МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Объектно-ориентированное программирование»

Тема: «Наследование»

Студент гр. 7304 _____ Комаров А.О.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2019

Цель работы:

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++.

Ход работы:

1. Реализованы классы:
    1. Класс Point. Содержит два поля, описывающие координаты x и y точки.
    2. Класс Color. Содержит три поля, содержащие числа от 0 до 255 и характеризуют цвет фигуры. Класс Color содержит методы для получения информации о цвете.
    3. Абстрактный класс Shape содержит поля цвета, номера фигуры, id, координаты центра фигуры, вектор, хранящий координаты вершин фигуры. Класс Shape содержит такие

методы как:set_color(для установления заданного цвета), get_id(для получения информации о id фигуры), moving(для смещения фигуры в заданную точку), rotation для поворота фигуры на заданный угол, scaling(виртуальный метод для масштабирования фигуры на заданный коэффициент).

4. Класс Triangle. наследуется от Shape. Класс имеет поле, которое характеризует стороны треугольника и его угол между сторонами. В конструкторе данного класса вычисляются третья сторона треугольника, и координаты вершин относительно центройда треугольника. В классе был переопределен метод scaling, который масштабирует треугольник на заданный коэффициент и метод вывода информации о фигуре.
5. Класс Right_Triangle, наследуется от Triangle. В конструкторе класса угол между сторонами задается как 90 градусов. Все методы он наследует от класса треугольника.
6. Класс fivePointedStar наследуется от Shape. В конструкторе определяются пять точек звезды и угол пятиконечной звезды. Так же переопределяет scaling и вывод.

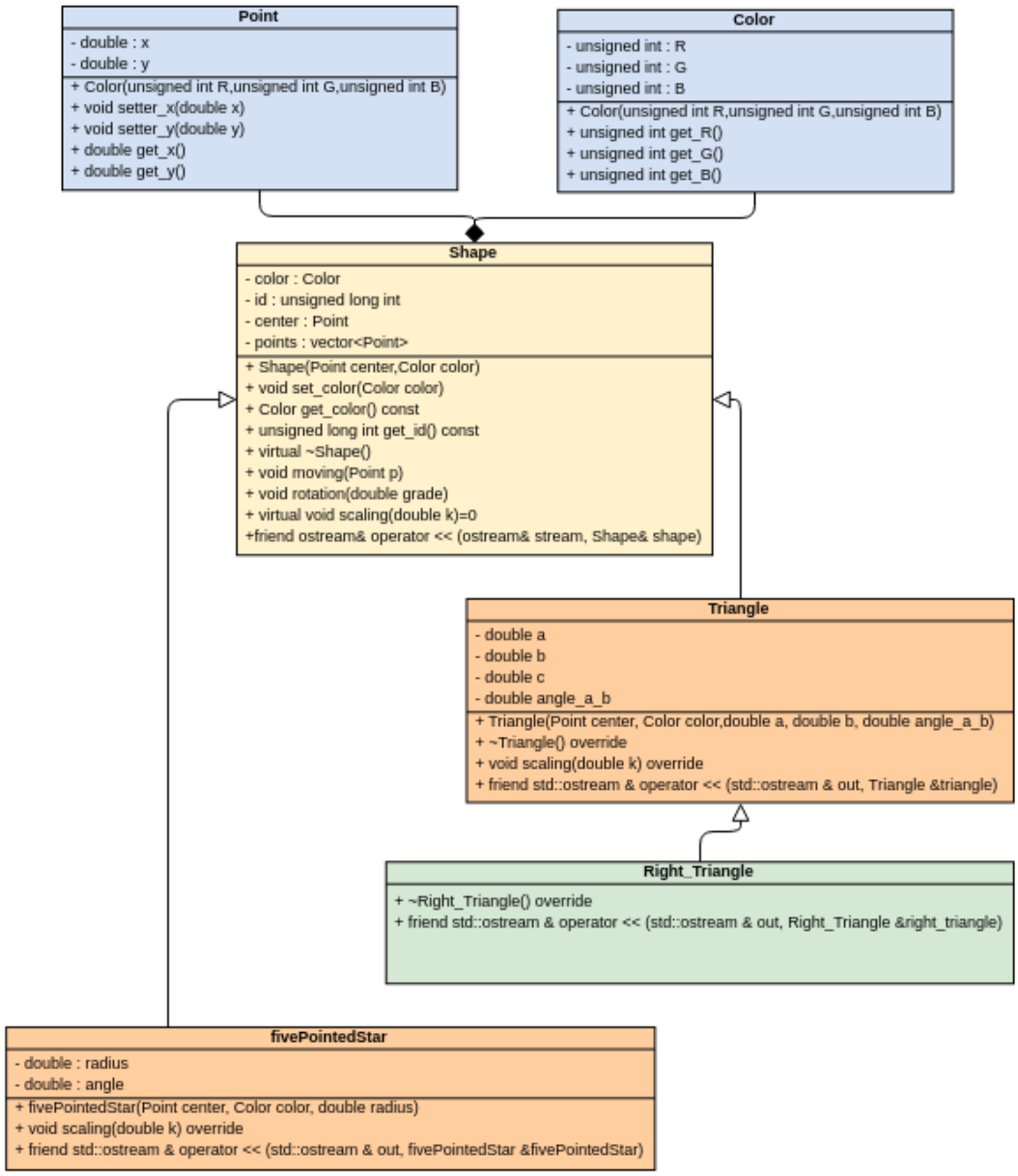
Обоснование решения:

