

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

Студентка гр. 7382

Дерябина П.С.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Изучить такие понятия, как полиморфизм, наследование, абстрактный класс, виртуальные функции и принцип их работы. В соответствии с индивидуальным заданием разработать систему классов для представления геометрических фигур.

Постановка 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:

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

Вариант №8: треугольник, пятиконечная звезда, прямоугольный треугольник.

Текстовое обоснование разработанных классов.

`Struct RGB` — структура для хранения значения цвета фигуры.

`Class Shape` — базовый абстрактный класс. Он содержит основные методы для геометрических фигур:

1. `virtual void scale(int num) = 0` — виртуальный метод масштабирования;
2. `void move(int x, int y)` — метод перемещения;

3. `void change_angle(int a)` — метод для поворота геометрической фигуры;
4. `void set_colors(int a, int b, int c)` — метод для задания цвета фигуры;
5. `RGB get_colors()` — метод для получения цвета фигуры.

`Class Triangle` — public-наследуемый от класса `Shape`. Содержит 3 дополнительных защищенных поля для хранения длины сторон треугольника и переопределенный метод масштабирования.

`Class RightTriangle` — public-наследуемый класс от класса `Triangle`. Не содержит дополнительных полей.

`Class Star5` — public-наследуемый класс от класса `Shape`. Содержит дополнительное поле длины стороны пятиконечной звезды и переопределение метода масштабирования.

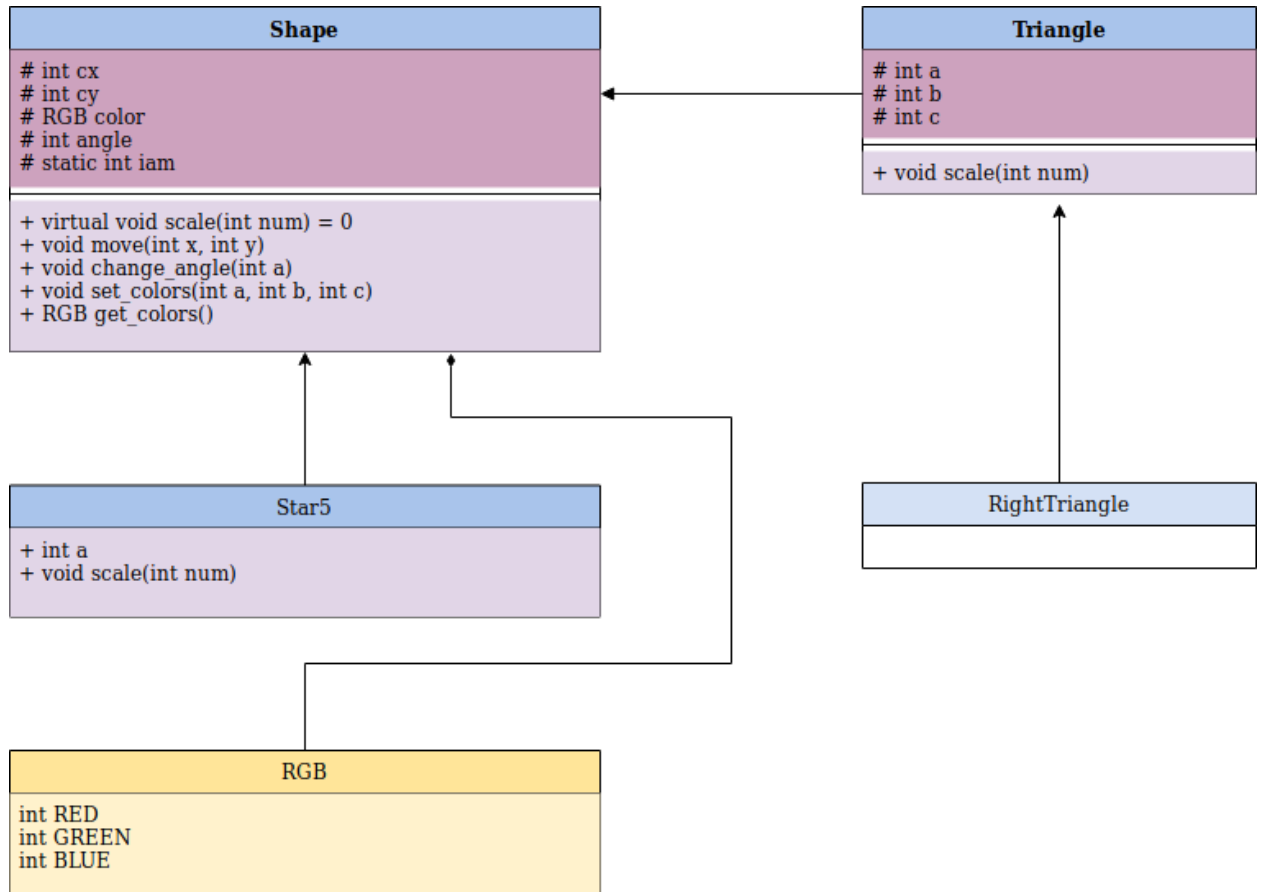
UML диаграмма разработанных классов приведена в приложении А, в приложении Б — исходный код программы.

