

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. 7304

Давыдов А.А.

Преподаватель

Размочаева Н.В.

Санкт-Петербург

2019

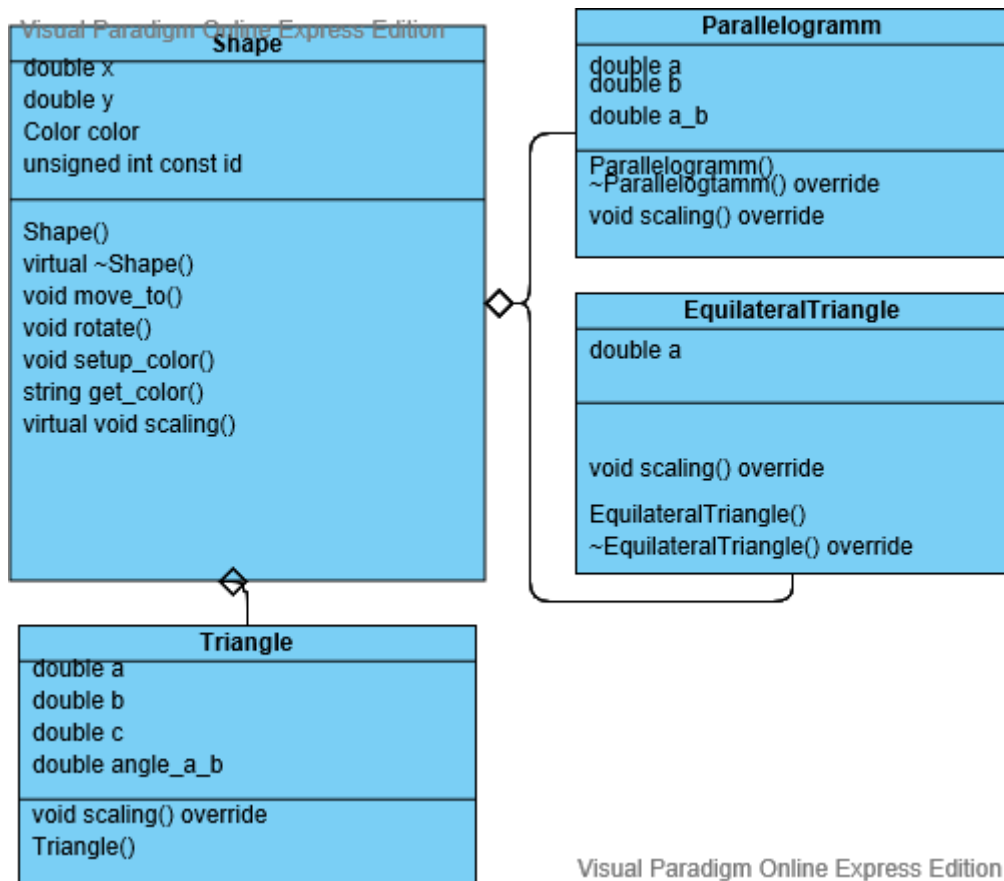
Цель работы

Задача

- 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++.

Экспериментальные результаты.

- 1) Была построена UML диаграмма разработанных классов



2) Был написано текстовое обоснование проектных решений

- В класс Shape были помещены поля `x`, `y`, `color`, `angle` (угол с осью `ox`), потому что эти параметры имеет любая фигура, которая будет наследоваться от Shape. Были реализованы методы `move_to()`, `rotate()`, `setup_color()`, `get_color()`, потому что их не требуется изменять в классах наследниках. Метод `scaling()` был объявлен как виртуальный, потому что его необходимо переопределять в зависимости от фигуры. Деструктор класса Shape объявлен как виртуальный, потому что класс абстрактный и поэтому необходимо обеспечить корректный вызов деструкторов в случае использования динамического полиморфизма. Конструктор класса Shape() необходим для инициализации приватных полей, которые будут содержать наследники. Оператор вывода на консоль также реализован в классе Shape() для передачи туда объектов наследников с помощью `dynamic_cast` и вывода общей для них информации из атрибутов класса Shape.
- В класс EquilateralTriangle - наследника Shape, был добавлен атрибут `a` - сторона треугольника и переопределен метод `scaling()` и реализован соответствующий оператор вывода.
- Класс Triangle был отнаследован от Shape. Были добавлены атрибуты `a`, `b` - две стороны и угол между сторонами `a` и `b` - атрибут `angle_a_b`, сторона `c` тоже является атрибутом и вычисляется по теореме косинусов.

В классе также переопределен метод `scaling()` и реализован соответствующий оператор вывода.

