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 _____ Овчинников Н.В.

Преподаватель _____ Размочаева Н.В

Санкт-Петербург

2019

Цель работы

Ознакомиться с наследованием в объектно-ориентированном программировании. Научиться проектировать системы классов с использованием абстрактного базового класса и виртуальных функций.

Задание

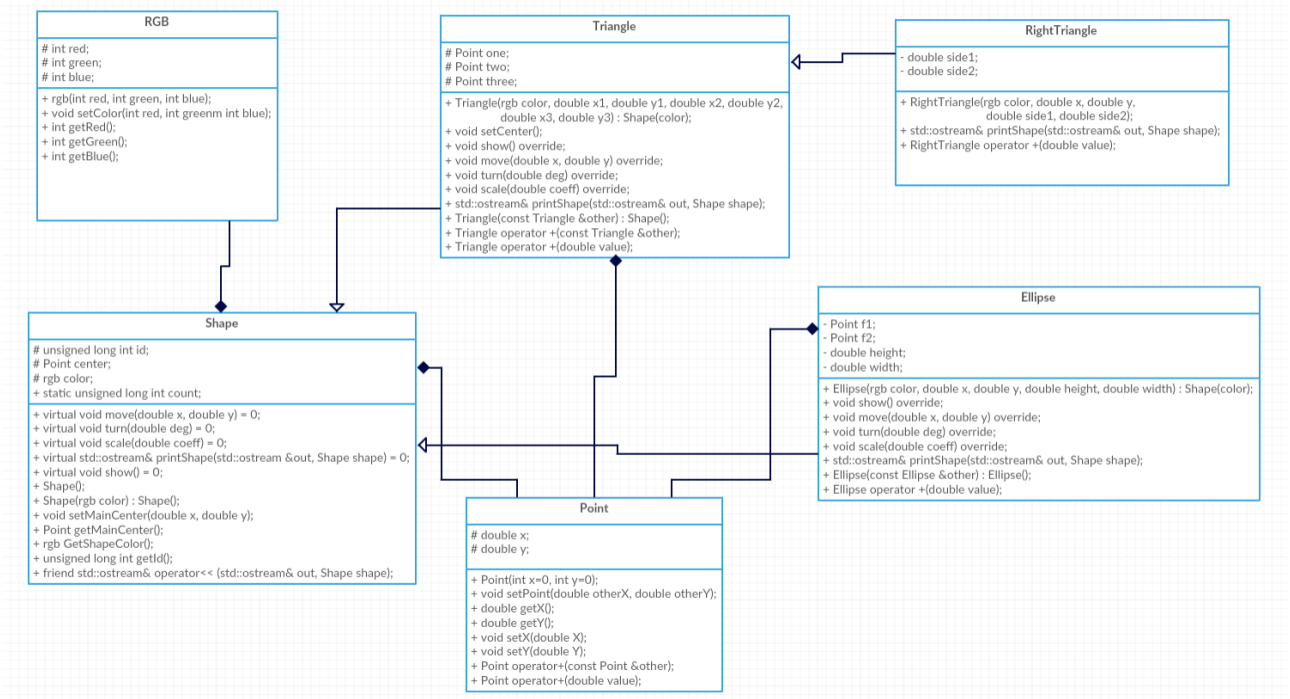
Необходимо спроектировать систему классов для моделирования геометрических фигур (в соответствии с полученным индивидуальным заданием). Задание предполагает использование виртуальных функций в иерархии наследования, проектирование и использование абстрактного базового класса. Разработанные классы должны 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++.

Фигуры: треугольник, эллипс, прямоугольный треугольник.

UML диаграмма разработанных классов



Текстовое обоснование проектных решений

Базовый абстрактный класс Shape содержит: координату центра, цвет и уникальный идентификатор. Также в нём определены виртуальные методы: перемещения в указанную координату (move), поворота фигуры на заданный угол (turn), масштабирование на заданный коэффициент (scale), вывода в поток, а также функция отображения фигуры в Qt Creator. От класса Shape наследуются классы Triangle и Ellipse. Triangle, помимо полей базового класса и реализацию виртуальных методов, содержит координаты трёх вершин, метод установки центра треугольника и перегруженные операторы сложения. Ellipse в дополнении к полям базового класса Shape содержит координаты двух фокусов, а также ширину и высоту. От класса Triangle унаследован класс RightTriangle, который помимо координат трёх вершин содержит размеры двух сторон при прямом угле. Для удобства были реализованы классы определяющие цвет (RGB) и координату точки (Point).