В лабораторной работе был реализован абстрактный класс Shape.

Поля цвет,координаты центра фигуры и координаты вершин фигуры являются общими, поэтому они содержатся в абстрактном классе Shape.

Для реализации треугольника нужно знать две его стороны и угол между сторонами.
Для реализации прямоугольного треугольника достаточно знать двух его сторон.
Для реализации пятиконечной звезды достаточно знать значения её радиуса(относительно центра до края) и угол между двумя краями.
Moving — это перемещение в заданную точку. Для любой фигуры можно найти расстояние между новой точкой и текущими центром фигуры. Перемещение фигуры — это смещение каждой из вершин этой фигуры на полученное расстояние.
Rotation — это поворот на заданный угол. Для каждой фигуры поворот на заданный угол можно получить, умножив все координаты фиугры на матрицу поворота.

UML диаграмма:



Вывод:

В ходе данной лабораторной работы была спроектирована система классов, представляющих геометрические фигуры: Треугольник, Прямоугольный треугольник, Пятиконечная звезда, наследуемых от общего базового класса Shape, реализованы необходимые методы для этих классов, а также обеспечена однозначная идентификация для каждого объекта.

Приложение

Исходный код программы

```
#include <iostream>
#include <vector>
#include <math.h>//PI
//Треугольник  Пятиконечная звезда    Прямоугольный треугольник
using namespace std;
class Point{
    double x,y;
public:
    Point(double x=0, double y=0):x(x),y(y){};
    void setter_x(double x){
        this->x = x;
    }
    void setter_y(double y){
        this->y=y;
    }
    double get_x(){
        return x;
    }
    double get_y(){
         return y;
     };
};
class Color{
    unsigned int R,G,B;
public:
    Color(unsigned int R,unsigned int G,unsigned int B): R(R%256),G(G%256),B(B
%256){};
 unsigned int get_R(){
      return R;
 }
    unsigned int get_G(){
        return G;
    }
    unsigned int get_B(){
        return B;
    }
};
class Shape {
protected:
    Color color;
    unsigned long int id;
    Point center;
    vector <Point> points;
public:
    Shape(Point center,Color color ):color(color), center(center) {
        static long int i = 0;
        id = i;
        i++;
    }
    void set_color(Color color) {
        this->color = color;
    }
    Color get_color() const {
        return color;
    }
    unsigned long int get_id() const {
        return id;
```

```
    }
    virtual ~Shape()
    {
        cout << "~Shape()" << endl;
    }
    void moving(Point p) {
        double offset_by_x = p.get_x() - center.get_x();
        double offset_by_y = p.get_y() - center.get_y();
        for (size_t i = 0; i < points.size(); i++) {
            double tmp_x = points[i].get_x() + offset_by_x;
            double tmp_y = points[i].get_y() + offset_by_y;
            points[i].setter_x(tmp_x);
            points[i].setter_y(tmp_y);
        }
        center = p;
    }
    void rotation(double grade) {
        double grade_in_rad = grade*M_PI/180.0;
        for (size_t i = 0; i < points.size(); i++) {
            double x =center.get_x()+(points[i].get_x()-
center.get_x())*cos(grade_in_rad)-(points[i].get_y()-
center.get_y())*sin(grade_in_rad);
            double y = center.get_y() + (points[i].get_x() -
center.get_x())*sin(grade_in_rad) + (points[i].get_y() -
center.get_y())*cos(grade_in_rad); ;
            points[i].setter_x(x);
            points[i].setter_y(y);
        }
    }
    virtual void scaling(double k)=0;
    friend ostream& operator << (ostream& stream, Shape& shape) {
        stream << "Object id: " << shape.get_id();
        stream << endl << "(x, y): " << shape.center.get_x() << ", " <<
shape.center.get_y() << endl;
        stream << "Цвет фигуры: " << shape.color.get_R() << " " <<
