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 _____ Находько А.Ю.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы.

Ознакомиться с понятиями наследование, полиморфизм, абстрактный класс, изучить виртуальные функции, принцип их работы, способ организации в памяти, раннее и позднее связывания в языке C++. В соответствии с индивидуальным заданием разработать систему классов для представления геометрических фигур.

Задание.

Вариант 14.

Реализовать систему классов для фигур:

1. Круг
2. Трапеция
3. Квадрат

Пояснение задачи.

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:

1. условие задания;
2. UML диаграмму разработанных классов;

3. текстовое обоснование проектных решений;
4. реализацию классов на языке C++.

Текстовое обоснование разработанных классов.

class Point – класс, который хранит координаты x,y точки, также содержит методы с помощью которых можно установить либо получить координату.

class Colour – класс, содержащий значения RGB, которые характеризуют цвет фигуры, также содержит методы с помощью которых можно получить информацию о цвете.

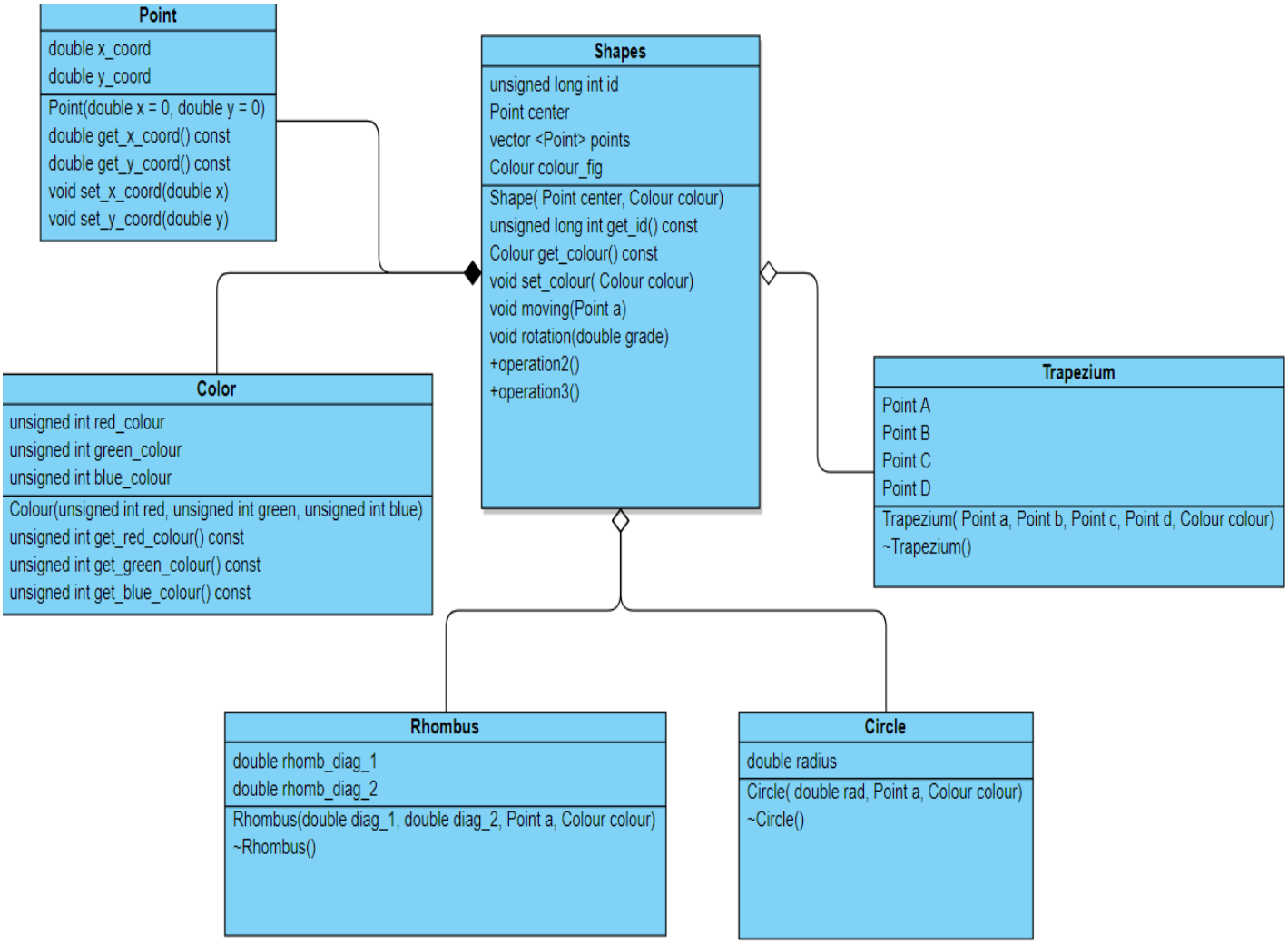
class Shape – абстрактный класс, содержащий методы для перемещения в указанные координаты, поворота на заданный угол, масштабирования на заданный коэффициент, установки и получения цвета, а также оператор вывода в поток. Также содержит информацию о фигуре: номер, координата центра, вектор с вершинами фигуры и цвет.

