**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Контейнеры

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Ковынев М.В. |
| Преподаватель |  | Терентьев А.О. |

Санкт-Петербург

2018

**Цель работы.**

Изучить наследование объектов в *C++.*

**Основные теоретические положения.**

*Наследование —* свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником, дочерним или производным классом.

Каждый производный класс полностью сохраняет интерфейс родительского класса. Обратное, очевидно, неверно.

*Перегрузка методов* ***–*** возможность использования методов с одним именем, но с разными параметрами.

*Переопределение методов* – возможность языка программирования, позволяющая производному классу обеспечивать специфическую реализацию методов, уже определенных в базовом классе.

В *C++* метод, обозначаемый ключевым словом *“virtual”*, может быть переопределен производным классом. Компилятор всегда сначала ищет возможность запустить метод, непосредственно объявленный в классе, и лишь при отсутствии такового будет искать метод в базовых классах.

**Постановка задачи.**

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/﻿*

**Ход работы.**

1. Реализована структура *Dot*, отвечающий за создание точек по координатам. Включает в себя данные (переменные *x* и *y*) и конструктор.
2. Написана структура *Color*, отвечающий за цвет. Включает в себя данные (переменные *rgbRed, rgbGreen* и *rgbBlue*) и конструктор.
3. Написан класс *Shape*, описывающий абстрактную фигуру. Класс хранит следующие переменные:

* Угол поворота фигуры типа *int*
* Цвет типа *Color*
* Набор точек типа *vector<Dot>*

В классе реализованы следующие методы:

* Конструктор класса
* Метод установки нового цвета *void changeColor(Color m\_color)*
* Метод получения цвета фигуры *Color getColor() const*
* Метод перемещения фигуры по координатам *void changePosition(double x, double y)*
* Метод для изменения размера фигуры *virtual void scaling(double factor) = 0;*
* Метод поворота фигуры на заданный угол относительно точки (0,0) *void rotateFigure(int deg)*
* Метод получения площади фигуры *virtual double getSquare() = 0;*
* Метод форматированного вывода информации о фигуре в заданный поток *virtual void info(ostream& out) = 0;*
* Деструктор *virtual ~Shape()*
* Перегружен оператор вывода в поток *friend std::ostream& operator<<(std::ostream& out, Shape& sh)*
* Вспомогательный метод для изменения точек фигуры при масштабировании void *scalingDots(double& factor)*

1. Написан класс Rectangle, наследуемый от Shape.

Хранит переменные:

* Периметр прямоугольник типа double
* Ширину прямоугольника типа double
* Высоту прямоугольника типа double

Методы класса:

* Переопределен конструктор
* Переопределен метод вывода
* Переопределен метод масштабирования
* Переопределен метод получения площади

1. Написан класс Circle, наследуемый от Shape.

Хранит переменные:

* Радиус окружности типа double
* Длину окружности типа double

Методы класса:

* + - Переопределен конструктор
    - Переопределен метод вывода
    - Переопределен метод масштабирования
    - Переопределен метод получения площади

1. Написан класс Trapeze, наследуемый от Shape.

Хранит переменные:

* Верхнюю сторону трапеции
* Нижнюю сторону трапеции
* Левую сторону трапеции
* Правую сторону трапеции
* Высоту трапеции

Методы класса:

* Переопределен конструктор
* Переопределен метод вывода
* Переопределен метод масштабирования
* Переопределен метод получения площади

1. Написаны тесты для классов Circle, Trapeze, Rectangle.
2. Код представлен в приложении А.
3. Нарисована UML-диаграмма классов. Рисунок в приложении Б.

**Выводы.**

Изучено наследование классов в C++. Написан базовый класс Shape, от которого наследуются Pentagon, Rectangle и Ellipse.

Приложение А

**КОД ПРОГРАММЫ**

#include <iostream>

#include <vector>

using namespace std;

# define M\_PI 3.14159265358979323846

struct Dot

{

double x;

double y;

};

struct Color

{

unsigned short rgbRed;

unsigned short rgbGreen;

unsigned short rgbBlue;

Color(unsigned short r, unsigned short g,

unsigned short height)

{

rgbRed = r;

rgbGreen = g;

rgbBlue = height;

}

};

class Shape

{

protected:

Color color;

int angle;

vector<Dot> dots;

public:

Shape(int dotsSize) :

color(0, 0, 0), angle(0)

{

dots.resize(dotsSize);

}

void changeColor(Color m\_color)

{

color = m\_color;

}

Color getColor() const

{

return color;

