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. 7303 _____ Шаталов Э.В.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2019

Цель работы.

Ознакомиться с понятиями наследование, полиморфизм, абстрактный класс, изучить виртуальные функции, спроектировать систему классов для моделирования геометрических фигур (в соответствии с полученным индивидуальным заданием).

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++.

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

Индивидуальное задание.

Вариант 16 – реализовать систему классов для фигур:

1) Сектор круга;
2) Эллипс;
3) Пятиконечная звезда.

Обоснование проектных решений.

Для представления цвета написана структура *color* с полями *r, g, b* целочисленного типа char, которые отвечают за соотношение в цвете красного, зеленого и синего цветов соответственно.

Базовый класс для представления всех фигур — абстрактный класс *Shape*. В нем определены параметры, которые не зависят от формы фигуры: координаты центра (double *s_x,s_y*), угол поворота (double *rotation_angle*), цвет (color *s_color*) и идентификационный номер (int *id)*.

Для работы с этими параметрами были реализованы следующие методы:

- *void setcoordinates(double newX, double newY)*

  Метод отвечает за перемещение центра фигуры в точку с координатами (`x,y`).

- *virtual void rotate(double angle)*

  Метод отвечает за поворот фигуры на угол `angle`.

- *virtual void scale(double coefficient) = 0*

  Метод отвечает за масштабирование фигуры на коэффициент `size`. В связи с тем, что данных, которые необходимо менять для совершения этой операции, в этом классе нет, метод чисто виртуальный.

- *void setColor(const Color& newColor)*

  Метод отвечает за замену цвета фигуры на цвет *newColor*.

- *Color getColor()*

  Метод отвечает за получение текущего цвета фигуры.

Класс *Sector* является public наследником класса *Shape* и используется для представления сектора окружности. Он содержит в себе защищенные поля вещественного типа *s_angle* и *s_radius* для хранения угла сектора и радиуса. В этом классе переопределен метод масштабирования (void scale(double)), который увеличивает радиус сектора.

Класс *Star* также является public наследником класса *Shape* и используется для представления звезды. Он содержит в себе защищенные поля целого типа s_radius и s_radius2 для хранения внешнего и внутреннего радиуса. В этом классе переопределен метод масштабирования (void scale(double)), который увеличивает размеры радиусов в size раз.

Класс *ellipse* является *public* наследником класса *Shape*. Он содержит в себе защищенные поля целого типа *e_a* и *e_b* для хранения двух радиусов. В этом классе переопределен метод масштабирования (void scale(double)), который увеличивает размеры радиусов в size раз.

Для перегрузки оператора вывода фигуры в поток оператор ≪ объявлен во всех классах со спецификатором friend, это необходимо, чтобы было возможно обращаться к защищенным полям и выводить их значения.

UML диаграмма разработанных классов.

UML диаграмма разработанных классов представлена в приложении А.

Реализация классов на языке C++.

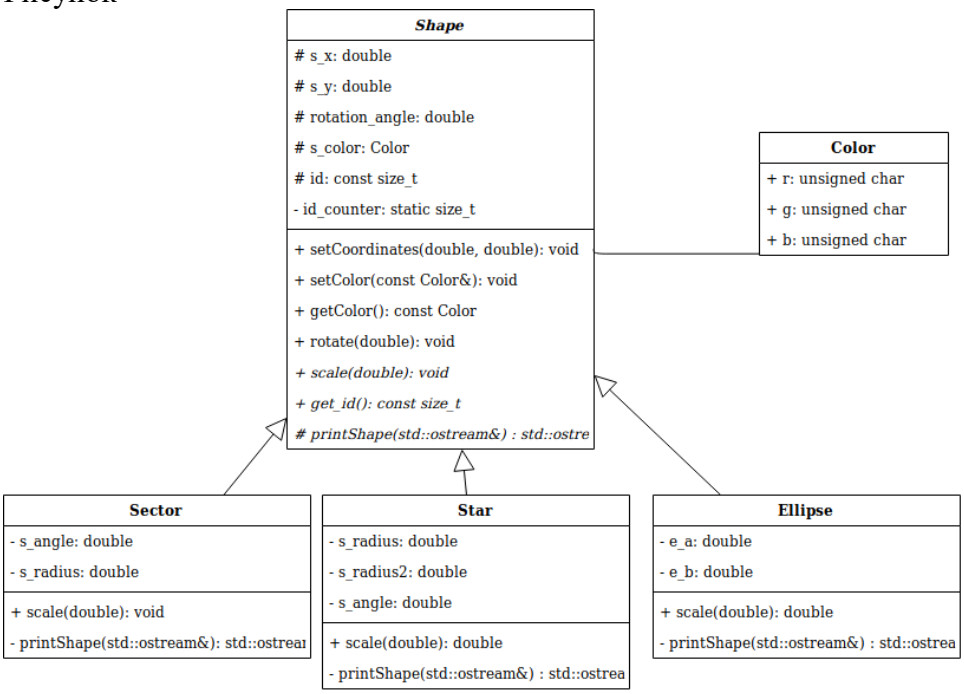
Реализация классов представлена в приложении Б.