Вывод

В ходе выполнения лабораторной работы была спроектирована система классов для моделирования геометрических фигур с использованием абстрактного базового класса и виртуальных функций. Также была выполнена визуализация фигур в Qt Creator.

Приложение №1. Реализация классов

1. Класс rgb:

```
class rgb
{
protected:
    int red;
    int green;
    int blue;

public:
    rgb(){}
    rgb(int red, int green, int blue) : red(red), green(green), blue(blue){}

    void setColor(int red, int green, int blue)
    {
        this->red = red;
        this->green = green;
        this->blue = blue;
    }

    int getRed()
    {
        return red;
    }

    int getGreen()
    {
        return green;
    }

    int getBlue()
    {
        return blue;
    }
};
```

2. Класс Point:

```
class Point
{
protected:
    double x;
    double y;

public:
    Point(int x=0, int y=0) : x(x), y(y){}

    void setPoint(double otherX, double otherY)
    {
        x = otherX;
        y = otherY;
    }

    double getX() const
    {
        return x;
    }
```

```
    double getY() const
    {
        return y;
    }

    void setX(double X)
    {
        x = X;
    }

    void setY(double Y)
    {
        y = Y;
    }

    Point operator +(const Point &other)
    {
        Point tmp;
        tmp.x = (this->x + other.x)/2;
        tmp.y = (this->y + other.y)/2;
        return tmp;
    }

    Point operator +(double value)
    {
        Point tmp;
        tmp.x = this->x + value;
        tmp.y = this->y + value;
        return tmp;
    }
};
```

3. Класс Shape:

```
class Shape
{
protected:
    unsigned long int id;
    Point center;
    rgb color;

public:
    static unsigned long int count;
    virtual void move(double x, double y) = 0;
    virtual void turn(double deg) = 0;
    virtual void scale(double coeff) = 0;
    virtual std::ostream& printShape(std::ostream &out, Shape &shape) = 0;
    virtual void show() = 0;

    Shape()
    {
        count++;
        id = count;
    }

    Shape(rgb color) : Shape()
    {
        this->color = color;
    }

    void setMainCenter(double x, double y)
    {
```

```
        center.setPoint(x, y);
    }

    Point getMainCenter(){
        return center;
    }

    rgb getShapeColor()
    {
        return color;
    }

    unsigned long int getId(){
        return id;
    }

    friend std::ostream& operator<< (std::ostream& out, Shape &shape)
    {
        return shape.printShape(out, shape);
    }
};
```

