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. 7381 _____ Кушкоева А.О.

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.

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:

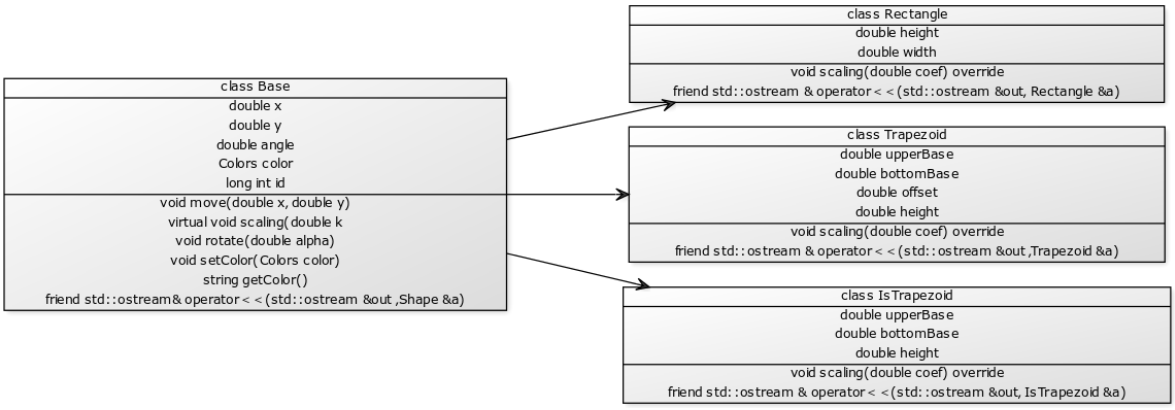
1. Условие задания;
2. UML диаграмму разработанных классов;
3. Текстовое обоснование проектных решений;
4. Реализацию классов на языке C++.

Индивидуализация.

Вариант 10 – реализовать систему классов для фигур:

1. Прямоугольник;
2. Трапеция;
3. Равнобедренная трапеция.

UML диаграмму разработанных классов.



Обоснование решения.

В данной лабораторной работе был реализован абстрактный класс Shape. Цвет, координаты левого нижнего угла фигуры и угол поворота являются общими, поэтому они содержатся в абстрактном классе Shape.

Для реализации прямоугольника необходима информация о ширине и высоте.

Для реализации трапеции необходима информация о длине верхнего и нижнего оснований, о смещении верхнего основания относительно нижнего и о его высоте.

Для реализации равнобедренной трапеции необходима информация о его высоте, о длине верхнего и нижнего оснований. В отличии от общего случая трапеции знать о смещении верхнего основания относительно нижнего не нужно, эту информация можно получить исходя из равенства боковых сторон.

Реализацию классов на языке C++.

Реализация классов на языке C++ представлена в приложении А.

Выводы.

В ходе выполнения лабораторной работы была спроектирована система классов для работы с геометрическими фигурами в соответствии с

индивидуальным заданием. В иерархии наследования были использованы виртуальные функции, базовый класс при этом является виртуальным (класс называется виртуальным, если содержит хотя бы одну виртуальную функцию). Были реализованы методы перемещения фигуры в заданные координаты, поворота на заданный угол, масштабирования на заданный коэффициент, была реализована однозначная идентификация объекта.