shape.color.get_G() << " " << shape.color.get_B()<< endl;
        stream << "Координаты фигуры: " << endl;
        for (size_t i = 0; i < shape.points.size(); i++) {
            stream << "(" << shape.points[i].get_x() << ";" <<
shape.points[i].get_y() << ")" << endl;
        }
        return stream;
    }
};
class Triangle : public Shape
{
public:
    Triangle(Point center, Color color,double a, double b, double angle_a_b) :
Shape (center,color), a(a), b(b)
    {
        if(angle_a_b >= 360)
            this->angle_a_b = angle_a_b - int(angle_a_b / 360) * 360;
        else
            this->angle_a_b = angle_a_b;
        c = sqrt(a*a + b*b - 2*a*b*cos(angle_a_b * M_PI / 180));
        double p=a+b+c/2;
        double S=sqrt(p*(p-a)*(p-b)*(p-c));
        double Bb1=sqrt(a*a+(b/2)*(b/2)-a*b*cos(angle_a_b));
        double qb=S*2/(3*b);
        double BG=Bb1-qb;
        double angle_BB1_B1C=180-asin(b*sin(angle_a_b)/(2*a))-angle_a_b;
```

```
        double Bx=sin(angle_BB1_B1C)*BG+center.get_x();
        double By=cos(angle_BB1_B1C)*BG+center.get_y();
        points.push_back(Point(Bx,By));
        double yC=a*sin(angle_a_b)+Bx;
        double xC=a*cos(angle_a_b)+By;
        points.push_back(Point(xC,yC));
        double yA=b*sin(angle_a_b)+yC;
        double xA=b*cos(angle_a_b)+xC;
        points.push_back(Point(xA,yA));
    }
    ~Triangle() override
    {
        cout << "~Triangle()" << endl;
    }
    void scaling(double k) override
    {
        a *= k;
        b *= k;
        c *= k;
        center.setter_y(center.get_y()*k);
        center.setter_x(center.get_x()*k);
        for (auto& pt : points) {
            pt.setter_x(pt.get_x()*k);
            pt.setter_y(pt.get_y()*k);
        }
    }
    friend std::ostream & operator << (std::ostream & out, Triangle &triangle)
    {
        out << dynamic_cast<Shape &>(triangle) << endl << "Side a: " <<
triangle.a << endl <<  "Side b: " << triangle.b << endl <<  "Side c:" <<
triangle.c<< endl <<  "angle:" << triangle.angle_a_b;
        return out;
    }
protected:
    double a;
    double b;
    double angle_a_b;
    //this side compute by theorem of cos
    double c;
};
class Right_Triangle : public Triangle{
public:
    Right_Triangle(Point center, Color color,double a, double b)
:Triangle(center,color,a,b,90) {
/*        if(angle_a_b >= 360)
            this->angle_a_c = angle_a_c - int(angle_a_c / 360) * 360;
        else
            this->angle_a_c = angle_a_c;*/
        /*c = sqrt(a * a + b * b);
       // angle_a_c = asin(b / c);
        double xA = c / 2, yA = 0, xB = -c / 2, yB = 0;
        //0. Длина катета AB (ab):
        //   ab = Sqrt((xa_ - xb_)^2+(ya_ - yb_)^2)
        //1. Вектор AB = B - A, покоординатно. Делим обе координаты на длину,
получаем единичный вектор (v1):
        //   v1.x =  (B.x - A.x) / ab  ===  v1x = (xb_ - xa_) / ab
        //   v1.y =  (B.y - A.y) / ab  ===  v1y = (yb_ - ya_) / ab
        //2. Поворачиваем вектор v1 на 90 градусов, получаем вектор вдоль
другого катета (v2). Поворот по формуле:
        //   v2.x = -v1.y              ===  v2x = -v1y
        //   v2.y =  v1.x              ===  v2y =  v1x
```

```
        //   Альтернативно поворот в другую сторону:
        //   v2.x =  v1.y;
        //   v2.y = -v1.x;
        //3. Имея единичный вектор v2 вдоль второго катета, умножаем
покоординатно на длину второго катета, получаем вектор AC:
        //   v3.x = v2.x * bc_          ===  v3x = v2x * bc_
        //   v3.y = v2.y * bc_          ===  v3y = v2y * bc_
        //4. Прибавляем к координатам A вектор AC, получаем точку C:
        //   xc_ = xa_ + v3x
        //   yc_ = ya_ + v3y
        double x2x1 = xA - xB;
        double y2y1 = yA - yB;
