

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.

Студент гр.7304

Сергеев И.Д.

Преподаватель

Размочаева Н.В.

Санкт-Петербург

2019

1. Постановка задачи

1.1. Цель работы

Исследование способов наследования классов. Изучение их реализации. Реализация классов геометрических фигур.

1.2. Формулировка 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++.

Для разработки диаграммы классов UML необходимо использовать какой-либо онлайн редактор, например <https://yuml.me/>

1.3. Основные теоретические положения

Что такое наследование

Это принцип создания класса на базе уже существующего, при этом у нас есть возможность пользоваться функционалом (свойствами и методами) базового. Классы созданные таким образом называются производными или дочерними, а на базе которого создаются — родителем или базовым.

Этот механизм в объектно ориентированном программировании очень сильная фишка. Она в несколько раз экономит время на создание проекта, а также не нагружает его повторяющимся кодом.

Производный класс мы можем усовершенствовать, добавляя:

- Новые переменные.
- Функции.
- Конструкторы.

И все это не изменяя базовый класс.

2. Ход работы

2.1. Были реализованы следующие классы на языке c++:

2.1.1. Regular_hexogram

2.1.2. Rectangle

2.1.3. Parallelogram

2.2. Код программы:

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

using namespace std;

string first_fig = "Rectangle";
string second_fig = "Parallelogram";
string third_fig = "Regular_hexagon";

enum Color{RED, ORANGE, YELLOW, GREEN, BLUE, DARK_BLUE, VIOLET};

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

    ~Shape()
    {}

    void move(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 col(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 << "(x, y): " << shape.x << ", " << shape.y << endl << "Angle with ox: " << shape.angle << " degrees" <<
endl << shape.get_color();
    return out;
}

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

private:
    double x, y;
    double angle;
    Color color;
};

class Regular_hexagon : public Shape
{
public:
    Regular_hexagon(double x, double y, double angle, Color color, double a) : Shape(x, y, angle, color), a(a),
name(third_fig)
    {}

    ~Regular_hexagon()
    {}
}

```

```

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

friend std::ostream & operator << (std::ostream & out, Regular_hexagon &hex)
{
    out << dynamic_cast<Shape &>(hex) << endl << "Object name: " << hex.name << endl << "Side a: " <<
hex.a;
    return out;
}
private:
    double a; //side of hexagon
    string name;
};

```

```

class Rectangle : public Shape
{
public:
    Rectangle(double x, double y, double angle, Color color, double a, double b) : Shape(x, y, angle, color), a(a), b(b),
name(first_fig)
    {}

    ~Rectangle()
    {}

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

    friend std::ostream & operator << (std::ostream & out, Rectangle &rec)
    {
        out << dynamic_cast<Shape &>(rec) << endl << "Object name: " << rec.name << endl << "Side a: " << rec.a
<< endl << "Side b:" << rec.b;
        return out;
    }
private:
    double a; //side a
    double b; //side b
    string name;
};

```

```

class Parallelogram : public Shape
{
public:
    Parallelogram(double x, double y, double angle, Color color, double a, double b, double sides_angle) : Shape(x, y,
angle, color), a(a), b(b), sides_angle(sides_angle), name(second_fig)
    {}

    ~Parallelogram()
    {}

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

    friend std::ostream & operator << (std::ostream & out, Parallelogram &par)

```

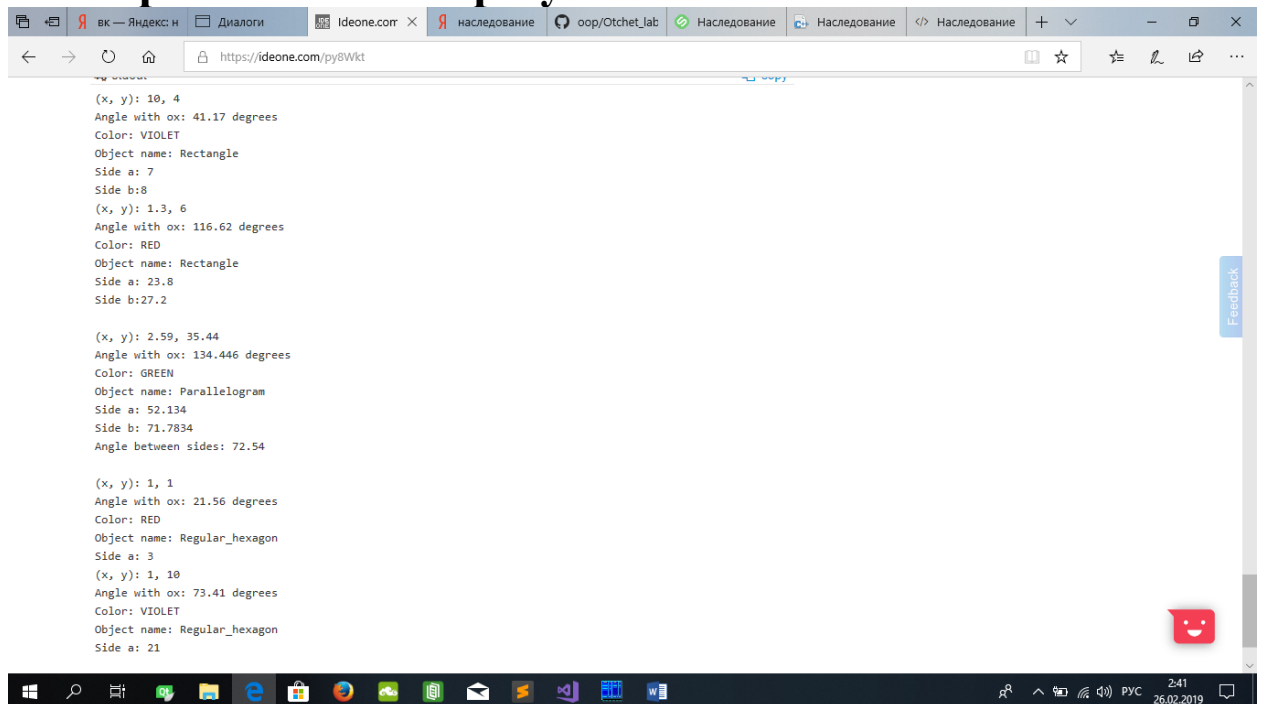
```