ПРИЛОЖЕНИЕ А

РЕАЛИЗАЦИЯ КЛАССОВ НА ЯЗЫКЕ С++

```
#include "pch.h"
#include <iostream>
#include <cmath>
#include <string>

using namespace std;

enum  Colors{ RED, GREEN, BLUE, NOCOLOR};

class Shape
{
public:
	Shape(double x = 0, double y = 0, double angle = 0, Colors color = Colors::NOCOLOR)
		: x(x)
		, y(y)
		, angle(angle)
		, color(color)
	{
		if (angle >= 360.0)
			this->angle = angle - int(angle / 360) * 360;
		else
			this->angle = angle;

		idCounter++;
		id = idCounter;
	}

	void move(double x, double y)
	{
		this->x = x;
		this->y = y;
	}

	virtual void scale(double k) = 0;

	void rotate(double alpha)
	{
		if (alpha >= 360)
			alpha = alpha - (alpha / 360) * 360;
		if (angle + alpha < 360.0)
			angle += alpha;
		else
			angle = (angle + alpha) - 360;
	}

	void setColor(Colors color)
	{
		this->color = color;
	}

	string getColor()
	{
		switch (color)
		{
		case Colors::RED :
			return "Red";
		case Colors::GREEN :
			return "Green";
```

```
                case Colors::BLUE :
                        return "Blue";
                case Colors::NOCOLOR :
                        return "No color";
                default:
                        return "Unknown color";
                }
        }

        friend std::ostream & operator << (std::ostream &out, Shape &a)
        {
                out << "Id:" << a.id << endl;
                out << "Color:" << a.getColor() << endl;
                out << "Angle:" << a.angle << " degrees" << endl;
                out << "x1:" << a.x << " y1:" << a.y << endl;
                return out;
        }

protected:
        double x;
        double y;
        double angle;
        Colors color;
        long int id;

private:
        static long int idCounter;
};

long int Shape::idCounter = 0;

class Rectangle : public Shape
{
public:
        Rectangle(double height, double width, double x = 0, double y = 0, double angle = 0,
Colors color = Colors::NOCOLOR)
                : Shape(x, y, angle, color)
                , height(height)
                , width(width)
        {
        }

        void scale(double coef) override
        {
                height *= coef;
                width *= coef;
        }
        friend std::ostream & operator <<(std::ostream &out, Rectangle &a)
        {
                out << "Rectangle" << endl;
                out << (Shape&) a;
                out << "x2:" << a.x << " y2:" << a.y + a.height << std::endl;
                out << "x3:" << a.x + a.width << " y3:" << a.y + a.height << std::endl;
                out << "x4:" << a.x + a.width << " y4:" << a.y << std::endl;

                return out;
        }
private:
        double height;
        double width;
};


class Trapezoid : public Shape
```

```
{
public:
	Trapezoid(double upper, double bottom, double offset, double height, double x = 0,
double y = 0, double angle = 0, Colors color = Colors::NOCOLOR)
		: Shape(x, y, angle, color)
		, upperBase(upper)
		, bottomBase(bottom)
		, offset(offset)
		, height(height)
	{
		if (upper > bottom)
		{
			upperBase = bottom;
			bottomBase = upper;
		}
	}

	void scale(double coef) override
	{
		upperBase *= coef;
		bottomBase *= coef;
		offset *= coef;
		height *= coef;
	}

	friend std::ostream & operator <<(std::ostream &out, Trapezoid &a)
	{
		out << "Trapezoid" << endl;
		out << (Shape&)a;
		out << "x2:" << (a.x + a.offset) << " y2:" << (a.y + a.height) << std::endl;
		out << "x3:" << (a.x + a.offset + a.upperBase) << " y3:" << (a.y + a.height)
<< std::endl;
		out << "x4:" << (a.x + a.bottomBase) << " y4:" << a.y  << std::endl;

		return out;
	}
private:
	double upperBase;
	double bottomBase;
	double offset;
	double height;
};

class IsTrapezoid : public Shape
{
public:
	IsTrapezoid(double upper, double bottom, double height, double x = 0, double y = 0,
double angle = 0, Colors color = Colors::NOCOLOR)
		: Shape(x, y, angle, color)
		, upperBase(upper)
		, bottomBase(bottom)
		, height(height)
	{
		if(upper > bottom)
		{
			upperBase = bottom;
			bottomBase = upper;
	}
	}

	void scale(double coef) override
	{
		upperBase *= coef;
		bottomBase *= coef;
```

```
                height *= coef;
        }
        friend std::ostream & operator <<(std::ostream &out, IsTrapezoid &a)
        {
                out << "Isosceles trapezoid" << endl;
                out << (Shape&)a;
                out << "x2:" << a.x + (a.upperBase - a.bottomBase)/2 << " y2:" << a.y +
a.height << std::endl;
                out << "x3:" << a.x + (a.upperBase + a.bottomBase)/2 << " y3:" << a.y +
a.height << std::endl;
                out << "x4:" << a.x + a.bottomBase << " y4:" << a.y << std::endl;

                return out;
        }

private:
        double upperBase;
        double bottomBase;
        double height;
        };
```