}

void rotateFigure(int deg)

{

angle += deg;

angle %= 360;

double rad = angle \* M\_PI / 180;

for (auto& current\_dot : dots)

{

double tmpx = current\_dot.x\*cos(rad) - current\_dot.y\*sin(rad);

double tmpy = current\_dot.y\*cos(rad) + current\_dot.x\*sin(rad);

current\_dot.x = tmpx;

current\_dot.y = tmpy;

}

}

void changePosition(double x, double y)

{

for (auto& current\_dot : dots)

{

current\_dot.x += x;

current\_dot.y += y;

}

}

virtual double getSquare() = 0;

virtual void scaling(double factor) = 0;

virtual void info(ostream& out) = 0;

virtual ~Shape() {}

friend ostream& operator<<(ostream& out, Shape& shape)

{

shape.info(out);

return out;

}

void scalingDots(double& factor)

{

factor = fabs(factor);

for (auto& current\_dot : dots)

{

current\_dot.x \*= factor;

current\_dot.y \*= factor;

}

}

};

class Rectangle : public Shape

{

protected:

double width;

double height;

double perimeter;

public:

Rectangle(const Dot& centerDot, double m\_width, double m\_height) :

Shape(4), width(m\_width), height(m\_height)

{

dots[0] = { centerDot.x + width / 2, centerDot.y + height / 2 };

dots[1] = { centerDot.x + width / 2, centerDot.y - height / 2 };

dots[2] = { centerDot.x - width / 2, centerDot.y - height / 2 };

dots[3] = { centerDot.x - width / 2, centerDot.y + height / 2 };

perimeter = 2 \* (width + height);

}

void scaling(double factor)

{

scalingDots(factor);

width \*= factor;

height \*= factor;

perimeter \*= factor;

}

double getSquare()

{

return width \* height;

}

void info(ostream& out)

{

out << "############################### INFO ###############################\n";

out << "Rectangle:\n";

out << "List of points:\n";

for (const auto& t : dots)

{

out << "(" << t.x << "," << t.y << ") ";

}

out << "\nColor: (" << color.rgbRed << "," << color.rgbGreen << "," << color.rgbBlue << ")\n";

out << "Width: " << width << " Height: " << height << "\n";

out << "Perimeter: " << perimeter << endl;

out << "####################################################################\n";

}

};

class Circle : public Shape

{

protected:

double radius;

double length;

public:

Circle(Dot centerDot, double r) :

Shape(1), radius(r)

{

dots[0] = centerDot;

length = 2 \* M\_PI \* radius;

}

void scaling(double factor)

{

length \*= abs(factor);

radius \*= abs(factor);

}

double getSquare()

{

return M\_PI \* radius\* radius;

}

void info(ostream& out)

{

out << "############################### INFO ###############################\n";

out << "Circle:\n";

out << "Center: (" << dots[0].x << "," << dots[0].y << ")\n";

out << "Color: (" << color.rgbRed << "," << color.rgbGreen << "," << color.rgbBlue << ")\n";

out << "Radius: " << radius << "\n";

out << "Length: " << length << endl;

out << "####################################################################\n";

}

};

class Trapeze : public Shape

{

protected:

double upperside;

double downside;

double rightside;

double leftside;

double height;

public:

Trapeze(Dot dot1, Dot dot2, Dot dot3, Dot dot4)

: Shape(4)

{

dots[0] = dot1;

dots[1] = dot2;

dots[2] = dot3;

dots[3] = dot4;

upperside = sqrt((dot1.x - dot2.x)\*(dot1.x - dot2.x) + (dot1.y - dot2.y)\*(dot1.y - dot2.y));

rightside = sqrt((dot2.x - dot3.x)\*(dot2.x - dot3.x) + (dot2.y - dot3.y)\*(dot2.y - dot3.y));

downside = sqrt((dot3.x - dot4.x)\*(dot3.x - dot4.x) + (dot3.y - dot4.y)\*(dot3.y - dot4.y));

leftside = sqrt((dot4.x - dot1.x)\*(dot4.x - dot1.x) + (dot4.y - dot1.y)\*(dot4.y - dot1.y));

double temp = ((downside - upperside)\*(downside - upperside) +

leftside\*leftside - rightside \* rightside) / (2\*(downside - upperside));

height = sqrt(leftside\*leftside - temp\*temp);

}

void scaling(double factor)

{

scalingDots(factor);

upperside \*= factor;

downside \*= factor;

rightside \*= factor;

leftside \*= factor;

height \*= factor;

