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. 7382 _____ Государкин Я.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Проектировка иерархии классов для моделирования геометрических фигур.

Основные теоретические положения.

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;

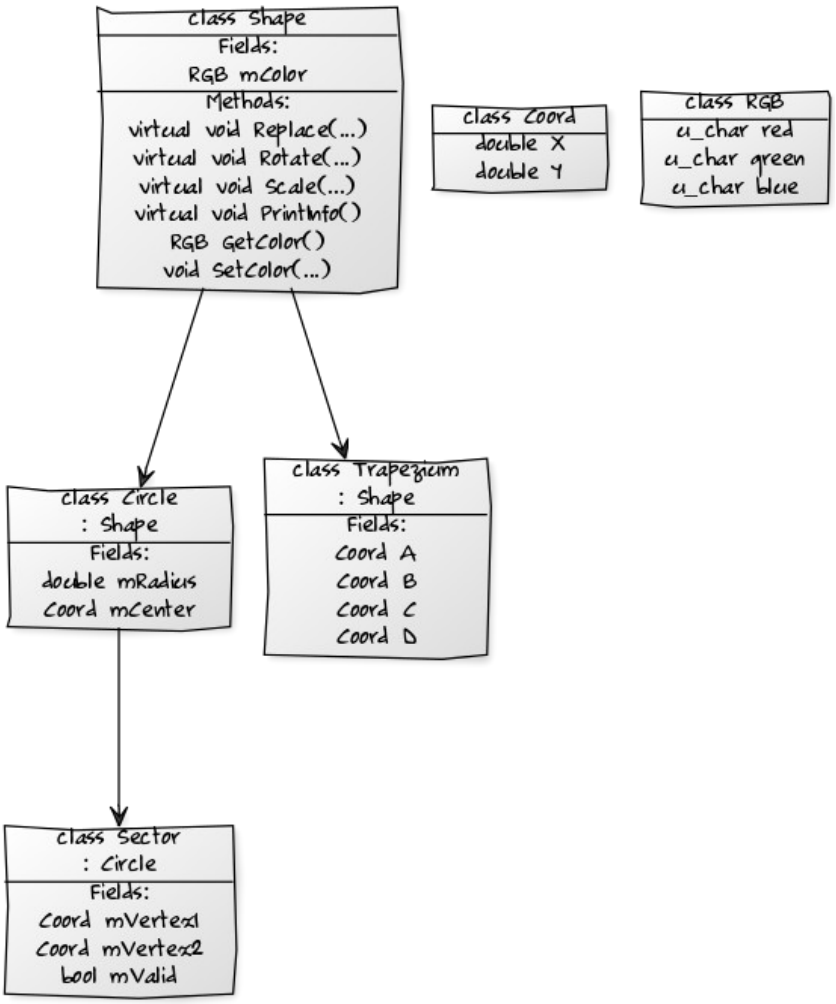
реализацию классов на языке C++.

Выводы.

В ходе выполнения лабораторной работы была реализованна система классов для моделирования геометрических фигур с помощью методологии объектно-ориентированного программирования.

ПРИЛОЖЕНИЕ

UML-диаграмма классов



Исходный код

main.cpp

```
#include <iostream>
#include <vector>
#include <chrono>

#include "shape.h"

int main()
{
    Circle example(Coord(1, 1), 3);
    example.SetColor(RGB(14, 68, 145));
    example.Replace(Coord(3.35,9.1));
    example.Scale(1.55);
    example.Rotate(PI, Coord(0,0));
    example.PrintInfo();

    Trapezium tr(Coord(-1, 2), Coord(1, 2), Coord(2, -1), Coord(-2, -1));
    tr.Rotate(PI, Coord(-2, -1));
```

```
    tr.PrintInfo();

    return 0;
}
```

shape.cpp

```
#include "shape.h"

// Class Circle methods
// ____________________________________________________

void Circle::PrintInfo()
{
    RGB color = GetColor();
    std::cout << "Circle info: \n"
              << "\tRadius: ["<< mRadius <<"]\n"
              << "\tCenter: ["<< mCenter.X << ", " << mCenter.Y << "]\n"
              << "\tColor: " << color << "\n";
}

void Circle::Replace(Coord vector) { mCenter += vector; }

void Circle::Rotate(double angle, const Coord& rotCenter) {
mCenter.Rotate(angle, rotCenter); }

void Circle::Scale(double factor) { mRadius *= factor; }

// Class Circle sector methods
// ____________________________________________________

void Sector::PrintInfo()
{
    RGB color = GetColor();

    std::cout << "Circle sector info: \n"
              << "\tRadius: ["<< mRadius <<"]\n"
              << "\tCenter: ["<< mCenter.X << ", " << mCenter.Y << "]\n"
              << "\tVertex1: ["<< mVertex1.X << ", " << mVertex1.Y << "]\n"
              << "\tVertex2: ["<< mVertex2.X << ", " << mVertex2.Y << "]\n"
              << "\tColor: " << color << "\n";
    if(!mValid)
        std::cout << "[WARNING] This Sector - invalid constructed!\n";
}

void Sector::Replace(Coord vector) { mCenter += vector; mVertex1 += vector;
mVertex2 += vector; }

void Sector::Rotate(double angle, const Coord& rotCenter) {
mCenter.Rotate(angle, rotCenter);

mVertex1.Rotate(angle, rotCenter);

mVertex2.Rotate(angle, rotCenter); }
void Sector::Scale(double factor) { mRadius *= factor; }

// Class Trapezium methods
// ____________________________________________________
```

```
void Trapezium::PrintInfo()
{
    RGB color = GetColor();
    std::cout << "Trapezium info: \n"
              << "\tA point: ["<< A.X << ", " << A.Y << "]\n"
              << "\tB point: ["<< B.X << ", " << B.Y << "]\n"
              << "\tC point: ["<< C.X << ", " << C.Y << "]\n"
              << "\tD point: ["<< D.X << ", " << D.Y << "]\n"
              << "\tColor: " << color << "\n";
    if(!mValid)
        std::cout << "[WARNING] This Trapezium - invalid constructed!\n";
}

void Trapezium::Replace(Coord vector) { A += vector; B += vector; C += vector; D
+= vector; }

void Trapezium::Rotate(double angle, const Coord& rotCenter) { A.Rotate(angle,
rotCenter); B.Rotate(angle, rotCenter);
                                                               C.Rotate(angle,
rotCenter); D.Rotate(angle, rotCenter); }

void Trapezium::Scale(double factor)
{
    Coord ABvec(B.X-A.X, B.Y-A.Y); ABvec *= factor/2; B += ABvec;
    Coord ACvec(C.X-A.X, C.Y-A.Y); ACvec *= factor/2; C += ACvec;
    Coord ADvec(D.X-A.X, D.Y-A.Y); ADvec *= factor/2; D += ADvec;
}
```