4. Класс Triangle:

```
class Triangle : public Shape
{
protected:
    Point one;
    Point two;
    Point three;

public:
    Triangle(){}
    Triangle(rgb color, double x1, double y1, double x2, double y2, double x3, double
y3) : Shape(color)
    {
        one.setPoint(x1,y1);
        two.setPoint(x2,y2);
        three.setPoint(x3,y3);
        setCenter();
    }

    void setCenter(){
        double x12 = (one.getX() + two.getX())/2;
        double y12 = (one.getY() + two.getY())/2;
        double x13 = (one.getX()+three.getX())/2;
        double y13 = (one.getY()+three.getY())/2;
        double expression1 = y12*(three.getX()-x12)/(three.getY()-y12)/(two.getX()-x13)
- x12/(two.getX()-x13) + x13/(two.getX()-x13) - y13/(two.getY()-y13);
        double expression2 = (three.getX()-x12)/(three.getY()-y12)/(two.getX()-x13) -
1/(two.getY()-y13);
        double y = expression1/expression2;
        double x = (y-y12)*(three.getX()-x12)/(three.getY()-y12) + x12;
        setMainCenter(x,y);
    }

    void show() override
    {
        QGraphicsScene *scene = new QGraphicsScene;
        scene->setSceneRect(-500, -500, 1000, 1000);
        scene->addLine(0,-500,0,500);
```

```
        scene->addLine(-500,0,500,0);

        rgb clr = getShapeColor();
        QColor color(clr.getRed(), clr.getGreen(), clr.getBlue());
        QPen pen;
        pen.setColor(color);
        pen.setWidth(3);

        scene->addLine(one.getX(), one.getY(), two.getX(), two.getY(), pen);
        scene->addLine(two.getX(), two.getY(), three.getX(), three.getY(), pen);
        scene->addLine(three.getX(), three.getY(), one.getX(), one.getY(), pen);

        QGraphicsView *view = new QGraphicsView;
        view->setScene(scene);
        view->show();
    }

    void move(double x, double y) override
    {
        Point firstCenter = getMainCenter();
        Point diff(x - firstCenter.getX(), y - firstCenter.getY());
        one.setX(one.getX() + diff.getX());
        one.setY(one.getY() + diff.getY());
        two.setX(two.getX() + diff.getX());
        two.setY(two.getY() + diff.getY());
        three.setX(three.getX() + diff.getX());
        three.setY(three.getY() + diff.getY());
        setMainCenter(x, y);
    }

    void turn(double deg) override
    {
        Point firstCenter = getMainCenter();
        one.setY((one.getY() - one.getX()*sin(deg*PI/180)/cos(deg*PI/180)) /
(sin(deg*PI/180)*sin(deg*PI/180)/cos(deg*PI/180)+cos(deg*PI/180)));
        one.setX((one.getX() + one.getY()*sin(deg*PI/180)) / cos(deg*PI/180));
        two.setY((two.getY() - two.getX()*sin(deg*PI/180)/cos(deg*PI/180)) /
(sin(deg*PI/180)*sin(deg*PI/180)/cos(deg*PI/180)+cos(deg*PI/180)));
        two.setX((two.getX() + two.getY()*sin(deg*PI/180)) / cos(deg*PI/180));
        three.setY((three.getY() - three.getX()*sin(deg*PI/180)/cos(deg*PI/180)) /
(sin(deg*PI/180)*sin(deg*PI/180)/cos(deg*PI/180)+cos(deg*PI/180)));
        three.setX((three.getX() + three.getY()*sin(deg*PI/180)) / cos(deg*PI/180));
        setCenter();
        move(firstCenter.getX(), firstCenter.getY());
    }

    void scale(double coeff) override
    {
        Point firstCenter = getMainCenter();
        Point sizeOneTwo(two.getX() - one.getX(), two.getY() - one.getY());
        Point sizeOneThree(three.getX() - one.getX(), three.getY() - one.getY());
        sizeOneTwo.setPoint(coeff * sizeOneTwo.getX(), coeff * sizeOneTwo.getY());
        sizeOneThree.setPoint(coeff * sizeOneThree.getX(), coeff *
sizeOneThree.getY());
        two.setX(one.getX() + sizeOneTwo.getX());
        two.setY(one.getY() + sizeOneTwo.getY());
        three.setX(one.getX() + sizeOneThree.getX());
        three.setY(one.getY() + sizeOneThree.getY());

        setCenter();
        move(firstCenter.getX(), firstCenter.getY());
    }
```

```
    std::ostream& printShape(std::ostream &out, Shape &shape) override
    {
        rgb clr = shape.getShapeColor();
        out << "Triangle: (" << one.getX() << ", " << one.getY() << ") (" << two.getX()
<< ", " << two.getY() << ") (" << three.getX() << ", " << three.getY() << ")\n";
        out << "          Center of shape: (" << center.getX() << ", " << center.getY()
<< ")\n";
        out << "          ID: " << shape.getId() << " Color: (" << clr.getRed() << ", "
<< clr.getGreen() << ", " << clr.getBlue() << ")\n";
        return out;
    }

    Triangle(const Triangle &other) : Shape()
    {
        one = other.one;
        two = other.two;
        three = other.three;
        center = other.center;
        color = other.color;
        id = count;
    }

    Triangle operator +(const Triangle& other)
    {
        Triangle tmp;
        tmp.one = this->one + other.one;
        tmp.two = this->two + other.two;
        tmp.three = this->three + other.three;
        tmp.setCenter();
        tmp.color.setColor((this->color.getRed()+other.color.getRed())/2, (this-
>color.getGreen()+other.color.getGreen())/2, (this-
>color.getBlue()+other.color.getBlue())/2);
        return tmp;
    }

    Triangle operator +(double value)
    {
        Triangle tmp = *this;
        tmp.one = tmp.one + value;
        tmp.two = tmp.two + value;
        tmp.three = tmp.three + value;
        tmp.setCenter();
        return tmp;
    }
};
```