    {
        out << dynamic_cast<Shape &>(par) << endl << "Object name: " << par.name << endl << "Side a: " << par.a
        << endl << "Side b: " << par.b << endl << "Angle between sides: " << par.sides_angle;
        return out;
    }
private:
    double a;
    double b;
    double sides_angle; //angle between sides a and b
    string name;
};

int main()
{
    Rectangle *shape = new Rectangle(10, 4, 41.17, Color::VIOLET, 7, 8);
    cout << *shape << endl;
    shape->scaling(3.4);
    shape->move(1.3, 6);
    shape->rotate(75.45);
    shape->col(Color::RED);
    cout << *shape << endl << endl;
    Parallelogram *p = new Parallelogram(2.59, 35.44, 134.4464, Color::GREEN, 52.134, 71.7834, 72.54);
    cout << *p << endl << endl;
    Regular_hexagon *hex = new Regular_hexagon(1, 1, 21.56, Color::RED, 3);
    cout << *hex << endl;
    hex->scaling(7);
    hex->move(1, 10);
    hex->rotate(771.85);
    hex->col(Color::VIOLET);
    cout << *hex << endl;
    return 0;
}

```

3. Экспериментальные результаты



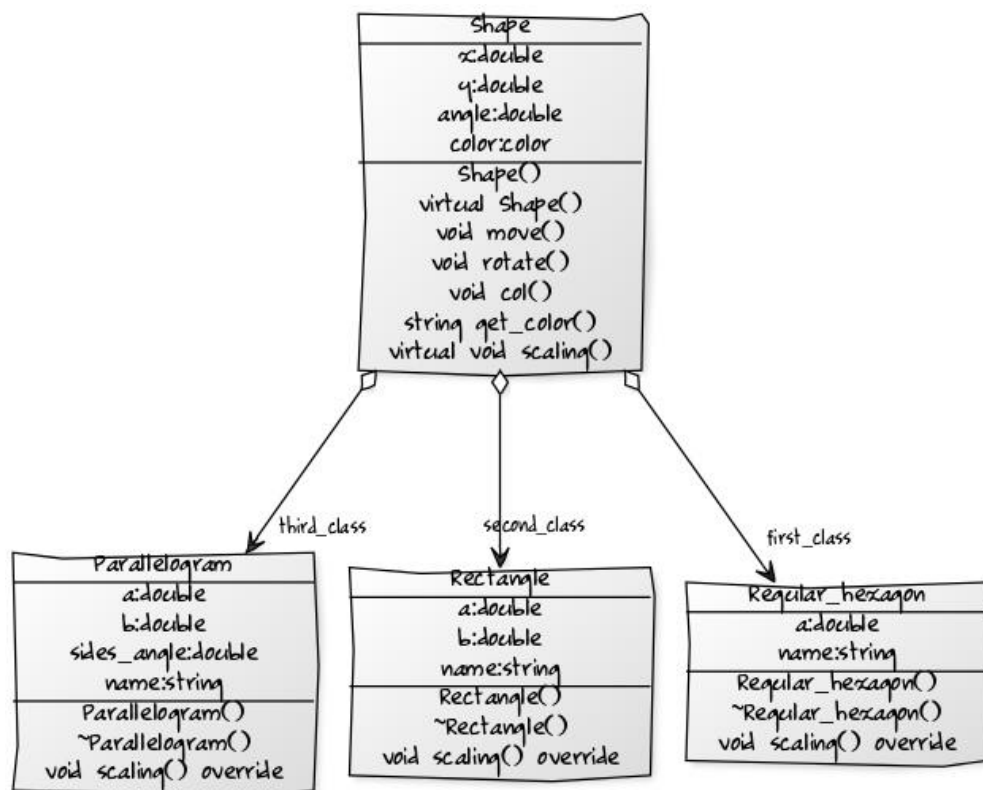
```

(x, y): 10, 4
Angle with ox: 41.17 degrees
Color: VIOLET
Object name: Rectangle
Side a: 7
Side b: 8
(x, y): 1.3, 6
Angle with ox: 116.62 degrees
Color: RED
Object name: Rectangle
Side a: 23.8
Side b: 27.2

(x, y): 2.59, 35.44
Angle with ox: 134.446 degrees
Color: GREEN
Object name: Parallelogram
Side a: 52.134
Side b: 71.7834
Angle between sides: 72.54

(x, y): 1, 1
Angle with ox: 21.56 degrees
Color: RED
Object name: Regular_hexagon
Side a: 3
(x, y): 1, 10
Angle with ox: 73.41 degrees
Color: VIOLET
Object name: Regular_hexagon
Side a: 21

```



4. Вывод

В результате работы были изучены способы наследования классов. Была написана программа, содержащая классы геометрических фигур и uml-диаграмма.