

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МОЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №2**  
**по дисциплине «Построение и анализ алгоритмов»**  
**Тема: Наследование**

Студент гр. 7304

\_\_\_\_\_

Нгуен К.Х.

Преподаватель

\_\_\_\_\_

Размочаева Н.В.

Санкт-Петербург

2019

## Цель работы.

Изучить механизм наследования в языке C++ и научиться проектировать 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.

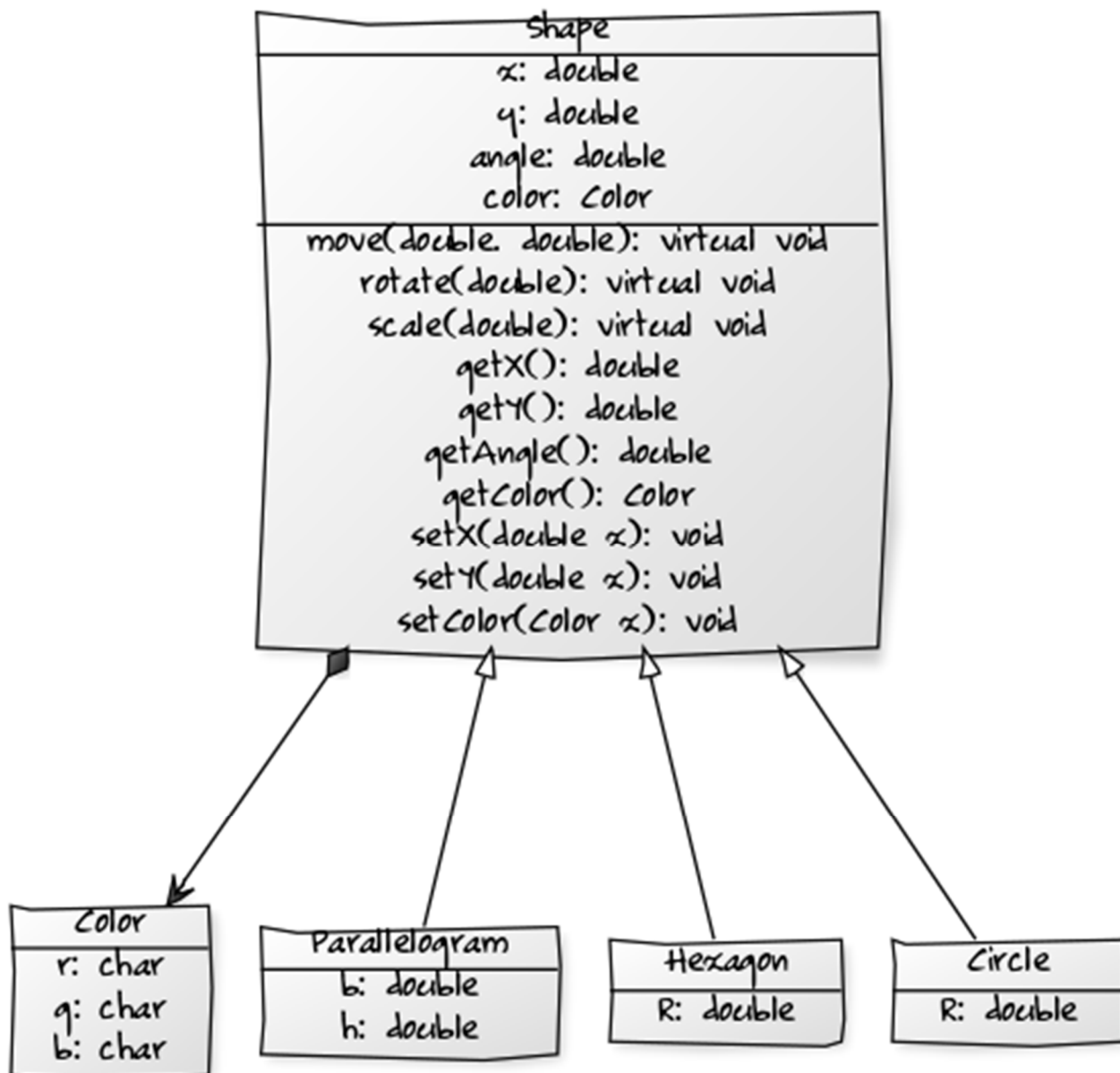
Необходимо также обеспечить однозначную идентификацию каждого объекта.

Решение должно содержать:

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

Вариант 19: Круг + Параллелограмм + Правильный шестиугольник

## Ход работы



1. Поскольку любая фигура будет иметь свои координаты, цвет и угол поворота (например, угол между направлением вверх фигуры и направлением вверх системы координат), в классе **Shape** 4 создаются поля: `x`, `y`, `color` и `angle`. Затем я добавил соответствующий метод получения и установки для них. Наконец, добавлена функция перемещения, поворота, масштабирования. Масштаб функции объявляется как чисто виртуальная функция, потому что она действует по-разному для каждой фигуры; например, для круга необходимо увеличить радиус, а для параллелограмма - изменить высоту и основание. Также добавлена функция `toString` для использования в операторе `<<`. В каждой фигуре мы можем объявить, как они могут распечатать свою информацию независимо.

