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. 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

Наследование - это принцип создание 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 фича. 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

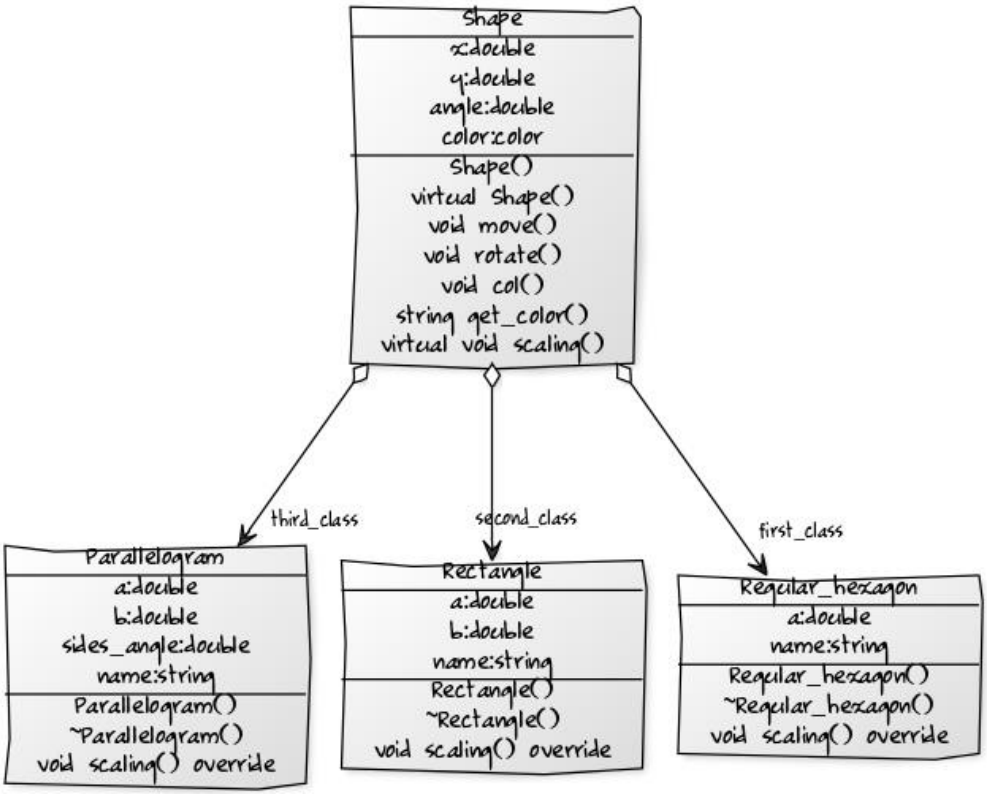
3. Экспериментальные результаты

x1:y1 17.097 : 6.42532
x2:y2 10.4731 : 4.16169
x3:y3 2.90297 : 1.57468
x4:y4 9.52686 : 3.83831
Object id: 1
(x, y): 10, 4
Angle with ox: 41.17 degrees
Color: VIOLET
Side a: 7
Side b:8
x1:y1 24.9708 : 15.4838
x2:y2 2.87805 : 6.63226
x3:y3 -22.3708 : -3.48385
x4:y4 -0.278053 : 5.36774
Object id: 1
(x, y): 1.3, 6
Angle with ox: 116.62 degrees
Color: RED
Side a: 23.8
Side b:27.2

x1:y1 11.2907 : 18.3103
x2:y2 51.377 : 9.58108
x3:y3 -6.11065 : 52.5697
x4:y4 -46.197 : 61.2989
Object id: 2
(x, y): 2.59, 35.44
Angle with ox: 134.446 degrees
Color: GREEN
Side a: 52.134
Side b: 71.7834
Angle between sides: 72.54

x1:y1 3.72546 : -0.253744
x2:y2 2.36273 : 2.08577
x3:y3 -0.36273 : 2.08577

x4:y4 -1.72546 : 2.25374
x5:y5 -0.36273 : -0.0857738
x6:y6 2.36273 : -0.0857738
Object id: 3
(x, y): 1, 1
Angle with ox: 21.56 degrees
Color: RED
Side a: 3
x1:y1 9.51361 : 29.1968
x2:y2 5.25681 : -6.62495
x3:y3 -3.25681 : -6.62495
x4:y4 -7.51361 : -9.19683
x5:y5 -3.25681 : 26.6249
x6:y6 5.25681 : 26.6249
Object id: 3
(x, y): 1, 10
Angle with ox: 73.41 degrees
Color: VIOLET
Side a: 21



