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.

Студентка гр.7304 _____ Каляева А.В

Преподаватель _____ Размочаева Н.В

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:
   3.1. void set_color(Color color) для установления заданного цвета фигуры.
   3.2. Color get_color() const для получения информации об установленном цвете фигуры.
   3.3. unsigned long int get_id() const для получения информации об id фигуры.
   3.4. void moving(Point p) для смещения фигуры в заданную точку.
   3.5. void rotation(double grade) для поворота фигуры на заданный угол.
   3.6. virtual void scaling(double coefficient)=0 – чисто виртуальный метод для масштабирования фигуры на заданный коэффициент.
   3.7. virtual ostream& print_shape(ostream& stream, Shape& shape) = 0 – чисто виртуальный метод для вывода информации о фигуре на экран.
   3.8. 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 Parallelogram, который наследуется от абстрактного класса Shape. Класс Parallelogram имеет три дополнительных поля, которые характеризуют вершины параллелограмма. В конструкторе данного класса вычисляется последняя вершина фигуры, а так же координаты ее центра. В классе был переопределен метод scaling, который масштабирует параллелограмм на заданный коэффициент. А так же метод print_shape, который выводит информацию о фигуре.
6. Класс Rhombus который наследуется от абстрактного класса Shape. Класс Rhombus имеет два дополнительных поля, содержащих информацию о длинах диагоналей ромба. В конструкторе данного класса вычисляются координаты вершин ромба, имея информацию о координатах одной из вершин и длинах диагоналей. В классе был переопределен метод scaling, который масштабирует ромб на заданный коэффициент. А так же метод print_shape, который выводит информацию о фигуре.

Обоснование решения:

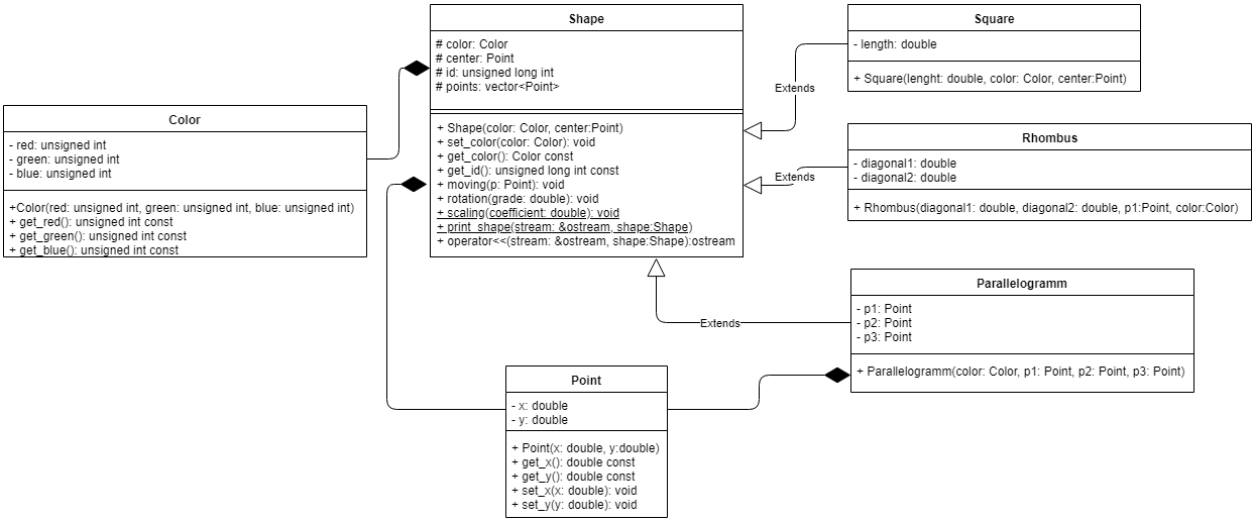
В данной лабораторной работе был реализован абстрактный класс Shape.

Поля цвет, координаты центра фигуры и координаты вершин фигуры являются общими, поэтому они содержатся в абстрактном классе Shape. Для реализации квадрата необходима информация о вершинах квадрата. Для получения данной информации достаточно знать центр квадрата и длину его стороны. Для реализации параллелограмма необходима информация о вершинах параллелограмма. Для получения данной информации достаточно знать информацию о трех точках. Для реализации ромба необходима информация о вершинах ромба. Для получения данной информации достаточно знать одну из точек ромба и длины двух диагоналей ромба.

Moving - это перемещение в заданную точку. Для любой фигуры можно найти расстояние между новой точкой и текущим центром фигуры. Перемещение фигуры – это смещение каждой из вершин этой фигура на полученное расстояние.

