МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**ЛАБОРАТОРНАЯ РАБОТА № 2**

по дисциплине

**«Объектно-ориентированное программирование»**

на тему:

**«Наследование в С++»**

*Вариант № 4*

Выполнил:

Студент группы

КТбо2-8

Данильченко Т. В.

Проверил:

доцент кафедры

МОП ЭВМ

Тарасов С. А.

Оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Таганрог 2020

# **1 Цель работы**

Ознакомление с отношением наследования и иерархией классов.

# **2 Задание, вариант № 4**

# Создать класс Body. На его основе реализовать классы Parallelepiped (прямоугольный параллелепипед), Cone (конус) и Ball (шар) с возможностью вычисления площади поверхности и объема.

# **3 Ход работы**

**3.1 Спецификации классов**

В классе Body определены 2 конструктора(один из создает пустой образец класса, а второй уже перезаписывает значения private полей), 4 метода получение private полей и два виртуальных метода.

В классе Parallelepiped, Cone, Ball определены конструкторы данных классов и уникальные методы поиска площади полной поверхности и объёма фигуры.

**3.2 Диаграмма классов**

В программе используется четыре класса: Body, Parallelepiped, Cone, Ball. Parallelepiped, Cone, Ball связаны с Body отношением наследования. Диаграмма классов представлена на Рисунке 2.

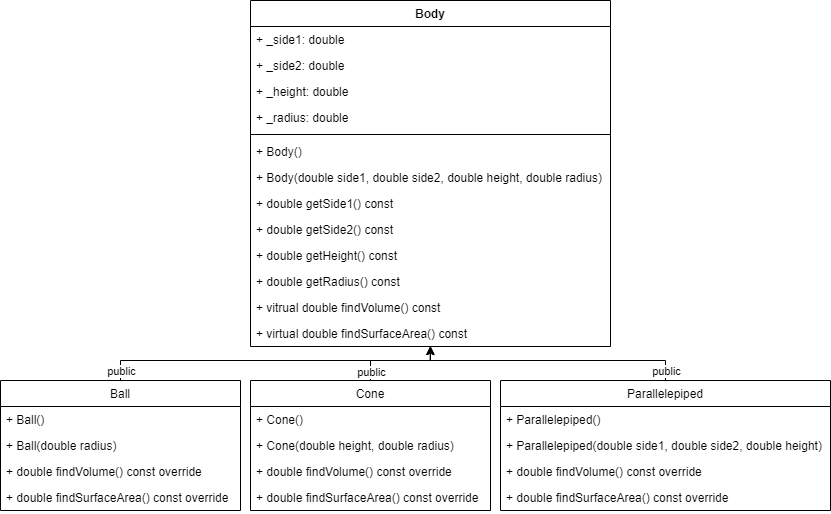


Рисунок 2 — Диаграмма классов

**4 Листинг**

*// Body.h*

#pragma once

#define \_USE\_MATH\_DEFINES

#include <cmath>

class Body {

public:

Body();

Body(double side1, double side2, double height, double radius);

double getSide1() const;

double getSide2() const;

double getHeight() const;

double getRadius() const;

virtual double findVolume() const = 0;

virtual double findSurfaceArea() const = 0;

private:

double \_side1;

double \_side2;

double \_height;

double \_radius;

};

*// Ball.h*

#pragma once

#include "Body.h"

class Ball :public Body {

public:

Ball();

Ball(double radius);

double findVolume() const override;

double findSurfaceArea() const override;

};

*// Cone.h*

#pragma once

#include "Body.h"

class Cone :public Body {

public:

Cone();

Cone(double height, double radius);

double findVolume() const override;

double findSurfaceArea() const override;

};

*// Parallelepiped.h*

#pragma once

#include "Body.h"

class Parallelepiped:public Body {

public:

Parallelepiped();

Parallelepiped(double side1, double side2, double height);

double findVolume() const override;

double findSurfaceArea() const override;

};

*// Console.h*

#pragma once

#include "Parallelepiped.h"

#include "Cone.h"