4. Вывод

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

Приложение:

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

```
#include <string>

using namespace std;


unsigned int our_id(){
   static unsigned int id = 0;
   ++id;
   return id;
}

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), id(our_id())
   {
      if(angle >= 360.0)
         this->angle = angle - int(angle / 360) * 360;
      else
         this->angle = angle;
   }

   virtual ~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 << "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 method
    virtual void scaling(double k) = 0;

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


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)
    {}

    virtual ~Regular_hexagon()
    {
    }

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

    friend std::ostream & operator << (std::ostream & out, Regular_hexagon &hex)
    {
        out << "x1:y1 " << hex.x-hex.a*cos(hex.angle) << " : " << hex.y-hex.a*sin(hex.angle) << endl;
        out << "x2:y2 " << hex.x-hex.a/2*cos(hex.angle) << " : " << hex.y+hex.a*sqrt(3)/2*sin(hex.angle)
<< endl;
        out << "x3:y3 " << hex.x+hex.a/2*cos(hex.angle) << " : " << hex.y+hex.a*sqrt(3)/2*sin(hex.angle)
<< endl;
        out << "x4:y4 " << hex.x+hex.a*cos(hex.angle) << " : " << hex.y+hex.a*sin(hex.angle) << endl;
        out << "x5:y5 " << hex.x+hex.a/2*cos(hex.angle) << " : " << hex.y-hex.a*sqrt(3)/2*sin(hex.angle)
<< endl;
        out << "x6:y6 " << hex.x-hex.a/2*cos(hex.angle) << " : " << hex.y-hex.a*sqrt(3)/2*sin(hex.angle) <<
endl;
        out << dynamic_cast<Shape &>(hex) << endl << "Side a: " << hex.a;
```

```
        return out;
    }
private:
    double a; //side of hexagon
};


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)
    {}

    virtual ~Rectangle()
    {
    }

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

    friend std::ostream & operator << (std::ostream & out, Rectangle &rec)
    {
        out << "x1:y1 " << rec.x-(rec.a+rec.b)/2*cos(rec.angle) << " : " << rec.y-(rec.a+rec.b)/2*sin(rec.angle) << endl;
        out << "x2:y2 " << rec.x+(rec.a-rec.b)/2*cos(rec.angle) << " : " << rec.y+(rec.a-rec.b)/2*sin(rec.angle) << endl;
        out << "x3:y3 " << rec.x+(rec.a+rec.b)/2*cos(rec.angle) << " : " << rec.y+(rec.a+rec.b)/2*sin(rec.angle) << endl;
        out << "x4:y4 " << rec.x-(rec.a-rec.b)/2*cos(rec.angle) << " : " << rec.y-(rec.a-rec.b)/2*sin(rec.angle) << endl;
        out << dynamic_cast<Shape &>(rec) << endl << "Side a: " << rec.a << endl <<  "Side b:" << rec.b;
        return out;
    }
private:
    double a; //side a
    double b; //side b
};


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)
    {}

    virtual ~Parallelogram()
    {
    }

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

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

```
        out <<  "x1:y1 " << par.x-(par.a*cos(par.sides_angle)+par.b)/2*cos(par.angle) << " : " << par.y-(par.a*sin(par.sides_angle)+par.b)/2*sin(par.angle) << endl;
        out <<  "x2:y2 " << par.x+(par.a*cos(par.sides_angle)-par.b)/2*cos(par.angle) << " : " << par.y+(par.a*sin(par.sides_angle)-par.b)/2*sin(par.angle) << endl;
        out <<  "x3:y3 " << par.x+(par.a*cos(par.sides_angle)+par.b)/2*cos(par.angle) << " : " << par.y+(par.a*sin(par.sides_angle)+par.b)/2*sin(par.angle) << endl;
        out <<  "x4:y4 " << par.x-(par.a*cos(par.sides_angle)-par.b)/2*cos(par.angle) << " : " << par.y-(par.a*sin(par.sides_angle)-par.b)/2*sin(par.angle) << endl;
        out << dynamic_cast<Shape &>(par) << endl << "Side a: " << par.a << endl << "Side b: " << par.b << endl <<  "Angle between sides: " << par.sides_angle << endl;
        return out;
    }
private:
    double a;
    double b;
    double sides_angle; //angle betweend sides a and b
};

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;
    delete shape;
    return 0;
}
```