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 абстрактным (класс называется абстрактным, если содержит хотя бы один чисто виртуальный метод). Были реализованы методы перемещения фигуры в заданные координаты, поворота на заданный угол, масштабирования на заданный коэффициент, установки и получения цвета, а также оператор вывода в поток.

ПРИЛОЖЕНИЕ А
UML ДИАГРАММА

Рисунок



1: UML диаграмма

ПРИЛОЖЕНИЕ Б
РЕАЛИЗАЦИЯ КЛАССОВ

shape.h:

```
     #include <iostream>
typedef struct Color {
    unsigned char r;
    unsigned char g;
    unsigned char b;

    Color(unsigned char red = 0, unsigned char green = 0, unsigned char
blue = 0) :
        r(red),
        g(green),
        b(blue)
    {
    }

    Color(const Color& item) :
        r(item.r),
        g(item.g),
        b(item.b)
    {
    }

    friend std::ostream& operator<<(std::ostream& stream, const Color&
color) {
        stream << '(' << int(color.r) << ',' << int(color.g) << ',' <<
int(color.b) << ')';
        return stream;
    }

} Color;

class Shape{
```

```
public:

    void setCoordinates(double newX, double newY);

    Shape(double x = 0.0, double y = 0.0, double r_angle = 0.0, const
Color& color = Color()) :
        s_x(x),
        s_y(y),
        id(id_counter++),
        s_color(color),
        rotation_angle(r_angle)
    {
    }

    virtual ~Shape() = default;

    void setColor(const Color& newColor);

    const Color getColor();

    virtual void rotate(double angle);

    virtual void scale(double) = 0;

    friend std::ostream& operator<<(std::ostream& stream, Shape& shape) {
        return shape.printShape(stream);
    }
    //virtual Shape& operator++(int) = 0;
    virtual Shape& operator++() = 0;
    int get_id();

protected:
    virtual std::ostream& printShape(std::ostream&) = 0;
    double s_x;
```

```
    double s_y;
    int id;
    Color s_color;
    double rotation_angle;
    static int id_counter;
};

class Sector : public Shape {
public:
    Sector(double x = 0, double y = 0, double r_angle = 0.0, Color color
= Color(), double radius = 1.0, double angle = 360.0) :
        Shape(x, y, r_angle, color),
        s_angle(angle),
        s_radius(radius)
    {
    }

    Sector(const Sector& item):
        Shape(item.s_x, item.s_y, item.rotation_angle, item.s_color),
        s_angle(item.s_angle),
        s_radius(item.s_radius)
    {
    }

    Sector& operator=(Sector& item) {
        s_angle = item.s_angle;
        s_color = item.s_color;
        rotation_angle = item.rotation_angle;
        s_radius = item.s_radius;
        s_x = item.s_x;
        s_y = item.s_y;
        return *this;
    }
```

```
    void scale(double coefficient) override;
    Sector& operator++(int) {
        Sector* a = new Sector(*this);
        ++(*this);
        a->id = this->id;
         id_counter--;
        return *a;

    }
    Sector& operator++(){
        ++s_radius;
        ++s_angle;
        return *this;
    }
    ~Sector() override = default;

private:
    std::ostream& printShape(std::ostream& stream = std::cout) override{
        stream << "Сектор круга с углом " << s_angle << " и радиусом " <<
s_radius << " с центром в координатах (" << this->s_x << ',' << this->s_y
<< ")" << " и повернут на угол " << this->rotation_angle << " градусов"
<< ". Цвет - " << this->s_color << " id - " << get_id() << std::endl;
        return stream;
    }

    double s_angle;
    double s_radius;
};


class Star:public Shape{
public:
    int dot1x,dot1y;
```

```
    Star(double x = 0, double y = 0, double r_angle = 0.0, Color color =
Color(), double radius = 1.0, double radius2 = 1.0) :
        Shape(x, y, r_angle, color),
