МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по практической работе №2

по дисциплине «Объектно-ориентированное программирование»

Тема: Наследование

Студент гр. 7382 _____ Токарев А.П.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы.

Ознакомиться с понятиями наследование, полиморфизм, абстрактный класс, изучить виртуальные функции, принцип их работы, способы реализации. В соответствии с индивидуальным заданием разработать систему классов

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:

- условие задания;
- UML диаграмму разработанных классов;
- текстовое обоснование проектных решений;
- реализацию классов на языке C++.

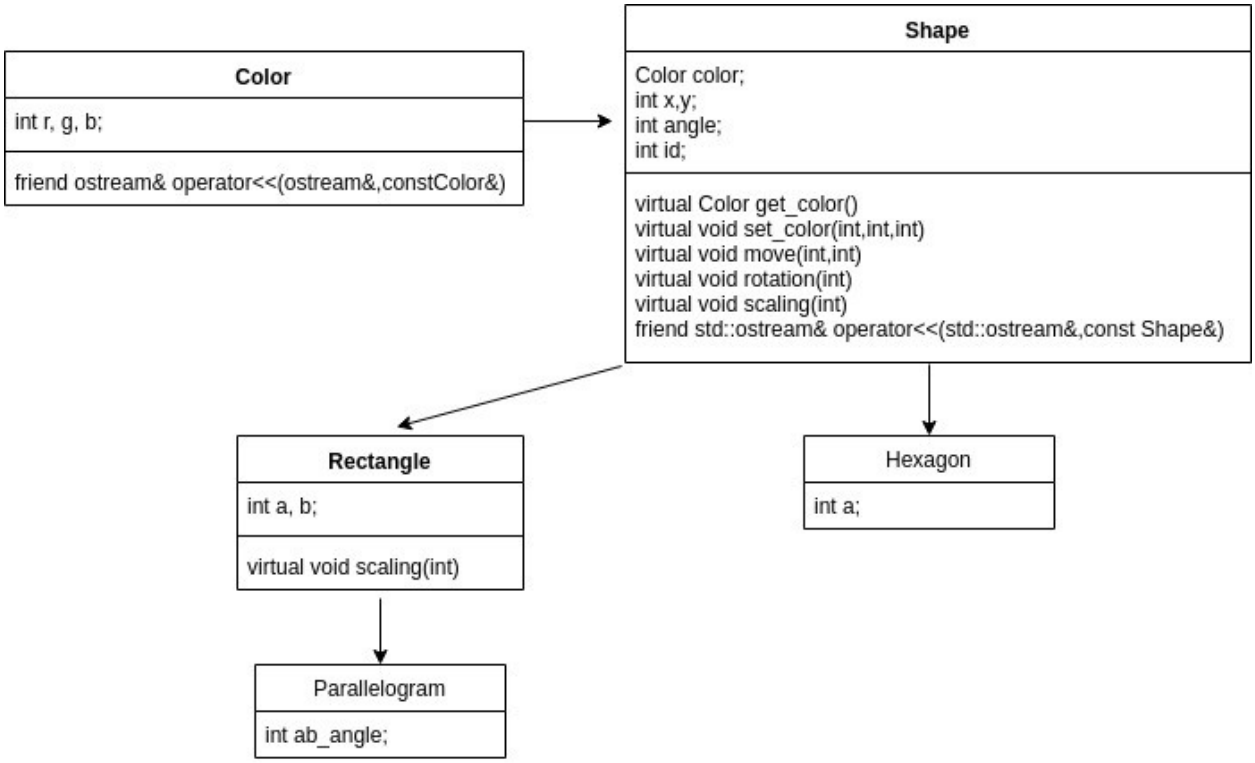
Индивидуальное задание:

Вариант 15.

1) Прямоугольник
2) Параллелограмм

3) Правильный шестиугольник

UML-диаграмма системы классов



Исходный код программы представлен в приложении А.

ПРИЛОЖЕНИЕ А

Исходный код программы

```
#include <iostream>
using namespace std;

static int global_id = 0;

struct Color{
    int r, g, b;
    friend ostream& operator << (ostream& output, const Color& color){
        output << color.r << ", " << color.g << ", " << color.b << endl;
        return output;
    }
};

class Shape
{
public:
    Shape(int x, int y): x(x), y(y){
            color.r = 0;
            color.g = 0;
            color.b = 0;
            angle = 0;
            id = global_id++;
    }
    virtual ~Shape();
    virtual Color get_color(){
        return color;
    }
    virtual void set_color(int r, int g, int b){
        color.r = r;
        color.r = g;
        color.r = b;
    }
    virtual void rotation(int angle){
        angle+= angle;
    }
    virtual void move(int dest_x, int dest_y){
        x = dest_x;
        y = dest_y;
    }
    friend std::ostream& operator<< (std::ostream& out, const Shape& shape){
            out <<"rectangle:\nID =  "<< shape.id<<
            "\ncoordinates = "<< shape.x <<
            "," << shape.y << "\nangle = "
            << shape.angle << "\ncolor(R,G,B) = " <<shape.color
            << "\n\n";
            return out;
        }
protected:
    Color color;
    int x, y;
    int angle;
    int id;
};

class Rectangle : public Shape
{
public:
    Rectangle(int x, int y, int a, int b): Shape (x,y), a(a), b(b){}
```

```
    virtual void scaling (int scale){
        a = a*scale;
        b = b*scale;
    }
    friend std::ostream& operator<< (std::ostream& out, const Rectangle&
rectangle){
            out <<"rectangle:\nID =  "<< rectangle.id<<
            "\ncoordinates = "<< rectangle.x <<
            "," << rectangle.y << "\nangle = "
            << rectangle.angle << "\ncolor(R,G,B) = " <<rectangle.color
            << "a = " << rectangle.a <<"\nb = "
            << rectangle.b
            << "\n\n";
            return out;
        }
protected:
    int a, b;
};

class Parallelogram : public Rectangle{
public:
    Parallelogram(int x, int y, int a, int b, int ab_angle): Rectangle
(x,y,a,b), ab_angle(ab_angle){}
    friend std::ostream& operator<< (std::ostream& out, const Parallelogram&
parallelogram){
            out <<"parallelogram:\nID =  "<< parallelogram.id<<
            "\ncoordinates = "<< parallelogram.x <<
            "," << parallelogram.y << "\nangle n = "
            << parallelogram.angle << "\ncolor(R,G,B) = "
<<parallelogram.color
            << "a = " << parallelogram.a <<"\nb = "
            << parallelogram.b << "/nab_angle = " << parallelogram.ab_angle
            << "\n\n";
            return out;
        }
protected:
    int ab_angle;
};

class Hexagon: public Shape{
public:
    Hexagon(int x, int y, int a): Shape (x, y), a(a){}
    void scaling (int scale){
        a = a*scale;
    }
    friend std::ostream& operator<< (std::ostream& out, const Hexagon&
hexagon){
            out <<"hexagon:\nID =  "<< hexagon.id<<
            "\ncoordinates = "<< hexagon.x <<
            "," << hexagon.y << "\nangle = "
            << hexagon.angle << "\ncolor(R,G,B) = " << hexagon.color
            << "a = " << hexagon.a
            << "\n\n";
            return out;
        }
protected:
    int a;
};
```