shape.h

```
#ifndef SHAPE_H
#define SHAPE_H

#pragma once

#include "math.h"

// BASIC CLASS
class Shape
{
public:
    Shape()
        : mColor(255, 255, 255) {}

    RGB GetColor() const { return mColor; }
    void SetColor(const RGB& color) { mColor = color; }

protected:
    virtual void Replace(Coord vector) = 0;
    virtual void Rotate(double angle, const Coord& rotCenter) = 0;  // angle in
rad
    virtual void Scale(double factor) = 0;
    virtual void PrintInfo() = 0;

    virtual ~Shape() {}

private:
    RGB mColor;
};
```

```
// TRAPEZIUM CLASS
class Trapezium : public Shape
{
public:
    Trapezium() : Shape()
        , A(-1, 2)
        , B(1, 2)
        , C(2, -1)
        , D(-2, -1) {}
    Trapezium(Coord a, Coord b, Coord c, Coord d) : Shape()
        , A(a)
        , B(b)
        , C(c)
        , D(d)
    {
        if( fabs(fabs(B.X-A.X)/fabs(C.X-D.X) - fabs(B.Y-A.Y)/fabs(C.Y-D.Y)) <=
0.000001
         || fabs(fabs(A.X-D.X)/fabs(B.X-C.X) - fabs(A.Y-D.Y)/fabs(B.Y-C.Y)) <=
0.000001 )
            mValid = true;
        else
            mValid = false;

    }

    void Replace(Coord vector) override;
    void Rotate(double angle, const Coord& rotCenter) override;
    void Scale(double factor) override;
    void PrintInfo() override;

private:
    Coord A, B;
    Coord C, D;
    bool mValid;
};

// CIRCLE CLASS
class Circle : public Shape
{
public:
    Circle() : Shape()
        , mCenter(0,0)
        , mRadius(1) {}
    Circle(Coord p, double rad) : Shape()
        , mCenter(p)
        , mRadius(rad) {}

    void Replace(Coord vector) override;
    void Rotate(double angle, const Coord& rotCenter) override;
    void Scale(double factor) override;
    void PrintInfo() override;

protected:
    Coord mCenter;
    double mRadius;
};

// CIRCLE SECTOR CLASS
class Sector : public Circle
{
public:
```

```
    Sector(Coord a, Coord b) : Circle()
        , mVertex1(a)
        , mVertex2(b)
    {
        if(fabs(Coord::Lenght(mCenter, mVertex1) - Coord::Lenght(mCenter,
mVertex2)) <= 0.00001)
        {
            mRadius = Coord::Lenght(mCenter, mVertex1);
            mValid = true;
        }
        else {
            mValid = false;
        }
    }

    void Replace(Coord vector) override;
    void Rotate(double angle, const Coord& rotCenter) override;
    void Scale(double factor) override;
    void PrintInfo() override;

private:
    Coord mVertex1, mVertex2;
    bool mValid;
};

#endif // SHAPE_H
```

math.h

```
#ifndef MATH_H
#define MATH_H

#pragma once

#include <iostream>
#include <cmath>

#define PI 3.141592 // 180 grad

using namespace std;

typedef unsigned char u_char;
typedef unsigned long ul;

class RGB
{
public:
    RGB(u_char red, u_char green, u_char blue) {mColor[0] = red; mColor[1] =
green; mColor[2] = blue;}

    friend std::ostream& operator <<(std::ostream& stream, const RGB& color) {
        stream << "[" << static_cast<int>(color.mColor[0]) << ", " <<
static_cast<int>(color.mColor[1]) << ", " << static_cast<int>(color.mColor[2])
<< "]";
        return stream;
    }
```

```
    u_char mColor[3];
};

class Coord
{
public:
    Coord(double x, double y)
        : X(x)
        , Y(y) {}

    Coord& operator +=(const Coord& vec) { X += vec.X; Y+= vec.Y; return *this;
}
    Coord& operator *=(double factor) { X *= factor; Y*= factor; return *this; }

    static double Lenght(const Coord& p1, const Coord& p2) { return sqrt((p1.X -
p2.X)*(p1.X - p2.X) + (p1.Y - p2.Y)*(p1.Y - p2.Y)); }

    void Rotate(double angle, const Coord& p0)
    {
      double x = X;
      X = p0.X + (x - p0.X)*cos(angle) - (Y - p0.Y)*sin(angle);
      Y = p0.Y + (Y - p0.Y)*cos(angle) + (x - p0.X)*sin(angle);
    }

    double X;
    double Y;
};

#endif // MATH_H
```