5. Класс Ellipse:

```
class Ellipse : public Shape
{
private:
    Point f1;
    Point f2;
    double height;
    double width;

public:
    Ellipse(){}
    Ellipse(rgb color, double x, double y, double height, double width) : Shape(color)
    {
```

```
        this->height = height;
        this->width = width;
        setMainCenter(x, y);
        double c = sqrt(width>height ? width*width/4 - height*height/4 :
height*height/4 - width*width/4);
        f1.setY(height/2);
        f1.setX(x - c);
        f2.setY(height/2);
        f2.setX(x + c);
    }

    void show() override
    {
        QGraphicsScene *scene = new QGraphicsScene;
        scene->setSceneRect(-500, -500, 1000, 1000);

        rgb clr = getShapeColor();
        QColor color(clr.getRed(), clr.getGreen(), clr.getBlue());
        QPen pen;
        pen.setColor(color);
        pen.setWidth(3);

        Point centralPoint = getMainCenter();
        scene->addEllipse(centralPoint.getX() - width/2, centralPoint.getY() -
height/2, width, height, pen);

        QGraphicsView *view = new QGraphicsView;
        view->setScene(scene);
        view->show();
    }

    void move(double x, double y) override
    {
        Point firstCenter = getMainCenter();
        Point diff(x - firstCenter.getX(), y - firstCenter.getY());
        f1.setX(f1.getX() + diff.getX());
        f1.setY(f1.getY() + diff.getY());
        f2.setX(f2.getX() + diff.getX());
        f2.setY(f2.getY() + diff.getY());
        setMainCenter(x,y);
        show();
    }

    void turn(double deg) override
    {
        f1.setY((f1.getY() - f1.getX()*sin(deg*PI/180)/cos(deg*PI/180)) /
(sin(deg*PI/180)*sin(deg*PI/180)/cos(deg*PI/180)+cos(deg*PI/180)));
        f1.setX((f1.getX() + f1.getY()*sin(deg*PI/180)) / cos(deg*PI/180));
        f2.setY((f2.getY() - f2.getX()*sin(deg*PI/180)/cos(deg*PI/180)) /
(sin(deg*PI/180)*sin(deg*PI/180)/cos(deg*PI/180)+cos(deg*PI/180)));
        f2.setX((f2.getX() + f2.getY()*sin(deg*PI/180)) / cos(deg*PI/180));

        QGraphicsScene *scene = new QGraphicsScene;
        scene->setSceneRect(-500, -500, 1000, 1000);

        rgb clr = getShapeColor();
        QColor color(clr.getRed(), clr.getGreen(), clr.getBlue());
        QPen pen;
        pen.setColor(color);
        pen.setWidth(3);

        Point cet = getMainCenter();
```

```
        scene->addEllipse(cet.getX(), cet.getY(), width, height, pen);

        QGraphicsView *view = new QGraphicsView;
        view->setScene(scene);
        view->rotate(360-deg);
        view->show();
    }

    void scale(double coeff) override
    {
        height *= coeff;
        width *= coeff;
        Point centr = getMainCenter();
        double c = sqrt(width>height ? width*width/4 - height*height/4 :
height*height/4 - width*width/4);
        f1.setY(height/2);
        f1.setX(centr.getX() - c);
        f2.setY(height/2);
        f2.setX(centr.getX() + c);
        show();
    }

    std::ostream& printShape(std::ostream &out, Shape &shape) override
    {
        rgb clr = shape.getShapeColor();
        out << "Ellipse: width = " << width << ", height = " << height << ", focus1("
<< f1.getX() << ", " << f1.getY() << "), focus2(" << f2.getX() << ", " << f2.getY() <<
")\n";
        out << "         Center of shape: (" << center.getX() << ", " << center.getY()
<< ")\n";
        out << "         ID: " << shape.getId() << " Color: (" << clr.getRed() << ", "
<< clr.getGreen() << ", " << clr.getBlue() << ")\n";
        return out;
    }

    Ellipse(const Ellipse &other) : Ellipse()
    {
        height = other.height;
        width = other.width;
        f1 = other.f1;
        f2 = other.f2;
        center = other.center;
        color = other.color;
        id = count;
    }

    Ellipse operator +(double value)
    {
        Ellipse tmp = *this;
        tmp.f1 = this->f1 + value;
        tmp.f2 = this->f2 + value;
        tmp.center = this->center + value;
        return tmp;
    }
};
```

