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.7382 Ленковский В.В.

Преподаватель Жангиров Т.Р.

г. Санкт-Петербург

2019 г.

Цель работы:

Необходимо спроектировать систему классов для моделирования геометрических фигур квадрата, параллелограмма, ромба. 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:

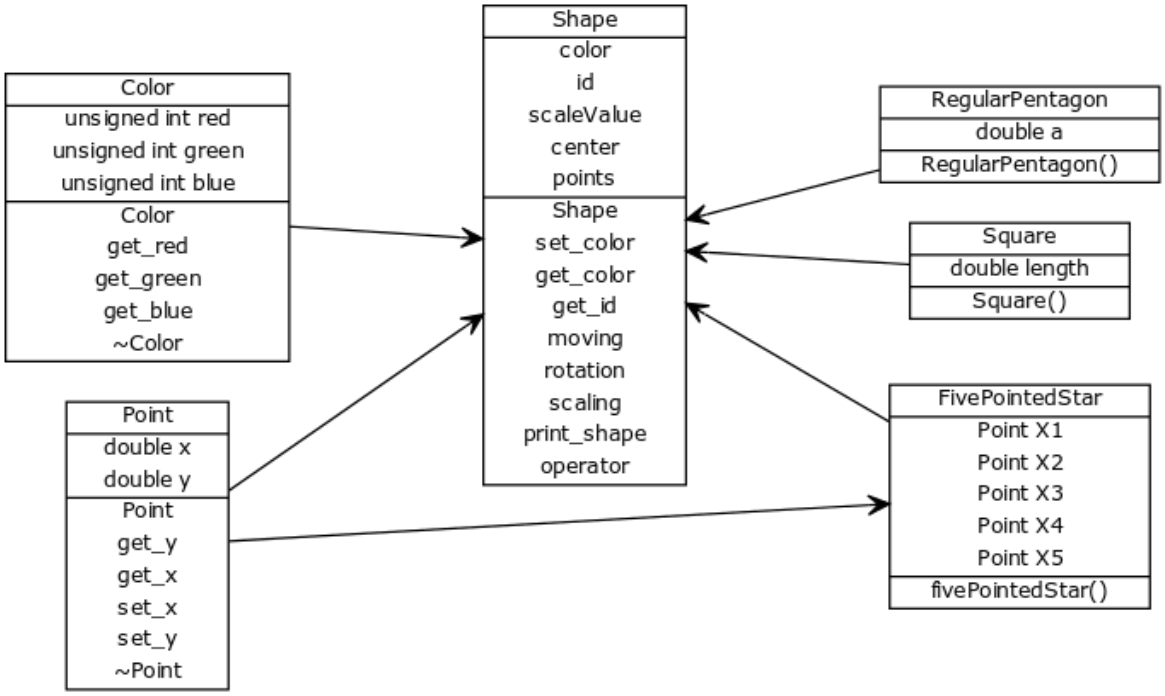
Для выполнения поставленной задачи были реализованы следующие классы:

1. Класс Point содержит два поля, которые описывают координаты x и y точки. Так же класс Point содержит методы для получения и установления координат.
2. Класс Color содержит три поля, в которых хранятся числа от 0 до 255 и характеризуют цвет фигуры. Так же 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 фигур. Класс Shape содержит следующие методы:
   - void set_color(Color color) для установления заданного цвета фигуры.

- Color get_color() const для получения информации об установленном цвете фигуры.
- unsigned long int get_id() const для получения информации об id фигуры.
- void moving(Point p) для смещения фигуры в заданную точку.
- void rotation(double grade) для поворота фигуры на заданный угол.
- virtual void scaling(double coefficient)=0 – чисто виртуальный метод для масштабирования фигуры на заданный коэффициент.
- virtual ostream& print_shape(ostream& stream, Shape& shape) = 0 – чисто виртуальный метод для вывода информации о фигуре на экран.
- friend ostream& operator << (ostream& stream, Shape& shape) для переопределения оператора вывода на экран.

4. Класс Square, который наследуется от абстрактного класса Shape. Класс имеет поле, которое характеризует длину стороны квадрата. В конструкторе данного класса вычисляются все вершины квадрата, имея информацию о координатах центра квадрата и длине его стороны. В классе был переопределен метод scaling, который масштабирует квадрат на заданный коэффициент. А так же метод print_shape, который выводит информацию о фигуре.
5. Класс fivePointedStar, который наследуется от абстрактного класса Shape. Класс fivePointedStar имеет пять дополнительных полей, которые характеризуют вершины пятиконечной звезды. В классе был переопределен метод scaling, который масштабирует звезду на заданный коэффициент. А так же метод print_shape, который выводит информацию о фигуре.
6. Класс RegularPentagon который наследуется от абстрактного класса Shape. Класс RegularPentagon имеет одно дополнительное поле, содержащих информацию о длине стороны. В классе был переопределен метод scaling, который масштабирует ромб на заданный коэффициент. А так же метод print_shape, который выводит информацию о фигуре.