        double v1x = (xB - xA) / c;
        double v1y = (yB - yA) / c;
        double v3x = (v1y > 0 ? -v1y : v1y) * a;
        double v3y = (v1x > 0 ? v1x : -v1x) * a;
        double xC = xA + v3x;
        double yC = yA + v3y;
        points.push_back(Point(xA,yA));
        points.push_back(Point(xB,yB));
        points.push_back(Point(xC,yC));*/
    }
    ~Right_Triangle() override
    {
        cout << "~Right_Triangle()" << endl;
    }
  /*  void scaling(double k) override
    {
        a *= k;
        b *= k;
        c *= k;
    }*/
    friend std::ostream & operator << (std::ostream & out, Right_Triangle
&right_triangle)
    {
        out << dynamic_cast<Shape &>(right_triangle) << endl << "Side a: " <<
right_triangle.a << endl <<  "Side b: " << right_triangle.b << endl <<  "Side
c:" << right_triangle.c<< endl <<  "angle:" << right_triangle.angle_a_b;
        return out;
    }
private:
    //double a;
    //double b;
    //double angle_a_c;
   // double c;
};
class fivePointedStar : public Shape {
private:
    double radius;
    double angle;
public:
    fivePointedStar(Point center, Color color, double radius): Shape
(center,color), radius(radius){
         angle = 2 * M_PI / 5;              //делится круг на 5 частей
        points.push_back(Point(center.get_x() + radius * cos(angle *
0),center.get_y() + radius * sin(angle * 0)));
        points.push_back(Point(center.get_x()  + radius * cos(angle *
1),center.get_y()  + radius * sin(angle * 1)));
        points.push_back(Point(center.get_x()  + radius * cos(angle *
2),center.get_y()  + radius * sin(angle * 2)));
```

```
        points.push_back(Point(center.get_x()  + radius * cos(angle *
3),center.get_y()  + radius * sin(angle * 3)));
        points.push_back(Point(center.get_x()  + radius * cos(angle *
4),center.get_y()  + radius * sin(angle * 4)));
            }
   void scaling(double k) override
   {
        radius*=k;
       center.setter_y(center.get_y()*k);
       center.setter_x(center.get_x()*k);
       for (auto& pt : points) {
           pt.setter_x(pt.get_x()*k);
           pt.setter_y(pt.get_y()*k);
       }
    }
    friend std::ostream & operator << (std::ostream & out, fivePointedStar
&fivePointedStar)
    {
        out << dynamic_cast<Shape &>(fivePointedStar) << endl << "Side radius: "
<<  fivePointedStar.radius << endl <<  "Side angle: " << fivePointedStar.angle
<< endl;
        return out;
    }
};
int main() {
    /*Shape *shape = new fivePointedStar({3,3},{1,1,1},4);
    shape->rotation(26);
    shape->moving(5);*/
    Shape *shape= new Right_Triangle({3,3},{12,123,123},3,4);
    shape->moving({10,10});
    cout << *dynamic_cast<Right_Triangle*>(shape) << endl;
    //cout << *dynamic_cast<fivePointedStar*>(shape) << endl;
/*    Shape *shape = new Triangle({5,5},{235,23,23},1,2,9);
    Shape *shape2 = new Triangle({5,5},{235,23,23},1,2,8);
    Shape *shape3 = new Triangle({5,5},{235,23,23},1,2,8);
    cout << *dynamic_cast<Triangle*>(shape3) << endl;*/
  //  Shape *shape = new Right_Triangle({5,5},{235,23,23},1,2);
    //shape->scaling(5);
   // cout << *dynamic_cast<Right_Triangle*>(shape) << endl;
 //   Shape *shape = new Triangle({6,8},{ 255, 255, 255 },3,4,90);
//shape->scaling(5);
//shape->rotation(180);
  //  shape->moving({7,8});
    //cout << *dynamic_cast<Triangle *>(shape) << endl;
/*Shape *shape1= new Right_Triangle({0,0},{ 255, 255, 255 },3,4);
    cout << *dynamic_cast<Right_Triangle*>(shape1) << endl;*/
/*    shape->scaling(4);
    cout << *dynamic_cast<Triangle *>(shape) << endl;
    shape->rotation(40);
    cout << *dynamic_cast<Triangle *>(shape) << endl;
    //cout << shape.
    4.80484;7.61437)
(7.27015;6.48683)
(5.47785;10.0628)*/
    return 0;
}
```