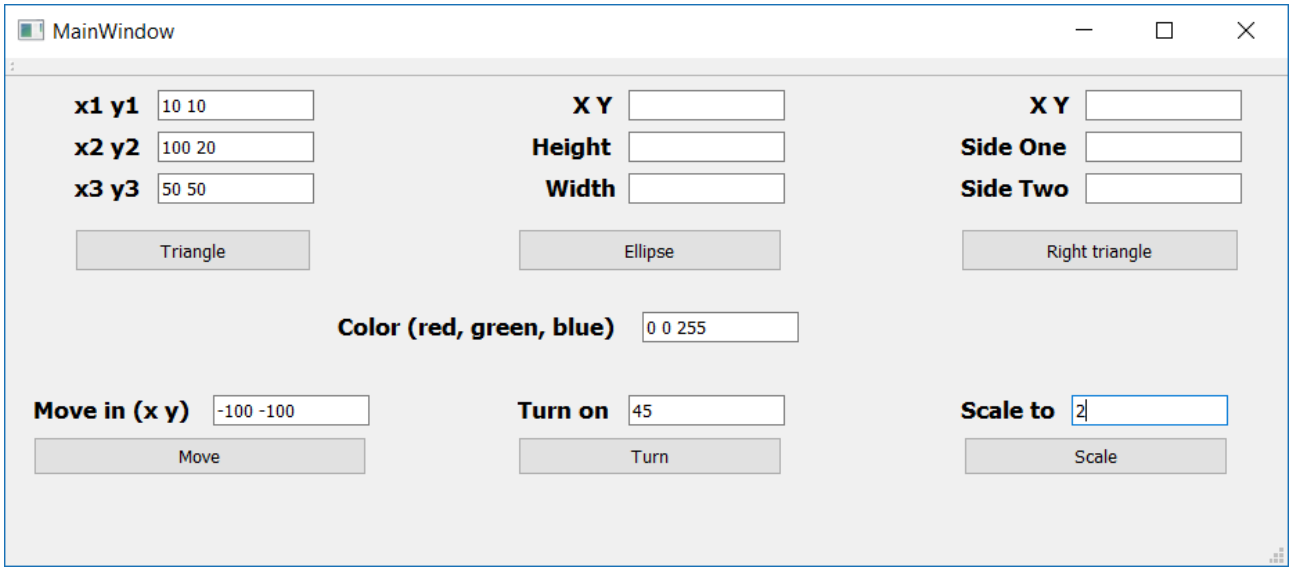
6. Класс RightTrianble:

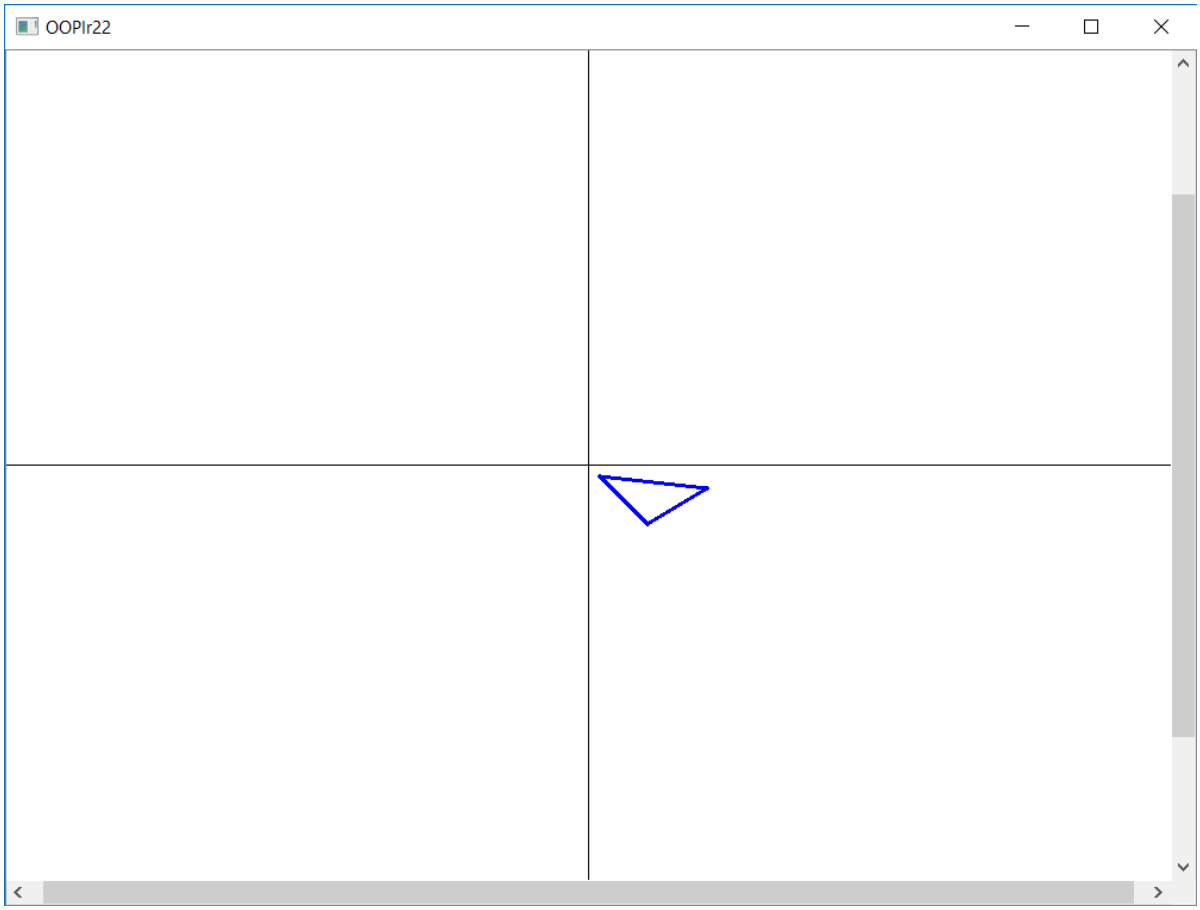
```
class RightTriangle : public Triangle
{
private:
```