UML диаграмма классов:



Вывод:

В ходе выполнения данной лабораторной работы была изучена тема наследование. Была спроектирована система классов для моделирования геометрических фигур. Были использованы виртуальные функции в иерархии наследования. Были разработаны классы, которые являются наследниками абстрактного класса Shape.

Приложение
Исходный код программы

```
#include <iostream>
#include <math.h>
#include <vector>

#define PI 3.14159265359

using namespace std;

class Point {
	double x;
	double y;
public:
	Point(double x=0, double y=0) : x(x), y(y) {};
	double get_x() const {
		return x;
	}
	double get_y() const {
		return y;
	}
	void set_x(double x) {
		this->x = x;
	}
	void set_y(double y) {
		this->y = y;
	}
};

class Color {
	unsigned int red;
	unsigned int green;
	unsigned int blue;
public:
	Color(unsigned int red, unsigned int green, unsigned int blue)
:red(red), green(green), blue(blue) {};
	unsigned int get_red() const {
		return red;
	}
	unsigned int get_green() const{
		return green;
	}
	unsigned int get_blue() const {
		return blue;
	}
};

void tabs() {
	cout << "\n\n";
}

class Shape {
protected:
	Color color;
	unsigned long int id;
	double scaleValue;
	Point center;
	vector <Point> points;
public:
```

```
      Shape(Color color, Point center):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;
      }

      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].set_x(tmp_x);
                  points[i].set_y(tmp_y);
            }
            center = p;
      }

       void rotation(double grade) {
            double grade_in_rad = grade*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].set_x(x);
                  points[i].set_y(y);
            }
      }

      virtual void scaling(double coefficient)=0;

      virtual ostream& print_shape(ostream& stream, Shape& shape) = 0;
      friend ostream& operator << (ostream& stream, Shape& shape) {
            return shape.print_shape(stream, shape);
      }
};

class Square : public Shape {
      double length;
public:
      Square(double lenght, Color color, Point center) :Shape(color,
center){
            this->length = length;
            points.push_back(Point(center.get_x() - lenght / 2,
center.get_y() - lenght / 2));
```

```
            points.push_back(Point(center.get_x() - lenght / 2,
center.get_y() + lenght / 2));
            points.push_back(Point(center.get_x() + lenght / 2,
center.get_y() + lenght / 2));
            points.push_back(Point(center.get_x() + lenght / 2,
center.get_y() - lenght / 2));
      }
      void scaling(double coefficient) override{
            double x;
            double y;
            for (size_t i = 0; i < points.size(); i++) {
                  x = center.get_x()+(points[i].get_x() -
center.get_x())*coefficient;
                  y = center.get_y()+(points[i].get_y() -
center.get_y())*coefficient;
                  points[i].set_x(x);
                  points[i].set_y(y);
            }
      }


      ostream& print_shape(ostream& stream, Shape& shape) override {
            stream << "Р¤РёРiСґСЂР°: РєРIР°РґСЂР°С," << endl;
            stream << "id С„РёРiСґСЂС‹: " << shape.get_id() << endl;
            stream << "Р¦РµРSС,СЂ С„РёРiСґСЂС‹: " << "(" << center.get_x()
<< ";" << center.get_y() << ")" << endl;
            stream << "РъРsРsСЂРґРёРSР°С,С‹ С„РёРiСґСЂС‹: " << endl;
            for (size_t i = 0; i < points.size(); i++) {
                  stream << "(" << points[i].get_x() << ";" <<
points[i].get_y() << ")" << endl;
            }
            stream << "Р¦РIРµС, С„РёРiСґСЂС‹: " <<
shape.get_color().get_red() << " " << shape.get_color().get_green() << " "
<< shape.get_color().get_blue() << endl;
            stream << "_____________________________________" << endl;
            stream << "_____________________________________" << endl;
            stream << "_____________________________________" << endl;
            stream << "_____________________________________" << endl;
            stream << "_____________________________________" << endl;
            return stream;
      }
};

class fivePointedStar : public Shape {
protected:
    Point X1, X2, X3, X4, X5;
public:
    fivePointedStar(double radius, Color color, Point center)
    : Shape(color, center) {

        double angle = 2 * PI / 5;
        X1.set_x(center.get_x() + radius * cos(angle * 0));
        X1.set_y(center.get_y() + radius * sin(angle * 0));
        X2.set_x(center.get_x() + radius * cos(angle * 1));
        X2.set_y(center.get_y() + radius * sin(angle * 1));
        X3.set_x(center.get_x() + radius * cos(angle * 2));
        X3.set_y(center.get_y() + radius * sin(angle * 2));
        X4.set_x(center.get_x() + radius * cos(angle * 3));
        X4.set_y(center.get_y() + radius * sin(angle * 3));
        X5.set_x(center.get_x() + radius * cos(angle * 4));
```

```