class Rhombus – класс, наследуемый от абстрактного Shape. Содержит 2 дополнительных защищённых поля для хранения длин диагоналей. Также содержит переопределённые методы масштабирования и метод вывода информации о фигуре.

class Circle – класс, наследуемый от абстрактного Shape. Содержит 1 защищённое поле для хранения радиуса круга. Также содержит переопределённые методы, как в классе вышеописанном классе.

class Trapezium – класс, наследуемый от абстрактного Shape. Содержит 4 дополнительных защищённых поля для хранения вершин трапеции. Также содержит переопределённые методы, как в классе вышеописанном классе.

Диаграмма.



Выводы.

В результате выполнения лабораторной работы ознакомился с понятиями наследование, полиморфизм, абстрактный класс, изучил виртуальные функции, принцип их работы, способ организации в памяти, раннее и позднее связывания в языке C++. Также выполнил индивидуальное задание для данных к л/р геометрических фигур.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

Lab1.cpp

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

#define PI 3.14159265359

using namespace std;

class Point
{

double x_coord;
double y_coord;

public:

Point(double x = 0, double y = 0) : x_coord(x), y_coord(y) {};

double get_x_coord() const
{
return x_coord;
}

double get_y_coord() const
{
return y_coord;
}

void set_x_coord(double x)
{
this->x_coord = x;
}

void set_y_coord(double y)
{
this->y_coord = y;
}

};

class Colour
{

unsigned int red_colour;
unsigned int green_colour;
unsigned int blue_colour;

public:
```

```
Colour(unsigned int red, unsigned int green, unsigned int blue) : red_colour(red),
green_colour(green), blue_colour(blue) {};

unsigned int get_red_colour() const
{
return red_colour;
}

unsigned int get_green_colour() const
{
return green_colour;
}

unsigned int get_blue_colour() const
{
return blue_colour;
}

};

class Shape
{

protected:

unsigned long int id;
Point center;
vector <Point> points;
Colour colour_fig;

public:

Shape( Point center, Colour colour) : center(center), colour_fig(colour)
{
static long int i = 0;
id = i;
i++;
}

unsigned long int get_id() const
{
return id;
}

Colour get_colour() const
{
return colour_fig;
}

void set_colour( Colour colour)
{
this->colour_fig = colour;
```

```
}

void moving(Point a)
{
double offset_x = a.get_x_coord() - center.get_x_coord();
double offset_y = a.get_y_coord() - center.get_y_coord();
for(size_t i = 0; i < points.size(); i++)
{
double new_x = points[i].get_x_coord() + offset_x;
double new_y = points[i].get_y_coord() + offset_y;
points[i].set_x_coord(new_x);
points[i].set_y_coord(new_y);
}
center = a;
}

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_coord()+(points[i].get_x_coord()-
center.get_x_coord())*cos(grade_in_rad)-(points[i].get_y_coord()-
center.get_y_coord())*sin(grade_in_rad);
double y = center.get_y_coord() + (points[i].get_x_coord() -
center.get_x_coord())*sin(grade_in_rad) + (points[i].get_y_coord() -
center.get_y_coord())*cos(grade_in_rad); ;
points[i].set_x_coord(x);
points[i].set_y_coord(y);
}
}

virtual void scaling(double coefficient)=0;

virtual ostream& figure_info(ostream& stream, Shape& figure) = 0;
friend ostream& operator « (ostream& stream, Shape& figure) {
return figure.figure_info(stream, figure);
}