2. Создал 3 дочерних класса из класса Shape: Circle, Hexagon и Parallelogram и реализовал / переопределил виртуальные функции.
3. Создан класс Circle. Круг определяется его радиусом, поэтому я добавил поле R. Масштабирование изменило бы значение R.
4. Создан класс Hexagon. Шестиугольник определен с его внешним радиусом, поэтому я добавил поле R. Масштабирование изменило бы значение R.
5. Создан класс параллелограмма. Параллелограмм определяется его основанием, высотой и углом между его стороной и верхней базой (назовем это альфа), поэтому я добавил 3 поля: b (основание), h (высота) и альфа. Масштабирование изменит значение как b, так и h.

## Экспериментальные результаты.

Реализованные классы в C ++

```
// LR2.cpp : Defines the entry point for the console application.
//
// Variant 19
#include "stdafx.h"
#include <iostream>
#include <sstream>
class Color {
public:
    unsigned char r, g, b;
    Color(unsigned char vr, unsigned char vg, unsigned char vb) :r(vr), g(vg), b(vb) {}
    friend std::ostream& operator<< (std::ostream& stream, const Color& color) {
        return stream << "Color(" << static_cast<unsigned>(color.r) << "," <<
static_cast<unsigned>(color.g) << "," << static_cast<unsigned>(color.b) << ")";
    }
};
class Shape {
protected:
    double x;
    double y;
    double angle;
    Color color;

    Shape(double vx, double vy, double vangle) : x(vx), y(vy), angle(vangle), color(0, 0,
0) {
    };
    Shape(double vx, double vy, double vangle, Color& c) : x(vx), y(vy), angle(vangle),
color(c) {
    };
public:
    virtual void move(double x, double y) {
        this->x = x;
        this->y = y;
    }
    virtual void rotate(double angle) {
        this->angle += angle;
    }

    virtual void scale(double sc) = 0;
```

```

    double getX() const { return this->x; }
    double getY() const { return this->y; }
    double getAngle() const { return this->angle; }
    Color getColor() const {
        return this->color;
    }

    void setX(double x) { this->x = x; }
    void setY(double y) { this->y = y; }

    void setColor(Color& c) {
        color = c;
    }

    virtual std::string toString() const = 0;

    friend std::ostream& operator<< (std::ostream& stream, const Shape& shape) {
        return stream << shape.toString();
    }
};

class Circle : public Shape {
protected:
    double R;
public:
    Circle(double vx, double vy, double vangle, double vR)
        : Shape(vx, vy, vangle), R(vR) {
    }
    Circle(double vx, double vy, double vangle, double vR, Color& c)
        : Shape(vx, vy, vangle, c), R(vR) {
    }
    virtual void scale(double sc) override{
        this->R *= sc;
    }

    virtual std::string toString() const override{
        std::stringstream ss;
        ss << "Circle: x = " << x << "; y = " << y << "; angle = " << angle << "; color
= " << color << "; R = " << R << std::endl;
        return ss.str();
    }
};

class Hexagon :public Shape {
protected:
    double R;
public:
    Hexagon(double vx, double vy, double vangle, double vR)
        : Shape(vx, vy, vangle), R(vR) {
    }
    Hexagon(double vx, double vy, double vangle, double vR, Color& c)
        : Shape(vx, vy, vangle, c), R(vR) {
    }
    virtual void scale(double sc) override {
        this->R *= sc;
    }

    virtual std::string toString() const override {
        std::stringstream ss;
        ss << "Hexagon: x = " << x << "; y = " << y << "; angle = " << angle << ";
color = " << color << "; R = " << R << std::endl;
        return ss.str();
    }
};

class Parallelogram :public Shape {
protected:

```

```

        double b;
        double h;
        double alpha;
public:
    Parallelogram(double vx, double vy, double vangle, double vb, double vh, double
valpha)
        : Shape(vx, vy, vangle), b(vb), h(vh), alpha(valpha) {
    }
    Parallelogram(double vx, double vy, double vangle, double vb, double vh, double
valpha, Color& c)
        : Shape(vx, vy, vangle, c), b(vb), h(vh), alpha(valpha) {
    }
    virtual void scale(double sc) override {
        b *= sc;
        h *= sc;
    }

    virtual std::string toString() const override {
        std::stringstream ss;
        ss << "Parallelogram: x = " << x << " "; y = " << y << " "; angle = " << angle <<
"; color = " << color << " "; Side = " << b << " "; Height = " << h << " "; Alpha = " << alpha <<
std::endl;
        return ss.str();
    }
};

int main()
{
    Color c_red(255, 0, 0);
    Circle c1(1, 2, 0, 3);
    Circle c2(1, 2, 0, 3, c_red);
    std::cout << "C1: " << c1 << "C2: " << c2;
    c1.move(2, 3);
    std::cout << "C1 after move(): " << c1;
    c1.rotate(10);
    std::cout << "C1 after rotate(): " << c1;
    c1.scale(3);
    std::cout << "C1 after scale(3): " << c1;
    c1.setColor(Color(100, 100, 100));
    std::cout << "C1 after change color: " << c1;

    Hexagon h1(1, 2, 0, 3);
    Hexagon h2(1, 2, 0, 3, Color(255, 0, 0));

    std::cout << h1 << h2;

    Parallelogram p1(1, 2, 0, 3, 2, 45);
    Parallelogram p2(1, 2, 0, 3, 2, 45, Color(1, 0, 0));

    std::cout << p1 << p2;

    getchar();
    return 0;
}

```

## Выводы.

В результате лабораторной работы я научился анализировать контекст и разрабатывать систему классов, подходящих для упражнения. Изучал, как

реализовать классы, абстрактные функции, наследование классов на языке C ++  
и практиковался в реализации разработанной системы классов на языке C ++.