        X5.set_y(center.get_y() + radius * sin(angle * 4));

        points.push_back(X1); points.push_back(X2);
        points.push_back(X3); points.push_back(X4);
        points.push_back(X5);

    }

    void scaling(double k) {

        scaleValue *= k;
        center.set_x(center.get_x()*k);
        center.set_y(center.get_y()*k);

        X2.set_x(X2.get_x()*k);
        X2.set_y(X2.get_y()*k);
        X3.set_x(X3.get_x()*k);
        X3.set_y(X3.get_y()*k);
        X4.set_x(X4.get_x()*k);
        X4.set_y(X4.get_y()*k);
        X5.set_x(X5.get_x()*k);
        X5.set_y(X5.get_y()*k);
    }

      ostream& print_shape(ostream& stream, Shape& shape) override {
            stream << "P¤PëPiCґCЂP°: PiCЏC,PëPєPsPSPµC‡PSP°CЏ P·PIPµP·PrP°"
<< endl;
            stream << "id C„PëPiCґCЂC‹: " << shape.get_id() << endl;
            stream << "P¦PµPSC,CЂ C„PëPiCґCЂC‹: " << "(" << center.get_x()
<< ";" << center.get_y() << ")" << endl;
            stream << "PЪPsPsCЂPrPëPSP°C,C‹ C„PëPiCґCЂC‹: " << endl;
            for (size_t i = 0; i < points.size(); i++) {
                  stream << "(" << points[i].get_x() << ";" <<
points[i].get_y() << ")" << endl;
            }
            stream << "P¦PIPµC, C„PëPiCґCЂC‹: " <<
shape.get_color().get_red() << " " << shape.get_color().get_green() << " "
<< shape.get_color().get_blue() << endl;
                stream << "_____________________________________" << endl;
                stream << "_____________________________________" << endl;
                stream << "_____________________________________" << endl;
                stream << "_____________________________________" << endl;
                stream << "_____________________________________" << endl;
            return stream;
      }

};

class RegularPentagon : public Shape {
private:
      double a;
public:
      RegularPentagon(double length, Color color, Point center)
            :a(length), Shape(color, center) {
            double R = (sqrt(50 + 10 * sqrt(5))) / 10 * a;
            double angleR = 2 * PI / 5;
            points.push_back(Point(center.get_x() + R * cos(angleR * 0),
center.get_y() + R * sin(angleR * 0)));
            points.push_back(Point(center.get_x() + R * cos(angleR * 1),
center.get_y() + R * sin(angleR * 1)));
```

```
            points.push_back(Point(center.get_x() + R * cos(angleR * 2),
center.get_y() + R * sin(angleR * 2)));
            points.push_back(Point(center.get_x() + R * cos(angleR * 3),
center.get_y() + R * sin(angleR * 3)));
            points.push_back(Point(center.get_x() + R * cos(angleR * 4),
center.get_y() + R * sin(angleR * 4)));

      }
      void scaling(double k) override {
            this->a *= k;
            for (int i = 0; i < points.size(); i++) {
                  double dx = points[i].get_x() - center.get_x();
                  double dy = points[i].get_y() - center.get_y();
                  points[i].set_x(center.get_x() + k * dx);
                  points[i].set_y(center.get_y() + k * dy);
            }
      }
      ostream& print_shape(ostream& stream, Shape& shape) override {
            stream << "Р¤РёРiСѓСЂР°: РїСЂР°РІРёР»СЊРSС‹ РЊ
РїСЏС,РёСѓРiРsР»СЊРSРёРє" << endl;
            stream << "id С„РёРiСѓСЂС‹: " << shape.get_id() << endl;
            stream << "Р¦РµРSС,СЂ С„РёРiСѓСЂС‹: " << "(" << center.get_x()
<< ";" << center.get_y() << ")" << endl;
            stream << "РьРsРsСЂРґРёРSР°С,С‹ С„РёРiСѓСЂС‹: " << endl;
            for (size_t i = 0; i < points.size(); i++) {
                  stream << "(" << points[i].get_x() << ";" <<
points[i].get_y() << ")" << endl;
            }
            stream << "Р¦РIРµС, С„РёРiСѓСЂС‹: " <<
shape.get_color().get_red() << " " << shape.get_color().get_green() << " "
<< shape.get_color().get_blue() << endl;
                  stream << "_____________________________________" << endl;
                  stream << "_____________________________________" << endl;
                  stream << "_____________________________________" << endl;
                  stream << "_____________________________________" << endl;
                  stream << "_____________________________________" << endl;
            return stream;
      }
};

int main() {
      setlocale(LC_ALL, "Russian");

      tabs();
      Square square(2, { 255, 255, 255 }, { 1, 2 });
      cout << square;
      square.moving({21, 43});
      square.rotation(130);
      square.scaling(4);
      cout << square;

      tabs();
      fivePointedStar pent(3, { 255, 255, 255 }, {2, 3});
      cout << pent;
      pent.moving({0, 0});
      pent.rotation(30);
      pent.scaling(2);
      cout << pent;

      tabs();
```

```
    RegularPentagon figr(3, { 0, 128, 255}, {4, 5});
    cout << figr;
    figr.moving({21, 21});
    figr.rotation(20);
    figr.scaling(4);
    figr.set_color({200, 201, 202});
    cout << figr;

    return 0;
}
```