Rotation – это поворот на заданный угол. Для каждой фигуры поворот на заданный угол можно получить, умножив все координаты фигуры на матрицу поворота.

UML диаграмма классов:



Заключение :

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

Приложение
Исходный код программы

```
#include <iostream>
#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;
	}
};

class Shape {
protected:
	Color color;
	unsigned long int id;
	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);
		}
		length *= coefficient;
	}
	ostream& print_shape(ostream& stream, Shape& shape) override {
		stream << "Фигура: квадрат" << endl;
		stream << "id фигуры: " << shape.get_id() << endl;
		stream << "Координаты фигуры: " << endl;
		for (size_t i = 0; i < points.size(); i++) {
			stream << "(" << points[i].get_x() << ";" << points[i].get_y() << ")" <<
endl;
		}
		stream << "Цвет фигуры: " << shape.get_color().get_red() << " " <<
shape.get_color().get_green() << " " << shape.get_color().get_blue() << endl;
		stream << "_____" << endl;
		return stream;
	}
};

class Parallelogram :public Shape {
	Point p1;
	Point p2;
	Point p3;
public:
	Parallelogram(Color color, Point p1, Point p2, Point p3) :Shape(color, center) {
		this->p1 = p1;
		this->p2 = p2;
		this->p3 = p3;
		points.push_back(p1);
		points.push_back(p2);
		points.push_back(p3);
		points.push_back(Point(points[0].get_x() + (points[2].get_x() - points[1].get_x()),
points[2].get_y() - (points[1].get_y() - points[0].get_y())));
		center.set_x((points[0].get_x() + points[2].get_x()) / 2);
		center.set_y((points[0].get_y() + points[2].get_y()) / 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 << "Фигура: параллелограмм" << endl;
		stream << "id фигуры: " << shape.get_id() << endl;
		stream << "Центр фигуры: " << "(" << center.get_x() << ";" << center.get_y()
<< ")" << endl;
		stream << "Координаты фигуры: " << endl;
		for (size_t i = 0; i < points.size(); i++) {
			stream << "(" << points[i].get_x() << ";" << points[i].get_y() << ")" <<
endl;
		}
		stream << "Цвет фигуры: " << shape.get_color().get_red() << " " <<
shape.get_color().get_green() << " " << shape.get_color().get_blue() << endl;
		stream << "_____" << endl;
		return stream;
	}

};

class Rhombus : public Shape {
	double diagonal1;
	double diagonal2;
public:
	Rhombus(double diagonal1, double diagonal2, Point p1, Color color) :Shape(color,
center) {
		this->diagonal1 = diagonal1;
		this->diagonal2 = diagonal2;
		double x = p1.get_x();
		double y = p1.get_y() + diagonal1;
		points.push_back(p1);
		points.push_back(Point(p1.get_x() - diagonal2 / 2, diagonal1 / 2));
		points.push_back(Point(x, y));
		points.push_back(Point(p1.get_x() + diagonal2 / 2, diagonal1 / 2));
		center.set_x((points[1].get_x() + points[3].get_x()) / 2);
		center.set_y((points[0].get_y() + points[2].get_y()) / 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);
            }
            diagonal1 *= coefficient;
            diagonal2 *= coefficient;
        }
        ostream& print_shape(ostream& stream, Shape& shape) override {
            stream << "Фигура: ромб" << endl;
            stream << "id фигуры: " << shape.get_id() << endl;
            stream << "Центр фигуры: " << "(" << center.get_x() << ";" << center.get_y()
<< ")" << endl;
            stream << "Координаты фигуры: " << endl;
            for (size_t i = 0; i < points.size(); i++) {
                stream << "(" << points[i].get_x() << ";" << points[i].get_y() << ")" <<
endl;
            }
            stream << "Цвет фигуры: " << shape.get_color().get_red() << " " <<
shape.get_color().get_green() << " " << shape.get_color().get_blue() << endl;
            stream << "_____" << endl;
            return stream;
        }
};

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

        Square test1(2, { 255, 255, 255 }, { 1, 2 });
        cout << test1;
        test1.rotation(30);
        cout << test1;
        Parallelogram test2({ 128,35,127 }, { 1, 5 }, { -1,9 }, { 7,9 });
        cout << test2;
        test2.scaling(2);
        cout << test2;
        Rhombus test3(7, 3, { 0,0 }, { 123, 23, 77 });
        cout << test3;
        test3.moving({ 3, 6 });
        cout << test3;
        return 0;
}
```