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

UML диаграмма разработанных классов



ПРИЛОЖЕНИЕ Б

Исходный код программы

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

struct RGB
{
    int RED;
    int GREEN;
    int BLUE;
};

class Shape
{
protected:
    int cx;
    int cy;
    RGB color;
    int angle;
    static int iam;
public:
    Shape() : cx(0), cy(0), color({0,0,0}), angle(0)
    { iam++; }

    Shape(int x, int y, RGB c, int a) : cx(x), cy(y), color(c),
angle(a)
    { iam++; }

    virtual void scale(int num) = 0;

    void move(int x, int y)
    {
        this->cx = x;
        this->cy = y;
    }

    void change_angle(int a)
    {

        this->angle += a;

        if (this->angle > 360)
            this->angle %= 360;
    }
};
```

```

        if (this->angle < 0)
            this->angle += 360;

    }

    void set_colors(int a, int b, int c)
    {
        color.RED = a;
        color.GREEN = b;
        color.BLUE = c;
    }

    RGB get_colors()
    {
        return color;
    }

};

class Triangle : public Shape
{
protected:
    int a;
    int b;
    int c;

public:
    Triangle() : Shape(), a(0), b(0), c(0)
    {}

    Triangle(int cx, int cy, RGB colour, int angle, int a, int
b, int c) : Shape(cx, cy, colour, angle), a(a), b(b), c(c)
    {}

    void scale(int num)
    {
        if (num >= 0)
        {
            a *= num;
            b *= num;
            c *= num;
        }
        else
        {

```

```

        a /= abs(num);
        b /= abs(num);
        c /= abs(num);
    }

}

friend std::ostream& operator<< (std::ostream& out, const
Triangle &t)
{
    out << "Triangle: " << std::endl;
    out << "Center (" << t.cx << ", " << t.cy << ")" <<
std::endl;
    out << "Sides: " << t.a << ", " << t.b << ", " << t.c
<< std::endl;
    out << "Angle: " << t.angle << std::endl;
    out << "Colors (RGB): " << t.color.RED << ", " << t.-
color.GREEN << ", " << t.color.BLUE << std::endl;
    out << "ID: " << t.iam << std::endl;
}

};

class RightTriangle : public Triangle
{
public:
    RightTriangle() : Triangle()
    {}

    RightTriangle(int cx, int cy, RGB c, int angle, int a, int
b) : Triangle(cx, cy, c, angle, a, b, sqrt(pow(a, 2) + pow(b,
2)))
    {}

    friend std::ostream& operator<< (std::ostream& out,
const RightTriangle &t)
    {
        out << "Triangle: " << std::endl;
        out << "Center (" << t.cx << ", " << t.cy << ")"
<< std::endl;
        out << "Sides: " << t.a << ", " << t.b << ", "
<< t.c << std::endl;
        out << "Angle: " << t.angle << std::endl;
    }
};

```

```

        out << "Colors (RGB): " << t.color.RED << ", " << t.-
color.GREEN << ", " << t.color.BLUE << std::endl;
        out << "ID: " << t.iam << std:: endl;
    }
};

class Star5 : public Shape
{
public:
    int a;

    Star5() : Shape(), a(0)
    {}

    Star5(int cx, int cy, RGB c, int angle, int a) : Shape(cx,
cy, c, angle), a(a)
    {}

    void scale(int num)
    {
        if (num >= 0)
        {
            a *= num;
        }
        else
        {
            a /= abs(num);
        }
    }

    friend std::ostream& operator<< (std::ostream& out, const
Star5 &s)
    {
        out << "Five-pointed star:" << std::endl;
        out << "Center (" << s.cx << ", " << s.cy << ")" <<
std::endl;
        out << "Side: " << s.a << std::endl;
        out << "Angle: " << s.angle << std::endl;
        out << "Colors (RGB): " << s.color.RED << ", " << s.-
color.GREEN << ", " << s.color.BLUE << std::endl;
        out << "ID: " << s.iam << std:: endl;
    }
};

```



```

int Shape::iam = 0;

int main(){

    std::cout << "Triangle
    -----" << std::endl <<
std::endl;
    Triangle tr(0, 0, {0, 0, 255}, 45, 10, 20, 30);
    std::cout << tr << std::endl;

    std::cout << "moving... (10, 10)" << std::endl;
    tr.move(10, 10);

    std::cout << "scaling... (-10)" << std::endl;
    tr.scale(-10);

    std::cout << "rotating... (-20)" << std::endl << std::endl;
    tr.change_angle(-20);

    std::cout << tr << std::endl;
    std::cout << "RightTriangle
    -----" << std::endl <<
std::endl;

    RightTriangle rtr(100, 2, {255, 0, 255}, 90, 2, 2);
    std::cout << rtr << std::endl;

    std::cout << "moving... (-100, 2)" << std::endl;
    rtr.move(-100, 2);

    std::cout << "scaling... (-2)" << std::endl;
    rtr.scale(-2);

    std::cout << "rotating... (100)" << std::endl <<
std::endl;
    rtr.change_angle(-100);

    std::cout << rtr << std::endl;
    std::cout << "Five-pointed Star
    -----" << std::endl <<
std::endl;

    Star5 st(5, 5, {255, 255, 255}, 0, 100);
    std::cout << st << std::endl;

    std::cout << "moving... (5, 6)" << std::endl;

```

```
    st.move(5, 6);

    std::cout << "scaling... (50)" << std::endl;
    st.scale(50);

    std::cout << "rotating... (100)" << std::endl <<
std::endl;
    st.change_angle(100);

    std::cout << st << std::endl;

}3
```