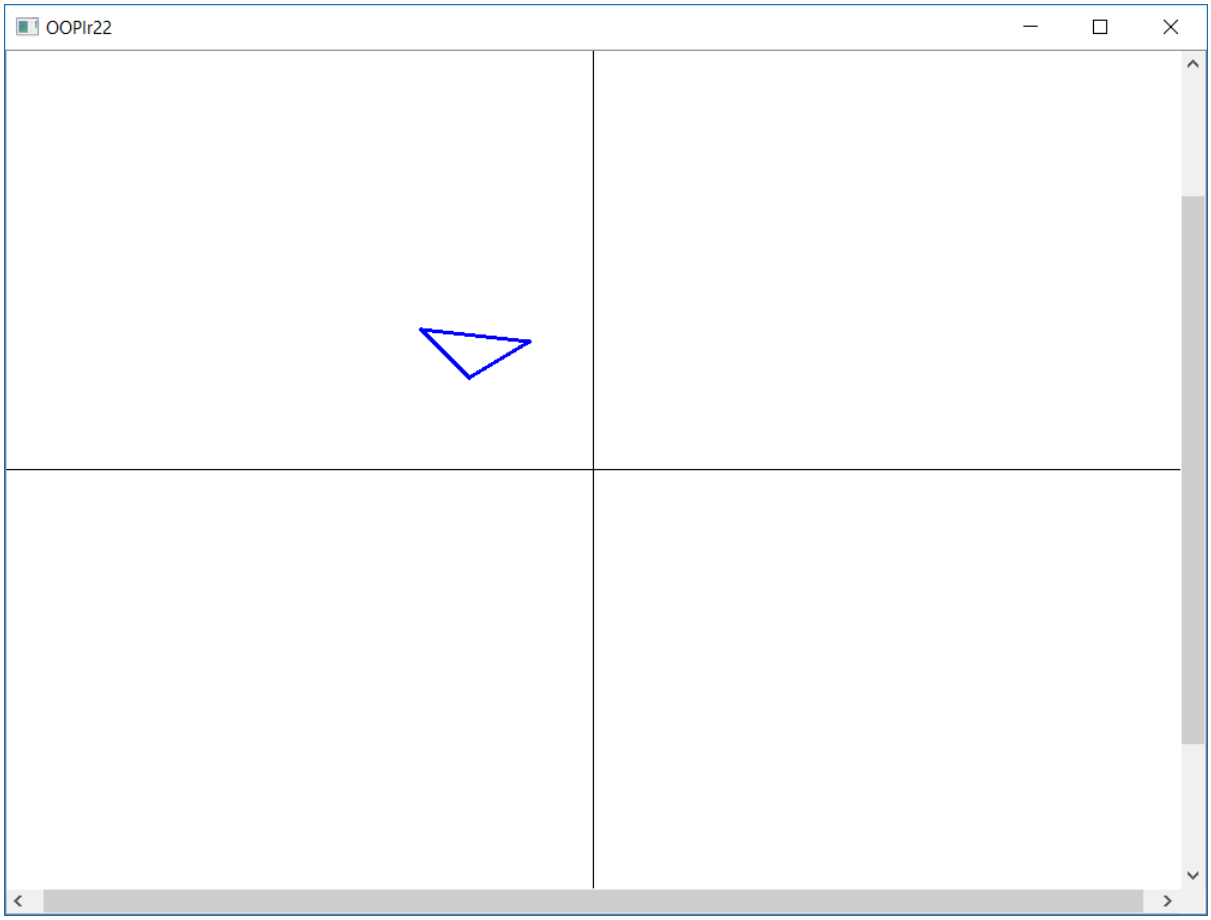
```
    double side1;
    double side2;

public:
    RightTriangle(){}
    RightTriangle(rgb color, double x, double y, double side1, double side2)
    {
        this->side1 = side1;    //вертикаль
        this->side2 = side2;    //горизонталь
        setMainCenter(x, y);
        one.setPoint(x-side2/4, y-side1/4);
        two.setPoint(one.getX(), one.getY() + 3*side1/4);     //вертикаль
        three.setPoint(one.getX() + 3*side2/4, one.getY());   //горизонталь
        this->color.setColor(color.getRed(), color.getGreen(), color.getBlue());
    }

    std::ostream& printShape(std::ostream &out, Shape &shape) override
    {
        rgb clr = shape.getShapeColor();
        out << "RightTriangle: (" << one.getX() << ", " << one.getY() << ") (" <<
two.getX() << ", " << two.getY() << ") (" << three.getX() << ", " << three.getY() <<
")\n";
        out << "               Center of shape: (" << center.getX() << ", " <<
center.getY() << ")\n";
        out << "               ID: " << shape.getId() << " Color: (" << clr.getRed() <<
", " << clr.getGreen() << ", " << clr.getBlue() << ")\n";
        return out;
    }

    RightTriangle operator +(double value)
    {
        RightTriangle tmp = *this;
        tmp.one = tmp.one + value;
        tmp.two = tmp.two + value;
        tmp.three = tmp.three + value;
        tmp.setCenter();
        return tmp;
    }

};
```