        s_radius(radius),
        s_radius2(radius2)
        {
        dot1x = x+s_radius;
        dot1y = y;
    }

    Star(const Star &item):
        Shape(item.s_x, item.s_y, item.rotation_angle, item.s_color),
        s_radius(item.s_radius),
        s_radius2(item.s_radius2)
    {
    }

    Star& operator=(Star& item) {
        s_color = item.s_color;
        rotation_angle = item.rotation_angle;
        s_x = item.s_x;
        s_y = item.s_y;
        s_radius = item.s_radius;
        s_radius2 = item.s_radius2;
        return *this;
    }

    Star& operator++(int) {
        Star* a = new Star(*this);
        ++(*this);
        a->id = this->id;
        id_counter--;
        return *a;
    }
```

```
    Star& operator++() override {
        ++s_radius;
        ++s_radius2;
        return *this;
    }
    void scale(double coefficient) override;

    void rotate(double angle) override;

    ~Star() override = default;
private:
    std::ostream& printShape(std::ostream& stream = std::cout) override{
        stream << "Звезда с внешним радиусом " << s_radius << " и
внутренним "<< s_radius2 << " с центром в координатах (" << this->s_x <<
',' << this->s_y << ")" << " и повернута на угол " <<
this->rotation_angle << " градусов" << " координаты 1 пика: ("<<dot1x
<<", " << dot1y << "). Цвет - " << this->s_color << " id - " << get_id()
<< std::endl;
        return stream;
    }
    double s_radius;
    double s_radius2;

};

class Ellipse : public Shape {
public:
    Ellipse(double x = 0, double y = 0, double r_angle = 0.0, const
Color& color = Color(), double a = 1.0, double b = 1.0) :
        Shape(x, y, r_angle, color),
        e_a(a),
        e_b(b)
    {
```

```
}

Ellipse(const Ellipse& item) :
    Shape(item.s_x, item.s_y, item.rotation_angle, item.s_color),
    e_a(item.e_a),
    e_b(item.e_b)
{
    rotation_angle = item.rotation_angle;
}

Ellipse& operator=(Ellipse& item) {
    e_a = item.e_a;
    e_b = item.e_b;
    rotation_angle = item.rotation_angle;
    s_x = item.s_x;
    s_y = item.s_y;
    return *this;
}
Ellipse& operator++(int) {
    Ellipse* a = new Ellipse(*this);
    ++(*this);
    a->id = this->id;
    id_counter--;
    return *a;
}

Ellipse& operator++(){
    ++e_a;
    ++e_b;
    return *this;
}
void scale(double coefficient) override;

~Ellipse() override = default;
```

```
private:
    std::ostream& printShape(std::ostream& stream = std::cout) override {
        stream << "Эллипс с параметрами " << e_a << " , " << e_b << " и
центром в координатах (" << this->s_x << "," << this->s_y << ")" << "
повернут на угол " << this->rotation_angle << " градусов" << ". Цвет - "
<< this->s_color << " id - " << get_id() << std::endl;
        return stream;
    }

    double e_a;
    double e_b;
};
shape.cpp:
#include <shape.h>
#include <math.h>
int Shape::id_counter = 0;
void Shape::setCoordinates(double newX, double newY) {
    s_x = newX;
    s_y = newY;
}
;
void Shape::setColor(const Color& newColor)
{
    s_color = newColor;
}

const Color Shape::getColor(){
    return s_color;
}

void Shape::rotate(double angle) {
    rotation_angle += angle;
}
```

```
int Shape::get_id(){
    return id;
}

void Sector::scale(double coefficient){
    s_radius *= coefficient;
}


void Star::scale(double coefficient){
    s_radius *=coefficient;
    s_radius2 *=coefficient;
}


void Star::rotate(double angle){
   rotation_angle += angle;
   dot1x =this->s_x + cos(rotation_angle/180*M_PI)*s_radius;
   dot1y =this->s_y - sin(rotation_angle/180*M_PI)*s_radius;
}

void Ellipse::scale(double coefficient) {
    e_a *= coefficient;
    e_b *= coefficient;}
```