};

class Rhombus : public Shape
{

double rhomb_diag_1;
double rhomb_diag_2;

public:

Rhombus(double diag_1, double diag_2, Point a, Colour colour) : Shape(center, colour) ,
rhomb_diag_1(diag_1) , rhomb_diag_2(diag_2)
{
points.push_back(a);
points.push_back(Point(a.get_x_coord() - (rhomb_diag_2 / 2), a.get_y_coord() + (rhomb_diag_1
```

```
/ 2)));
points.push_back(Point(a.get_x_coord(), a.get_y_coord() + rhomb_diag_1));
points.push_back(Point(a.get_x_coord() + (rhomb_diag_2 / 2), a.get_y_coord() +
(rhomb_diag_1 / 2)));
center.set_x_coord((points[1].get_x_coord() + points[3].get_x_coord()) / 2);
center.set_y_coord((points[0].get_y_coord() + points[2].get_y_coord()) / 2);
}

~Rhombus()
{
points.clear();
}

void scaling(double coeff) override
{
double x_coord, y_coord;
for(size_t i = 0; i < points.size(); i++)
{
x_coord = center.get_x_coord() + (points[i].get_x_coord() - center.get_x_coord())*coeff;
y_coord = center.get_y_coord() + (points[i].get_y_coord() - center.get_y_coord())*coeff;
points[i].set_x_coord(x_coord);
points[i].set_y_coord(y_coord);
}
}

ostream& figure_info(ostream& stream, Shape& figure) override
{
stream «
"*****************************************************************************************" «
endl;
stream « "Информация о фигуре:" « endl;
stream « "Фигура — ромб" « endl;
stream « "id: " « figure.get_id() « endl;
stream « "Цвет фигуры: " « figure.get_colour().get_red_colour() « " " «
figure.get_colour().get_green_colour() « " " « figure.get_colour().get_blue_colour() « endl;
stream « "Центр фигуры находится в точке с

координатами: " « "(" « center.get_x_coord() « ";" « center.get_y_coord() « ")" « endl;
stream « "Фигура находится в следующих координатах: " « endl;
for(size_t i = 0; i < points.size(); i++)
{
stream « "(" « points[i].get_x_coord() « ";" « points[i].get_y_coord() « ")" « endl;
}
stream «
"*****************************************************************************************" «
endl;
return stream;
}

};

class Circle : public Shape
{
```

```
double radius;

public:

Circle( double rad, Point a, Colour colour) : Shape(center, colour), radius(rad)
{
center = a;
points.push_back(Point(a.get_x_coord() - radius, a.get_y_coord()));
points.push_back(Point(a.get_x_coord(), a.get_y_coord() + radius));
points.push_back(Point(a.get_x_coord() + radius, a.get_y_coord()));
points.push_back(Point(a.get_x_coord(), a.get_y_coord() - radius));

}

void scaling(double coeff) override
{
double x_coord, y_coord;
for(size_t i = 0; i < points.size(); i++)
{
x_coord = center.get_x_coord() + (points[i].get_x_coord() - center.get_x_coord())*coeff;
y_coord = center.get_y_coord() + (points[i].get_y_coord() - center.get_y_coord())*coeff;
points[i].set_x_coord(x_coord);
points[i].set_y_coord(y_coord);
}
}

~Circle()
{
points.clear();
}

ostream& figure_info(ostream& stream, Shape& figure) override
{
stream «
"*****************************************************************************************" «
endl;
stream « "Информация о фигуре:" « endl;
stream « "Фигура — круг" « endl;
stream « "id: " « figure.get_id() « endl;
stream « "Цвет фигуры: " « figure.get_colour().get_red_colour() « " " «
figure.get_colour().get_green_colour() « " " « figure.get_colour().get_blue_colour() « endl;
stream « "Центр фигуры находится в точке с координатами: " « "(" « center.get_x_coord()
« ";" « center.get_y_coord() « ")" « endl;
stream « "Фигура находится в следующих координатах: " « endl;
for(size_t i = 0; i < points.size(); i++)
{
stream « "(" « points[i].get_x_coord() « ";" « points[i].get_y_coord() « ")" « endl;
}
stream «
"*****************************************************************************************" «
endl;
return stream;
```

