**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

|  |  |  |
| --- | --- | --- |
| Студентка гр. 7382 |  | Головина Е.С. |
| Преподаватель |  | Жангиров Т. Р. |

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 С++.

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

**Индивидуальное задание.**

Вариант 5 – реализовать систему классов для фигур:

1. Прямоугольник;
2. Эллипс;
3. Сектор эллипса.

**Обоснование проектных решений.**

Для представления цвета написана структура colorShape с полями r, g, b целочисленного типа int, 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 (int x; int y), угол поворота (int angle), цвет (colorShape color) и идентификационный номер (int id).

Для работы с этими параметрами были реализованы следующие методы:

* void move(int x, int y)

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

* virtual void rotate(int angle)

Метод отвечает за поворот фигуры на угол angle*.*

* virtual void scale(int size) = 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(colorShape color)

Метод отвечает за замену цвета фигуры на цвет color.

* colorshape getColor()

Метод отвечает за получение текущего цвета фигуры.

Класс Rectangle является publicнаследником класса Shapeи используется для представления прямоугольника. Он содержит в себе защищенные поля целого типа length и width для хранения длины и ширины прямоугольника соответственно. В этом классе переопределен метод масштабирования (void scale(int size)), который увеличивает длину и ширину прямоугольника в size раз.

Класс Ellipse также является publicнаследником класса Shapeи используется для представления эллипса. Он содержит в себе защищенные поля целого типа big\_semiaxis и small\_semiaxis для хранения размеров большой и малой полуоси эллипса соответственно. В этом классе переопределен метод масштабирования (void scale(int size)), который увеличивает размеры полуосей эллипса в size раз.

Класс *ellipseSector* является *public* наследником класса *Ellipse*. Одно из его отличий от класса Ellipse состоит в наличии защищенных полей целого типа angle\_start и angle\_finish, отвечающие за угол начала сектора и его конца соответственно. Также в этом классе переопределен метод отвечающий за поворот фигуры (void rotate(int angle)), так как в данном методе есть другие угловые параметры — они также должны измениться при повороте на размер angle.

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 С++.**

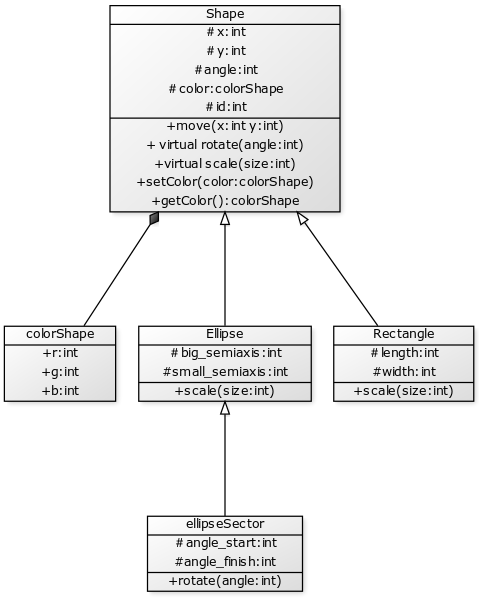
Реализация классов представлена в приложении Б.

**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**

#include <iostream>

#include <cstdlib>

#include <algorithm>

int global\_id=0;

struct colorShape{

int r;

int g;

int b;

friend std::ostream& operator<< (std::ostream &out, const colorShape &color){

out << color.r << ", " << color.g << ", " << color.b

<< "\n";

return out;

}

};

class Shape{

public:

Shape(int x=0, int y=0): x(x), y(y){

color= {0,0,0};

angle=0;

id=global\_id++;

}

virtual ~Shape(){}

void move(int x, int y){

this->x=x;

this->y=y;

}

virtual void rotate(int angle=0){

angle = angle % 360;

if (angle<0) angle+=360;

this->angle += angle;

if (this->angle >= 360) this->angle %= 360;

}

virtual void scale(int size)= 0;

void setColor(colorShape color){

this->color = color;

}

colorShape getColor(){

return color;

}

protected:

int x;

int y;

int angle;

colorShape color;

int id;

};

class Rectangle: public Shape {

public:

Rectangle(int x=0, int y=0, int length=1, int width=1):

Shape(x,y), length(length), width(width){

}

~Rectangle(){}

void scale(int size=1) override{

length\*=size;

width\*=size;

}

friend std::ostream& operator<< (std::ostream& out, const Rectangle& some\_rect){

out <<"Rectangle:\nID = "<< some\_rect.id<<

"\nCentre coordinates = "<< some\_rect.x <<

"," << some\_rect.y << "\nAngle of rotation = "

<< some\_rect.angle << "\nColor(R,G,B) = " <<some\_rect.color

<< "Length = " << some\_rect.length <<"\nWidth = "

<< some\_rect.width

<< "\n\n";

return out;

}

protected:

int length;

int width;

};

class Ellipse: public Shape {

public:

Ellipse(int x=0, int y=0, int big\_semiaxis=2, int small\_semiaxis=1):

Shape(x,y),big\_semiaxis(big\_semiaxis), small\_semiaxis(small\_semiaxis){

if (this->big\_semiaxis < this->small\_semiaxis)

std::swap(this->small\_semiaxis,this->big\_semiaxis);

}

~Ellipse(){}

void scale(int size=1) override{

big\_semiaxis\*=size;

small\_semiaxis\*=size;

}

friend std::ostream& operator<< (std::ostream& out, const Ellipse& some\_ell){

out <<"Ellipse:\nID = "<< some\_ell.id<<

"\nCentre coordinates = "<< some\_ell.x <<

"," << some\_ell.y << "\nAngle of rotation = "

<< some\_ell.angle << "\nColor(R,G,B) = " <<some\_ell.color

<< "Big\_semiaxis = " << some\_ell.big\_semiaxis

<<"\nSmall\_semiaxis = "<< some\_ell.small\_semiaxis

<< "\n\n";

return out;

}

protected:

int big\_semiaxis;

int small\_semiaxis;

};

class ellipseSector: public Ellipse{

public:

ellipseSector(int x=0, int y=0, int b\_semi=2, int s\_semi=1, int angle\_start=0, int angle\_finish=360):

Ellipse(x,y,b\_semi,s\_semi),angle\_start(angle\_start), angle\_finish(angle\_finish){

if (this->angle\_finish > 360) this->angle\_finish%=360;

if (this->angle\_start > 360) this->angle\_finish%=360;

}

~ellipseSector(){}

void rotate (int angle=0) override{

Shape::rotate(angle);

angle\_start+=angle;

angle\_finish+=angle;

if (angle\_finish > 360) angle\_finish%=360;

if (angle\_start > 360) angle\_finish%=360;

}

friend std::ostream& operator<< (std::ostream& out, const ellipseSector& some\_ell){

out <<"Ellipse sector:\nID = "<< some\_ell.id<<

"\nCentre coordinates = "<< some\_ell.x <<

"," << some\_ell.y << "\nAngle of rotation = "

<< some\_ell.angle << "\nColor(R,G,B) = " <<some\_ell.color

<< "Big\_semiaxis = " << some\_ell.big\_semiaxis

<<"\nSmall\_semiaxis = "<< some\_ell.small\_semiaxis

<<"\nStart of the sector(angle) = "<< some\_ell.angle\_start

<<"\nEnd of the sector(angle) = " << some\_ell.angle\_finish

<< "\n\n";

return out;

}

protected:

int angle\_start;

int angle\_finish;

};