }
        }
    }

    void Moving(Point dot)
    {
        Point Delta1 = RightFocus - centre;
        Point Delta2 = LeftFocus - centre;

        centre = dot;

        RightFocus = centre + Delta1;
        LeftFocus = centre + Delta2;
    }

    void info(ostream& os)
    {
        os << "This is Ellipse!" << endl;
        os << "Count angle: " << angle << endl;
        os << "Centre: (" << centre.X << "; " << centre.Y << ")" << endl;
        os << "First focus: (" << LeftFocus.X << "; "  << LeftFocus.Y << ")" <<
endl;
        os << "Second focus: (" << RightFocus.X << "; "  << RightFocus.Y << ")"
<< endl;
        os << "Length: " << length << endl;
        os << endl;
    }
};

class SectorEllipse : public Ellipse
{
protected:
    double AngleWithX;
    double AngleSector;

public:
    SectorEllipse(Point LF, Point RF, double space, double Angle1, double
Angle2) : Ellipse(LF, RF, space)
    {
        AngleWithX = Angle1;
        AngleSector = Angle2;
        angle = 1;
    }
    ~SectorEllipse(){}

    void Turn(double value)
    {
        Ellipse::Turn(value);
        AngleWithX += value;
        AngleSector += value;
    }

    void info(ostream &os)
    {
        os << "This is SectorEllipse!" << endl;
        os << "Count angle: " << angle << endl;
        os << "Centre: (" << centre.X << "; " << centre.Y << ")" << endl;
        os << "First focus: (" << LeftFocus.X << "; "  << LeftFocus.Y << ")" <<
endl;
        os << "Second focus: (" << RightFocus.X << "; "  << RightFocus.Y << ")"
<< endl;
```

```
        os << "Length: " << length << endl;
        os << "MinAngle: " << AngleWithX << endl;
        os << "MaxAngle: " << AngleSector << endl;
        os << endl;
    }
};

int main()
{
    Rectangle test1({-1.0, -2.0}, {5.0, 2.0});
    cout << test1;
    test1.Turn(M_PI/2);
    test1.toScale(2);
    test1.Moving({0.0, 0.0});
    cout << test1;

    Ellipse test2({-3.3, 0.0}, {3.3, 0.0}, 15.5);
    cout << test2;
    test2.Turn(M_PI/2);
    test2.Turn(M_PI/2);
    test2.toScale(3.0);
    test2.toScale(1.0/3.0);
    test2.Moving({4.0, 4.0});
    cout << test2;

    SectorEllipse test3({0.0, 4.2}, {0.0, -4.2}, 10.95, 0.52, 1.4);
    cout << test3;
    test3.getColor();
    test3.setColor({255, 0, 0});
    test3.getColor();

    return 0;
}
```

Результат работы.

```
C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
This is Rectangle!
Count angle: 4
Centre: (2; 0)
First point: (-1; -2)
Second point: (5; 2)

This is Rectangle!
Count angle: 4
Centre: (0; 0)
First point: (4; 0)
Second point: (-4; 0)

This is Ellipse!
Count angle: 0
Centre: (0; 0)
First focus: (-3.3; 0)
Second focus: (3.3; 0)
Length: 15.5

This is Ellipse!
Count angle: 0
Centre: (4; 4)
First focus: (7.3; 4)
Second focus: (0.7; 4)
Length: 15.5

This is SectorEllipse!
Count angle: 1
Centre: (0; 0)
First focus: (0; 4.2)
Second focus: (0; -4.2)
Length: 10.95
MinAngle: 0.52
MaxAngle: 1.4

Red: 0 Green: 0 Blue: 0
Red: 255 Green: 0 Blue: 0
```

Выводы.

В ходе выполнения данной лабораторной работы был изучен механизм наследования в языке программирования с++, была спроектирована, реализована и обоснована система классов геометрических фигур – прямоугольника, эллипса и сектора эллипса.