**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. 6304 |  | Виноградов К. А. |
| 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 неверно.

В C++ метод, обозначаемый ключевым словом “virtual”, может быть переопределен производным классом.

Если в классе есть виртуальные функции, нужно использовать *виртуальные деструкторы*. Это связано с тем, что при создании класса в динамической памяти удаление может производится через базовый класс, а деструктор базового класса не может вызвать деструктор производного. При использовании виртуального деструктора будет вызван деструктор производного класса и уже после этого – базового.

**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.**

Первым делом создадим абстрактный класс Shape, через который будет осуществляться доступ к информации о фигурах, а также будет производиться трансформация фигур. Фигуры будем хранить в виде их основных точек, а также точки центра. Для хранения позиций точек по осям X и Y создадим структуру point, а для хранения массива этих точек – vector<point>. Помимо этих точек в данном классе содержится общая информация о фигурах: положение центра, градус поворота относительно вертикальной оси, цвет, множитель размера.

Pure virtual методы print, setScale, setAngle, shape\_clear, имеющие область видимости private, для того, чтобы к ним не могли обратиться из других классов. Эти методы отвечают за обработку специфической информации, которая уникальна для каждой фигуры.

Далее реализуем непосредственно фигуры. Первым будет eclipse – класс для реализации эллипса. Он будет непосредственно унаследован от shape. Добавим новые переменные – координаты фокусов, фокусное расстояние. Изменим конструктор, чтобы производился расчет необходимых величин. Также переопределим pure virtual методы, описанные выше, чтобы они корректно изменяли значения нужных величин.

Следующим будет класс Trap – для реализации трапеции. Также наследник shape. Новые переменные – высота, длины верхнего и нижнего оснований, отступ середины верхнего основания от центра.

Наконец, нарисуем UML-диаграмму классов. Она представлена в приложении Е.

**Выводы.**

В ходе выполнения данной работы изучено наследование классов в C++. Написан базовый класс Shape, от которого наследуются Eclpise, и Trap. Также нарисована UML-диаграмма этих классов.

Приложение А

shape.h

#pragma once

#define \_USE\_MATH\_DEFINES

#include <cmath>

#include <algorithm>

#include <iostream>

#include <cstdint>

#include <vector>

struct point

{

double x;

double y;

};

class Shape

{

public:

Shape(double ins\_x, double ins\_y)

{

scale = 1;

cen.x = ins\_x;

cen.y = ins\_y;

angle = 0;

colour = { 0, 0, 0 };

}

double getCenx()

{

return cen.x;

}

double getCeny()

{

return cen.y;

}

double toDeg(double rad)

{

return rad \* 180 / M\_PI;

}

double toRad(double deg)

{

return deg \* M\_PI / 180;

}

void setCenter(double ins\_x, double ins\_y)

{

for (auto& p : arr)

{

p.x += ins\_x - cen.x;

p.y += ins\_y - cen.y;

}

this->cen.x = ins\_x;

this->cen.y = ins\_y;

}

virtual void setAngle(double ins\_angle)

{

if (ins\_angle == 360 || ins\_angle == 0)

return;

angle += ins\_angle;

angle = std::fmod(angle, 360);

double temp\_x;

double rang = toRad(ins\_angle);

for (auto& p : arr)

{

temp\_x = p.x;

p.x = cen.x + (temp\_x - cen.x)\*cos(rang) - (p.y - cen.y)\*sin(rang);

p.y = cen.y + (temp\_x - cen.x)\*sin(rang) + (p.y - cen.y)\*cos(rang);

}

}

virtual void setScale(double ins\_scale)

{

scale \*= ins\_scale;

for (auto& p : arr)

{

p.x = (p.x - cen.x) \* ins\_scale + cen.x;

p.y = (p.y - cen.y) \* ins\_scale + cen.y;

}

}

void setColour(uint8\_t r, uint8\_t g, uint8\_t b)

{

colour = { r, g, b };

}

virtual void shape\_clear()

{

cen.x = cen.y = angle = 0;

scale = 1;

colour = { 0, 0, 0 };

arr.clear();

}

virtual void print(std::ostream& out)

