МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Объектно-ориентированное программирование»

Тема: Наследование

Студент гр. 7303 _____ Шаломов А.Д.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2019

Цель работы.

Ознакомление с понятиями объектно-ориентированного программирования, такими как полиморфизм, наследование, виртуальные функции, принципы их работы, способ организации в памяти в языке C++.

Постановка задачи.

Необходимо спроектировать систему классов для моделирования геометрических фигур, состоящую из прямоугольника, параллелограмма, правильного шестиугольника. 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.

Выполнение работы.

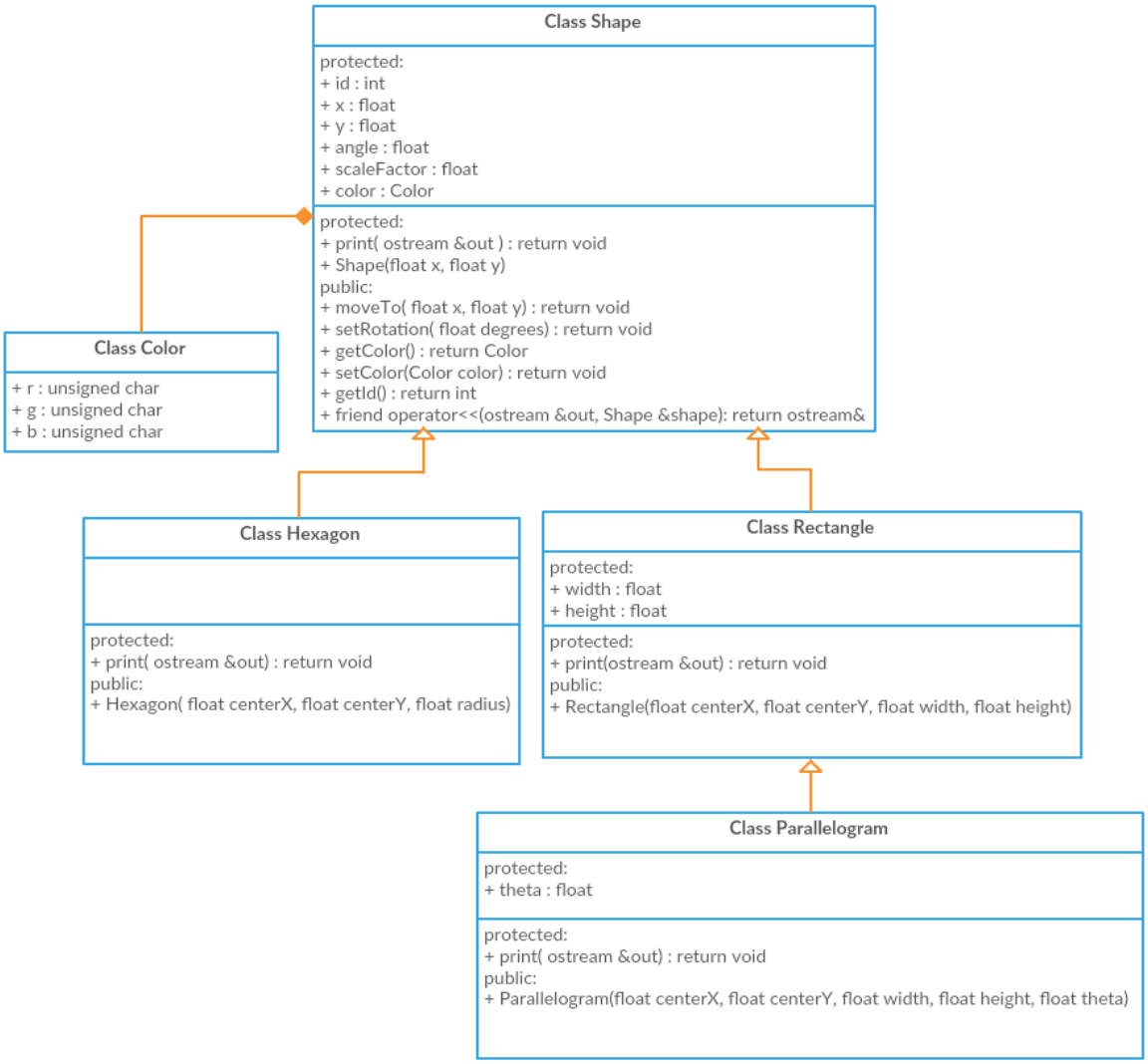
Разработан базовый класс `Shape`, содержащий общие поля для всех геометрических фигур: координаты центра `x`, `y`, угол поворота `angle` относительно центра, множитель масштаба `scaleFactor`, цвет фигуры `color`, а также уникальный идентификатор объекта `id`. Поле color представляет собой структуру Color, имеющую три поля длиной в 1 байт для хранения значений RGB в диапазоне [0 – 255] для каждого компонента.

Разработан класс правильного шестиугольника `Hexagon`, наследующий класс `Shape`. При наследовании не добавляются новые поля, а только переопределяются функции `print`() и конструктор класса.

Разработан класс прямоугольника `Rectangle`, наследующий класс `Shape`. Добавлены поля width и height, для хранения ширины и высоты

прямоугольника соответственно. Также переопределены функции `print`() и конструктор класса.

Разработан класс параллелограмма Parallelogram, наследующий класс Rectangle. Добавлено поле theta, хранящее значение левого нижнего (или правого верхнего) угла параллелограмма. Также переопределены функции `print`() и конструктор класса. На рис. 1 изображена диаграмма классов UML. Исходный код представлен в приложении А.