```
}

};

class Trapezium : public Shape
{
Point A;
Point B;
Point C;
Point D;

// B C
//
//
// A D

public:
Trapezium( Point a, Point b, Point c, Point d, Colour colour) : Shape(center, colour) , A(a), B(b),
C(c), D(d)
{
points.push_back(A);
points.push_back(B);
points.push_back(C);
points.push_back(D);
double k = sqrt((B.get_x_coord() - A.get_x_coord()) * (B.get_x_coord() - A.get_x_coord()) +
(B.get_y_coord() - A.get_y_coord()) * (B.get_y_coord() - A.get_y_coord()));
double p = sqrt((C.get_x_coord() - D.get_x_coord()) * (C.get_x_coord() - D.get_x_coord()) +
(C.get_y_coord() - D.get_y_coord()) * (C.get_y_coord() - D.get_y_coord()));
center.set_x_coord((D.get_x_coord() - A.get_x_coord()) / 2 + ((2 * (C.get_x_coord() -
B.get_x_coord()) + (D.get_x_coord() - A.get_x_coord())) * (pow(k, 2) - pow(p, 2))) / (6 *
(pow(D.get_x_coord() - A.get_x_coord(), 2) - pow(C.get_x_coord() - B.get_x_coord(), 2))) +
A.get_x_coord());
center.set_y_coord((sqrt(pow(B.get_x_coord() - B.get_x_coord(), 2) + pow(B.get_y_coord() -
A.get_y_coord(), 2))) * ((D.get_x_coord() - A.get_x_coord()) + 2 * (C.get_x_coord() -
B.get_x_coord())) / (3 * ((C.get_x_coord() - B.get_x_coord()) + (D.get_x_coord() -
A.get_x_coord()))) + A.get_y_coord());
}

void scaling(double coeff) override
{
double x_coord, y_coord;
for(size_t i = 0; i < points.size(); i++)
{
x_coord = center.get_x_coord() + (points[i].get_x_coord() - center.get_x_coord())*coeff;
y_coord = center.get_y_coord() + (points[i].get_y_coord() - center.get_y_coord())*coeff;
points[i].set_x_coord(x_coord);
points[i].set_y_coord(y_coord);
}
}

~Trapezium()
{
points.clear();
```

```
}

ostream& figure_info(ostream& stream, Shape& figure) override
{
stream «
"*********************************************************************************************" «
endl;
stream « "Информация о фигуре:" «

endl;
stream « "Фигура — трапеция" « endl;
stream « "id: " « figure.get_id() « endl;
stream « "Цвет фигуры: " « figure.get_colour().get_red_colour() « " " «
figure.get_colour().get_green_colour() « " " « figure.get_colour().get_blue_colour() « endl;
stream « "Центр фигуры находится в точке с координатами: " « "(" « center.get_x_coord()
« ";" « center.get_y_coord() « ")" « endl;
stream « "Фигура находится в следующих координатах: " « endl;
for(size_t i = 0; i < points.size(); i++)
{
stream « "(" « points[i].get_x_coord() « ";" « points[i].get_y_coord() « ")" « endl;
}
stream «
"*********************************************************************************************" «
endl;
return stream;
}

};

int main()
{
Rhombus r_figure(10, 20 , {30, 30}, { 187, 255, 211 });
cout « r_figure;
cout « "Поворот" « endl;
r_figure.rotation(30);
cout « r_figure;
cout « "Масштабирование" « endl;
r_figure.scaling(2);
cout « r_figure;
cout « "Перемещение" « endl;
r_figure.moving({ 3,6 });
cout « r_figure;
Circle c_figure( 5, {0, 0}, { 255, 255, 255 });
cout « c_figure;
cout « "Масштабирование" « endl;
c_figure.scaling(3);
cout « c_figure;
cout « "Перемещение" « endl;
c_figure.moving({ 1,8 });
cout « c_figure;
Trapezium t_figure({0, 0}, {3, 5}, {7, 5}, {10, 0}, { 14, 218, 111 });
cout « t_figure;
cout « "Поворот" « endl;
```

```
t_figure.rotation(40);
cout « t_figure;
cout « "Масштабирование" « endl;
t_figure.scaling(4);
cout « t_figure;
cout « "Перемещение" « endl;
t_figure.moving({ 2,10 });
cout « t_figure;
return 0;
}
```