}

double getSquare()

{

return (upperside + downside) \* height / 2.0;

}

void info(ostream& out)

{

out << "############################### INFO ###############################\n";

out << "Trapeze:\n";

out << "List of points:\n";

for (const auto& t : dots)

{

out << "(" << t.x << "," << t.y << ") ";

}

out << "\nColor: (" << color.rgbRed << "," << color.rgbGreen << "," << color.rgbBlue << ")\n";

out << "Upperside: " << upperside << " Downside: " << downside << "\n" ;

out << "Rightside: " << rightside << " Leftside: " << leftside << "\n";

out << "Height: " << height << endl;

out << "####################################################################\n";

}

};

int main()

{

Color Red(255, 0, 0);

Color Green(0, 255, 0);

// ------------- Rectangle tests -------------

Rectangle rect({10.0, 20.0}, 300, 400);

cout << rect;

rect.changeColor(Red);

rect.changePosition(-10, -20);

rect.rotateFigure(90);

rect.scaling(0.01);

cout << rect;

// ------------- Circle tests -------------

Circle cir({ 99.0, 88.0 }, 2.0);

cout << cir;

cir.changeColor(rect.getColor());

cir.changePosition(1, 12);

cir.scaling(4);

cout << cir;

// ------------- Trapeze tests -------------

Trapeze tr({2, 7}, {6, 7}, {9, 0}, {0,0});

cout << tr;

tr.changeColor(Green);

tr.changePosition(0, -10);

tr.rotateFigure(180);

tr.scaling(5);

cout << tr;

cout << "Square of rectangle: " << rect.getSquare() << endl;

cout << "Square of circle: " << cir.getSquare() << endl;

cout << "Square of trapeze: " << tr.getSquare() << endl;

cout << "Sum of squares: " << rect.getSquare() + cir.getSquare() + tr.getSquare() << endl;

return 0;

}

Приложение Б

**UML диаграмма классов**

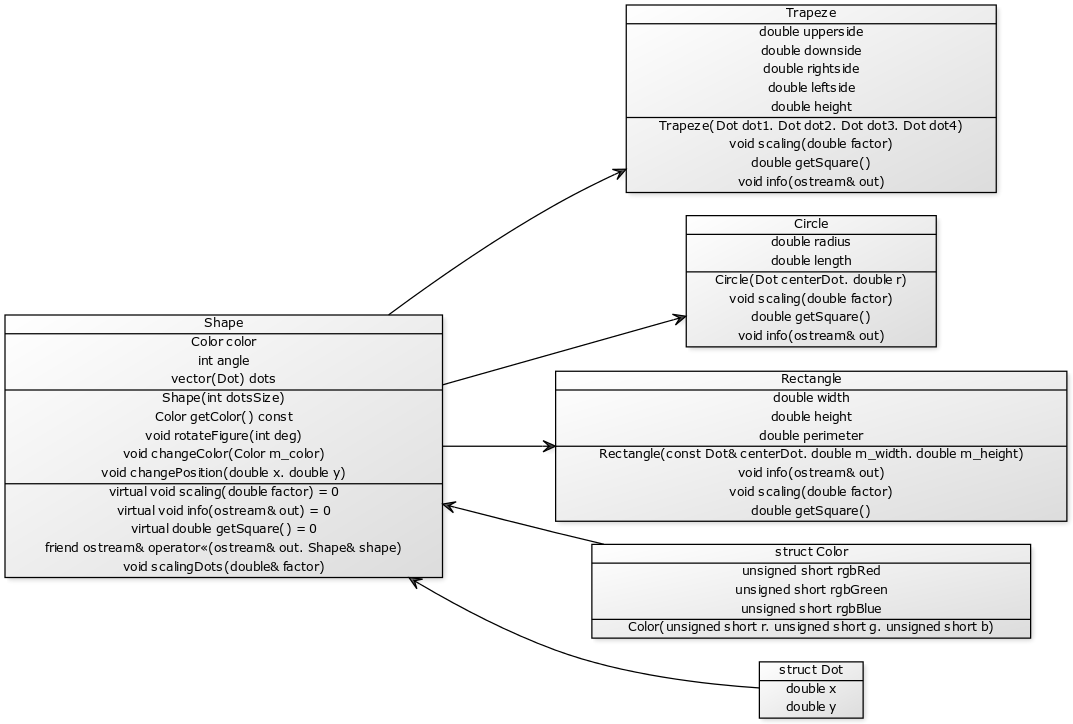


Рисунок 1 — UML диаграмма классов