{

out << "Class: shape" << '\n';

out << "Center: " << cen.x << ',' << cen.y << '\n';

out << "Angle: " << angle << '\n';

out << "Scale: " << scale << '\n';

out << "RGB: " << colour << '\n';

out << "Point number: " << arr.size() << '\n';

for (auto p : arr)

out << "x: " << p.x << " y: " << p.y << '\n';

out << '\n';

}

friend std::ostream& operator<<(std::ostream& out, Shape& in)

{

in.print(out);

return out;

}

protected:

struct RGB

{

uint8\_t R;

uint8\_t G;

uint8\_t B;

friend std::ostream& operator<<(std::ostream& out, RGB& in)

{

out << (unsigned int)in.R << ' ' << (unsigned int)in.G << ' ' << (unsigned int)in.B << '\n';

return out;

}

} colour;

point cen;

double angle;

double scale;

std::vector<point> arr;

};

class Ellipse : public Shape

{

public:

Ellipse(double ins\_x, double ins\_y, double ins\_foc, double rad\_sml, double rad\_lrg)

:Shape(ins\_x, ins\_y)

{

foc\_rang = 2 \* std::abs(ins\_foc);

l\_foc.x = ins\_x - std::abs(ins\_foc);

l\_foc.y = ins\_y;

r\_foc.x = ins\_x + std::abs(ins\_foc);

r\_foc.y = ins\_y;

point dot;

//Left edge

dot.x = cen.x - std::abs(rad\_lrg);

dot.y = cen.y;

arr.push\_back(dot);

//Right edge

dot.x = cen.x + std::abs(rad\_lrg);

dot.y = cen.y;

arr.push\_back(dot);

//Top edge

dot.x = cen.x;

dot.y = cen.y + std::abs(rad\_sml);

arr.push\_back(dot);

//Bottom edge

dot.x = cen.x;

dot.y = cen.y - std::abs(rad\_sml);

arr.push\_back(dot);

}

void print(std::ostream& out)

{

out << "Class: ellipse" << '\n';

out << "Center: " << cen.x << ',' << cen.y << '\n';

out << "Focus range: " << foc\_rang << "\n";

out << "Left focus: x: " << l\_foc.x << " y: " << l\_foc.y << '\n';

out << "Right focus: x: " << r\_foc.x << " y: " << r\_foc.y << '\n';

out << "Angle: " << angle << '\n';

out << "Scale: " << scale << '\n';

out << "RGB: " << colour << '\n';

out << "Point number: " << arr.size() << '\n';

for (auto p : arr)

out << "x: " << p.x << " y: " << p.y << '\n';

out << '\n';

}

void setAngle(double ins\_angle)

{

if (ins\_angle == 360 || ins\_angle == 0)

return;

angle += ins\_angle;

angle = std::fmod(angle, 360);

double temp\_x;

double rang = toRad(ins\_angle);

temp\_x = l\_foc.x;

l\_foc.x = cen.x + (temp\_x - cen.x)\*cos(rang) - (l\_foc.y - cen.y)\*sin(rang);

l\_foc.y = cen.y + (temp\_x - cen.x)\*cos(rang) + (l\_foc.y - cen.y)\*sin(rang);

temp\_x = r\_foc.x;

r\_foc.x = cen.x + (temp\_x - cen.x)\*cos(rang) - (r\_foc.y - cen.y)\*sin(rang);

r\_foc.y = cen.y + (temp\_x - cen.x)\*cos(rang) + (r\_foc.y - cen.y)\*sin(rang);

for (auto& p : arr)

{

temp\_x = p.x;

p.x = cen.x + (temp\_x - cen.x)\*cos(rang) - (p.y - cen.y)\*sin(rang);

p.y = cen.y + (temp\_x - cen.x)\*sin(rang) + (p.y - cen.y)\*cos(rang);

}

}

void setScale(double ins\_scale)

{

scale \*= ins\_scale;

foc\_rang \*= ins\_scale;

for (auto& p : arr)

{

p.x = (p.x - cen.x) \* ins\_scale + cen.x;

p.y = (p.y - cen.y) \* ins\_scale + cen.y;

}

}

void shape\_clear()

{

cen.x = cen.y = l\_foc.x = l\_foc.y = r\_foc.x = r\_foc.y = angle = foc\_rang = 0;

scale = 1;

colour = { 0, 0, 0 };

arr.clear();

