**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

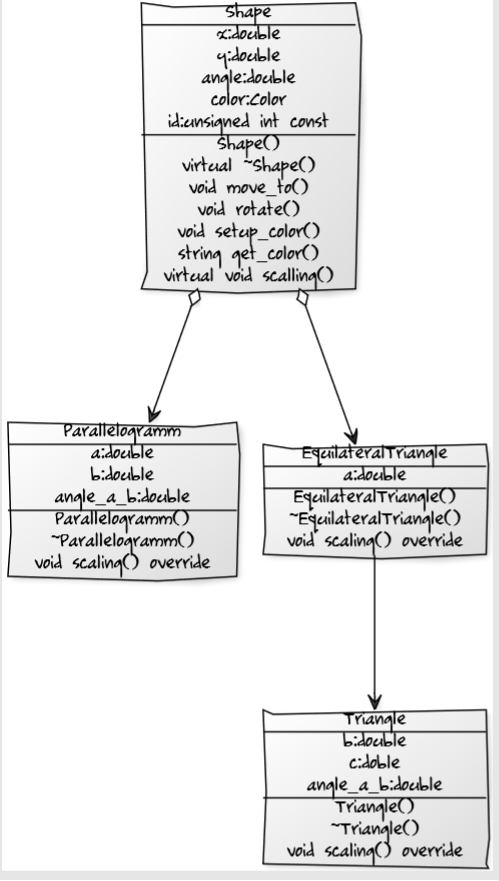
**Цель работы.**

**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 С++.

**Экспериментальные результаты.**

1. Была построена UML диаграмма разработанных классов



1. Был написано текстовое обоснование проектных решений

* В класс Shape были помещены поля x, y, color, angle(угол с осью ох), 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  EquilateralTriangle как расширение частного случая треугольника. Были добавлены атрибуты b - вторая сторона и угол между сторонами а 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 атриубуты a, b - прилежащие стороны и angle\_a\_b - угол между ними. В классе был переопределен метод scaling(), реализован соответствующий оператор вывода.

1. Была сделана реализация классов на языке С++

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;

++next\_id;

}

~**Shape**()

{}

//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**()

{}

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 EquilateralTriangle

{

public:

**Triangle**(double x, double y, double angle, Color color,double a, double b, double angle\_a\_b) : EquilateralTriangle (a, b, angle, color, 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**()

{}

void ***scaling***(double k) override

{

EquilateralTriangle::scaling(k);

b \*= k;

c \*= k;

}

friend std::ostream & operator << (std::ostream & out, Triangle &triangle)

{

out << dynamic\_cast<EquilateralTriangle &>(triangle) << endl << "Side b: " << triangle.b << endl << "Side c:" << triangle.c;

return out;

}

private:

double b;

double angle\_a\_b;

//this side compute by theorem of cos

double c;

};

class **Parallelogram** : public Shape

{

public:

**Parallelogram**(double x, double y, double angle, Color color, double a, double b, double angle\_a\_b) : Shape(x, y, angle, color), a(a), b(b), angle\_a\_b(angle\_a\_b)

{}

~**Parallelogram**()

{}

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 << "Angel between a and b: " << p.angle\_a\_b;

return out;

}

private:

double a;

double b;

//angle betweend sides a and b

double angle\_a\_b;

};

**Выводы.**

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