**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 С++.

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 publicнаследником 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 publicнаследником 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 С++.**

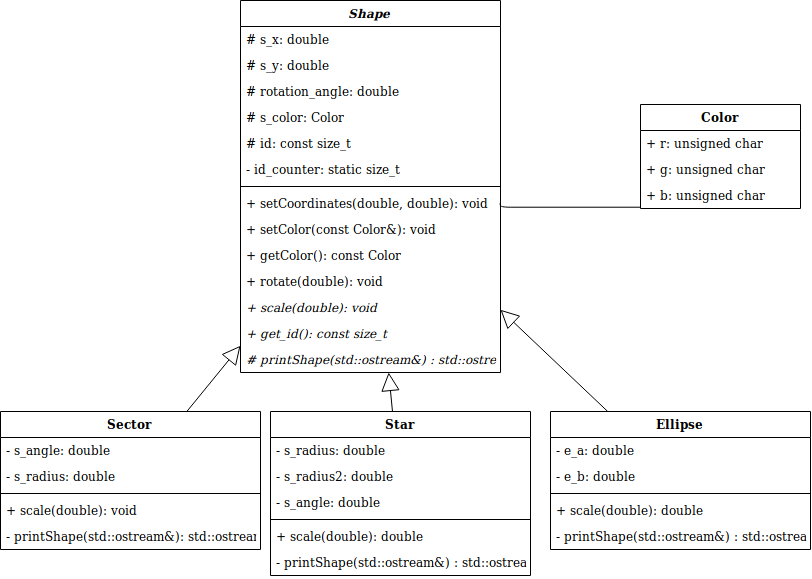
Реализация классов представлена в приложении Б.

**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**

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;}