}

protected:

point l\_foc;

point r\_foc;

double foc\_rang;

};

class Trap : public Shape

{

public:

Trap(double ins\_x, double ins\_y, double height\_length, double top\_base\_length, double bottom\_base\_length, double top\_base\_offset)

: Shape(ins\_x, ins\_y), height(std::abs(height\_length)), top\_base(std::abs(top\_base\_length)), bot\_base(std::abs(bottom\_base\_length))

{

point dot;

//Top left

dot.x = cen.x - top\_base / 2 + top\_base\_offset;

dot.y = cen.y + height / 2;

arr.push\_back(dot);

//Top right

dot.x = cen.x + top\_base / 2 + top\_base\_offset;

dot.y = cen.y + height / 2;

arr.push\_back(dot);

//Bot left

dot.x = cen.x - bot\_base / 2;

dot.y = cen.y - height / 2;

arr.push\_back(dot);

//Bot right

dot.x = cen.x + bot\_base /2 ;

dot.y = cen.y - height / 2;

arr.push\_back(dot);

}

void setScale(double ins\_scale)

{

scale \*= ins\_scale;

top\_base \*= ins\_scale;

bot\_base \*= ins\_scale;

height \*= ins\_scale;

for (auto& p : arr)

{

p.x = (p.x - cen.x) \* ins\_scale + cen.x;

p.y = (p.y - cen.y) \* ins\_scale + cen.y;

}

}

void print(std::ostream& out)

{

out << "Class: trapezium" << '\n';

out << "Center: " << cen.x << ',' << cen.y << '\n';

out << "Top base length: " << top\_base << '\n';

out << "Bot base length: " << bot\_base << '\n';

out << "Height length: " << height << '\n';

out << "Angle: " << angle << '\n';

out << "Scale: " << scale << '\n';

out << "RGB: " << colour << '\n';

out << "Point number: " << arr.size() << '\n';

for (auto p : arr)

out << "x: " << p.x << " y: " << p.y << '\n';

out << '\n';

}

void shape\_clear()

{

cen.x = cen.y = angle = top\_base = bot\_base = height = 0;

scale = 1;

colour = { 0, 0, 0 };

arr.clear();

}

protected:

double top\_base;

double bot\_base;

double height;

};

Приложение Б

main.cpp

#include "stdafx.h"

#include "shape.h"

int main()

{

Shape figure(5,5);

Ellipse ellipse(5, 5, 3, 2, 4);

Trap trap(5, 7, 5, 4, 7, 3);

Shape\* shapes[3] = { &figure, &ellipse, &trap };

for (auto& p : shapes)

{

p->print(std::cout);

std::cout << "New info" << std::endl << '\n';

p->setCenter(3, 1);

p->setAngle(65);

p->setColour(255, 210, 255);

p->setScale(3.4);

std::cout << \*p;

std::cout << "Delete info" << std::endl << '\n';

p->shape\_clear();

std::cout << \*p;

std::cout << "Next object ..." << std::endl << '\n';

}

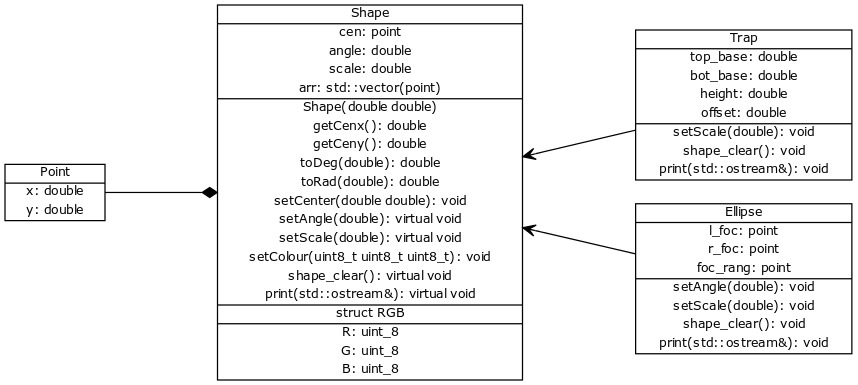
system("pause");

return 0;

}

Приложение В

UML-диаграмма

****