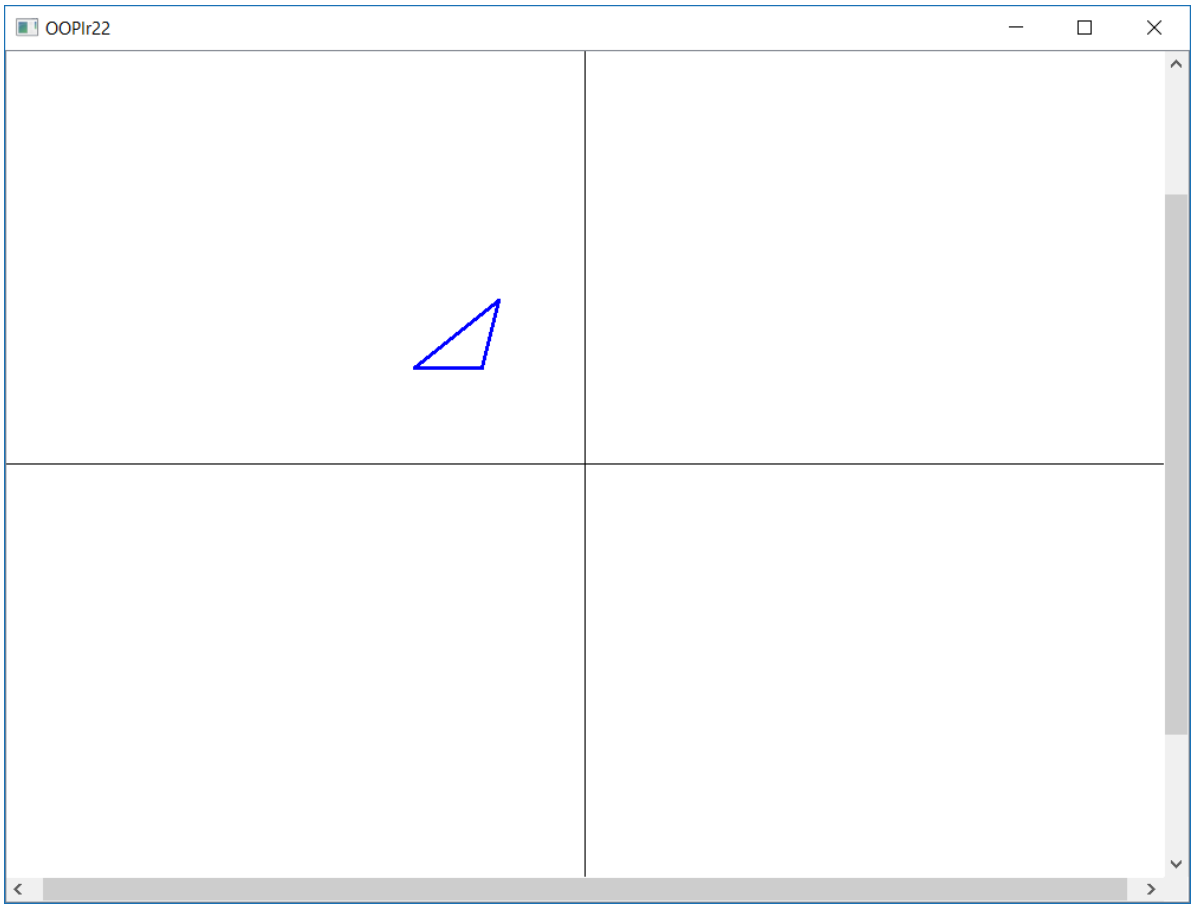
Приложение №2. Визуализация

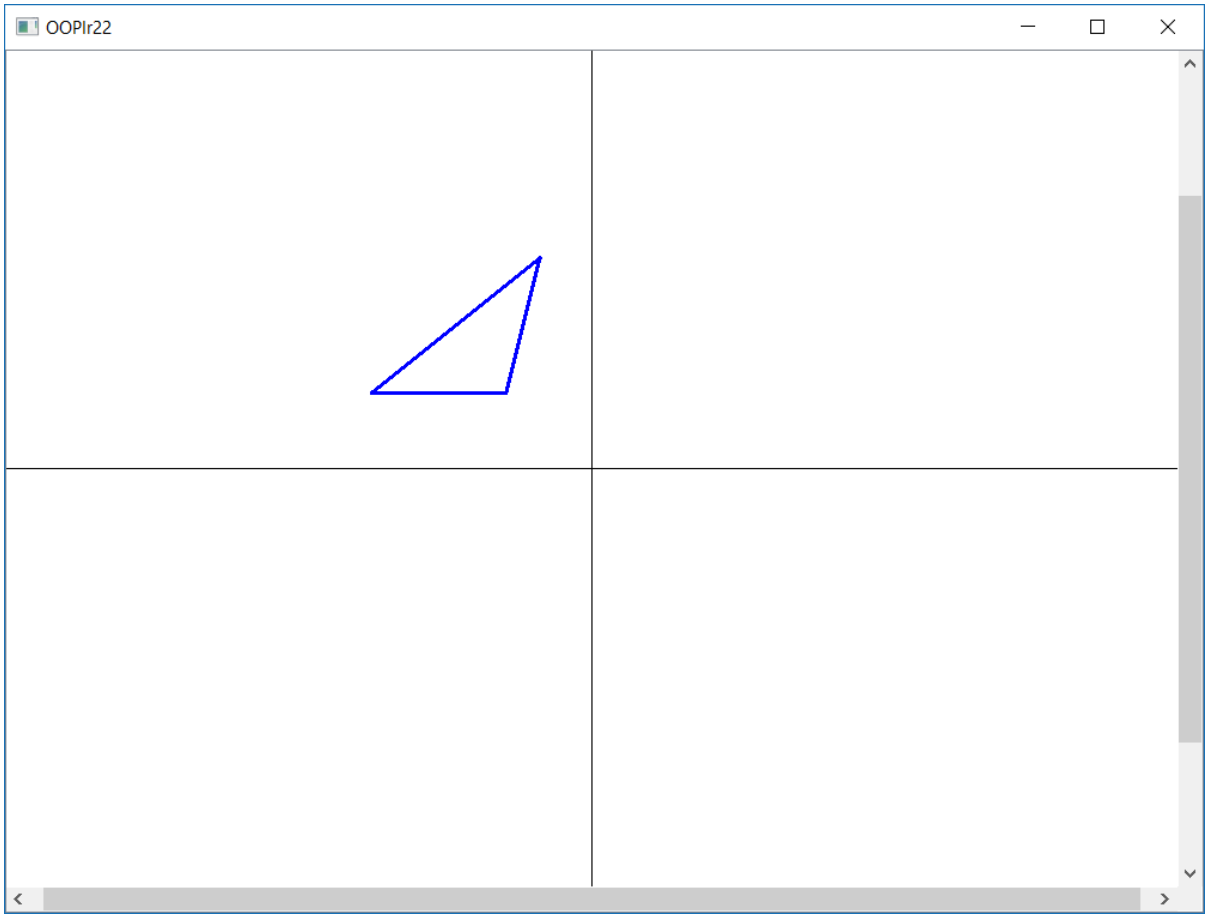
Для наглядности работы была реализована визуализация геометрический фигур в Qt Creator. Для примера будет создан синий треугольник, который будет перемещен в точку (-100, -100), повернут на 45 градусов против часовой стрелки и увеличен вдвое:


MainWindow
x1 y1
10 10
x2 y2
100 20
x3 y3
50 50
Triangle
X Y
Height
Width
Ellipse
X Y
Side One
Side Two
Right triangle
Color (red, green, blue)
0 0 255
Move in (x y)
-100 -100
Move
Turn on
45
Turn
Scale to
2
Scale


OOPIr22


OOPlr22


OOPlr22


OOPlr22