Выводы.

В ходе работы на языке C++ разработана система из 4 классов, для моделирования геометрических фигур (прямоугольник, параллелограмм, правильный шестиугольник) с использованием виртуальных функций в иерархии наследования и абстрактного базового класса, содержащего общие свойства всех фигур.

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

```
#define toRadian(x) x*6.28/360
inline int sign(float x){
     if(x==0)return 0;
     return (x>0)?1:-1;
}
class Shape{
protected:
    int id;
    float x, y;
    float angle = 0;
    float scaleFactor = 1;
    struct Color{
        unsigned char r,g,b;
    } color;
    virtual void print(ostream &out){}
    Shape(float centerX, float centerY)
        : x(centerX)
        , y(centerY){
           static int ID_COUNTER = 1;
        id = ID_COUNTER++;
     }
public:

    void moveTo(float centerX, float centerY){
        x = centerX;
        y = centerY;
    }
    void setRotation(float degrees){
        this->angle = degrees;
    }
    void scale(float scaleFactor){
        if(scaleFactor==0 || isnan(scaleFactor) || isinf(scaleFactor))
return;
        this->scaleFactor *= scaleFactor;
    }
    Color getColor(){ return color;}
    void setColor(Color color){
        this->color = color;
    }
    int getId(){return id;}
```

```
    friend ostream& operator<<(ostream &out,Shape &shape){
        shape.print(out);
        return out;
    }
};
class Rectangle : public Shape {
protected:
    float width, height;
    void print(ostream &out){
           out << "id: " << id  <<", Rectangle " << width*scaleFactor
<< "X" << height*scaleFactor << " with center in (" << x << "; " << y
<< ") ";
        out << "colored 0x" << std::hex
        << ((color.r&(unsigned char)0xf0)>>4) << (color.r&(unsigned
char)0x0f)
        << ((color.g&(unsigned char)0xf0)>>4) << (color.g&(unsigned
char)0x0f)
        << ((color.b&(unsigned char)0xf0)>>4) << (color.b&(unsigned
char)0x0f);
        if(angle!=0) out << "  rotated on " << angle << " degrees
(clockwise)";
           out << "\nvertices:\n";
           for(int i = 0; i < 4; i++)
                out << '(' << x -
sign(cos(0.785+i*1.57))*(cos(toRadian(angle))*width -
sin(toRadian(angle))*height)*scaleFactor/2
                << "; " << y +
sign(sin(0.785+i*1.57))*(sin(toRadian(angle))*width +
cos(toRadian(angle))*height)*scaleFactor/2 << ")\n";
    }
public:
    Rectangle(float centerX, float centerY, float width, float height)
        : Shape(centerX, centerY){
        scaleFactor = sqrt(width*width + height*height);
        this->width = width/scaleFactor;
        this->height = height/scaleFactor;
    }
};
class Parallelogram : public Rectangle{
protected:
    float theta;
    void print(ostream &out){
```

```
        out << "id: " << id  <<", Parallelogram " << width*scaleFactor
<< "X" << height*scaleFactor << " theta: " << theta<< " with center in
(" << x << "; " << y << ") ";
        out << "colored 0x" << std::hex
        << ((color.r&(unsigned char)0xf0)>>4) << (color.r&(unsigned
char)0x0f)
        << ((color.g&(unsigned char)0xf0)>>4) << (color.g&(unsigned
char)0x0f)
        << ((color.b&(unsigned char)0xf0)>>4) << (color.b&(unsigned
char)0x0f);
        if(angle!=0) out << "  rotated on " << angle << " degrees
(clockwise)";
           out << "\nvertices:\n";
           for(int i = 0; i < 4; i++)
                out << '(' << x -

     sign(cos(0.785+i*1.57))*(cos(toRadian(angle))*scaleFactor/2*(widt
h-sign(tan(0.785+i*1.57))*height*(theta!=0?1/tan(toRadian(theta)):0))
                 - sin(toRadian(angle))*height*scaleFactor/2)
                << "; " << y +

     sign(sin(0.785+i*1.57))*(sin(toRadian(angle))*scaleFactor/2*(widt
h-sign(tan(0.785+i*1.57))*height*(theta!=0?1/tan(toRadian(theta)):0))
                 + cos(toRadian(angle))*height*scaleFactor/2) <<
")\n";
    }
public:
    Parallelogram(float centerX, float centerY, float width, float
height, float theta)
        : Rectangle(centerX, centerY, width, height)
        , theta(theta){}
};
class Hexagon : public Shape{
protected:
     void print(ostream &out){
          out << "id: " << id  <<", Hexagon inscribed in circle of
radius " << scaleFactor << " with center in (" << x << "; " << y << ")
";
        out << "colored 0x" << std::hex
            << ((color.r&(unsigned char)0xf0)>>4) <<
(color.r&(unsigned char)0x0f)
            << ((color.g&(unsigned char)0xf0)>>4) <<
(color.g&(unsigned char)0x0f)
```

```
            << ((color.b&(unsigned char)0xf0)>>4) <<
(color.b&(unsigned char)0x0f);
        if(angle!=0) out << "  rotated on " << angle << " degrees
(clockwise)";
           out << "\nvertices:\n";
           for(int i = 0; i < 6; i++)
            out << '(' << x +
sin(1.0472*i+toRadian(angle))*scaleFactor << "; " << y + cos(1.0472*i
+ toRadian(angle))*scaleFactor << ")\n";
     }
public:
    Hexagon(float centerX,float centerY,float radius)
        : Shape(centerX, centerY){
        this->scaleFactor = radius;
    }

};
```