- Класс `Parallelogramm` был отнаследован от `Shape`, в него были помещены атрибуты `a`, `b` - прилежащие стороны и `angle_a_b` - угол между ними. В классе был переопределен метод `scaling()`, реализован соответствующий оператор вывода.

3) Была сделана реализация классов на языке C++

```
//abstract class for next inheritance
class Shape
{
public:
    Shape(double x, double y, double angle, Color color) : x(x), y(y),
color(color), id(next_id())
    {
        if(angle >= 360.0)
            this->angle = angle - int(angle / 360) * 360;
        else
            this->angle = angle;
    }

    virtual ~Shape()
    {
        cout << "~Shape()" << endl;
    }

    //common methods
    void move_to(double x, double y)
    {
        this->x = x;
        this->y = y;
    }

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

    void setup_color(Color const &c)
    {
        color = c;
    }

    string get_color() const
    {
        switch(color)
        {
            case Color::RED:
                return "Color: RED";

            case Color::ORANGE:
                return "Color: ORANGE";

            case Color::YELLOW:
                return "Color: YELLOW";
        }
    }
};
```

```

        case Color::GREEN:
            return "Color: GREEN";

        case Color::BLUE:
            return "Color: BLUE";

        case Color::DARK_BLUE:
            return "Color: DARK BLUE";

        case Color::VIOLET:
            return "Color: VIOLET";

        default:
            return "Unknown color";
    }
}

friend std::ostream & operator <<(std::ostream &out, Shape &shape)
{
    out << "Object id: " << shape.id << endl << "(x, y): " << shape.x <<
    ", " << shape.y << endl << "Angle with ox: " << shape.angle << " degrees" <<
    endl << shape.get_color();
    return out;
}

//abstract methods
virtual void scaling(double k) = 0;

private:
    double x, y;
    double angle;
    Color color;
    unsigned int const id;
};

class EquilateralTriangle : public Shape
{
public:
    EquilateralTriangle(double x, double y, double angle, Color color, double
a) : Shape(x, y, angle, color), a(a)
    {}
    ~EquilateralTriangle() override
    {
        cout << "~EquilateralTriangle()" << endl;
    }

    void scaling(double k) override
    {
        a *= k;
    }

    friend std::ostream & operator << (std::ostream & out,
EquilateralTriangle &eq_triangle)
    {
        out << dynamic_cast<Shape &>(eq_triangle) << endl << "Side a: " <<
eq_triangle.a;
        return out;
    }
private:
    //side of triangle
    double a;
};

```

```

class Triangle : public Shape
{
public:
    Triangle(double x, double y, double angle, Color color, double a, double
b, double angle_a_b) : Shape(a, b, angle, color), a(a), b(b)
    {
        if(angle_a_b >= 360)
            this->angle_a_b = angle_a_b - int(angle_a_b / 360) * 360;
        else
            this->angle_a_b = angle_a_b;
        c = sqrt(a*a + b*b - 2*a*b*cos(angle_a_b * M_PI / 180));
    }

    ~Triangle() override
    {
        cout << "~Triangle()" << endl;
    }

    void scaling(double k) override
    {
        a *= k;
        b *= k;
        c *= k;
    }

    friend std::ostream & operator << (std::ostream & out, Triangle
&triangle)
    {
        out << dynamic_cast<Shape> &(triangle) << endl << "Side a: " <<
triangle.a << endl << "Side b: " << triangle.b << endl << "Side c:" <<
triangle.c;
        return out;
    }
private:
    double a;
    double b;
    double angle_a_b;
    //this side compute by theorem of cos
    double c;
};

class Parallelogram : public Shape
{
public:
    Parallelogram(double x = 1, double y = 1, double angle = 1, Color color =
Color::RED, double a = 1, double b = 1, double angle_a_b = 1) : Shape(x, y,
angle, color), a(a), b(b), angle_a_b(angle_a_b)
    {}

    ~Parallelogram() override
    {
        cout << "~Parallelogram()" << endl;
    }

    void scaling(double k) override
    {
        a *= k;
        b *= k;
    }
}

```

```

        friend std::ostream & operator << (std::ostream & out, Parallelogram &p)
        {
            out << dynamic_cast<Shape &>(p) << endl << "Side a: " << p.a << endl
            << "Side b: " << p.b << endl << "Angle between a and b: " << p.angle_a_b;
            return out;
        }
    private:
        double a;
        double b;
        //angle between sides a and b
        double angle_a_b;
};

```

Выводы.

Была реализована иерархия классов, описывающих поведение фигур. За базовый абстрактный класс был взят класс Shape, в котором были реализованы методы перемещения, установки и получения цвета, оператор вывода. Метод масштабирования был объявлен как виртуальный, потому что его необходимо переопределять в наследниках.