#include "Ball.h"

class Console {

public:

void run();

void commands() const;

void createFigure();

void findVolume() const;

void findSurfaceArea() const;

private:

int \_sizeB = 0;

int \_sizeC = 0;

int \_sizeP = 0;

Ball\* \_b = new Ball[\_sizeB];

Cone\* \_c = new Cone[\_sizeC];

Parallelepiped\* \_p = new Parallelepiped[\_sizeP];

};

*// Body.cpp*

#include "Body.h"

Body::Body()

: \_side1(0.0), \_side2(0.0), \_height(0.0), \_radius(0.0) {}

Body::Body(double side1, double side2, double height, double radius)

: \_side1(side1), \_side2(side2), \_height(height), \_radius(radius) {}

double Body::getSide1() const {

return \_side1;

}

double Body::getSide2() const {

return \_side2;

}

double Body::getHeight() const {

return \_height;

}

double Body::getRadius() const {

return \_radius;

}

*// Ball.cpp*

#include "Ball.h"

Ball::Ball()

: Body(0.0, 0.0, 0.0, 0.0) {}

Ball::Ball(double radius)

: Body(0.0, 0.0, 0.0, radius) {}

double Ball::findVolume() const {

return 4 \* M\_PI \* getRadius() \* getRadius() \* getRadius() / 3;

}

double Ball::findSurfaceArea() const {

return 4 \* M\_PI \* getRadius() \* getRadius();

}

*// Cone.cpp*

#include "Cone.h"

Cone::Cone()

: Body(0.0, 0.0, 0.0, 0.0) {}

Cone::Cone(double height, double radius)

: Body(0.0, 0.0, height, radius) {}

double Cone::findVolume() const {

return M\_PI \* getRadius() \* getRadius() \* getHeight() / 3;

}

double Cone::findSurfaceArea() const {

double l = sqrt(getRadius() \* getRadius() + getHeight() \* getHeight());

return M\_PI \* getRadius() \* (getRadius() + l);

}

*// Parallelepiped.cpp*

#include "Parallelepiped.h"

Parallelepiped::Parallelepiped()

: Body(0.0, 0.0, 0.0, 0.0) {}

Parallelepiped::Parallelepiped(double side1, double side2, double height)

: Body(side1, side2, height, 0.0) {}

double Parallelepiped::findVolume() const {

return getSide1() \* getSide2() \* getHeight();

}

double Parallelepiped::findSurfaceArea() const {

return 2 \* (getSide1() \* getSide2() + getSide1() \* getHeight() + getSide2() \* getHeight());

}

*// Console.cpp*

#include "Console.h"

#include <iostream>

#include <algorithm>

using std::cin;

using std::cout;

using std::copy;

void Console::commands() const {

cout << "Commands list:\n";

cout << "0: Exit.\n";

cout << "1: Create figure.\n";

cout << "2: Find volume of i-th figure.\n";

cout << "3: Find surface area of i-th figure.\n";

cout << "4: Show commands\n";

}

void Console::createFigure() {

cout << "Enter figure(ball, cone, parallelepiped):";

std::string ans;

cin >> ans;

if (ans == "ball") {

Ball\* buffer = new Ball[\_sizeB + 1];

copy(\_b, \_b + \_sizeB, buffer);

delete[] \_b;

\_b = buffer;

double radius;

cout << "Enter radius value: ";

cin >> radius;

\_b[\_sizeB++] = Ball(radius);

}

else if (ans == "cone") {

Cone\* buffer = new Cone[\_sizeC + 1];

copy(\_c, \_c + \_sizeC, buffer);

delete[] \_c;

\_c = buffer;

double radius, height;

cout << "Enter radius and height values: ";

cin >> radius >> height;

\_c[\_sizeC++] = Cone(height, radius);

}

else if (ans == "parallelepiped") {

Parallelepiped\* buffer = new Parallelepiped[\_sizeP + 1];

copy(\_p, \_p + \_sizeP, buffer);

delete[] \_p;

\_p = buffer;

double side1, side2, height;

cout << "Enter values of base sides and height: ";

cin >> side1 >> side2 >> height;

\_p[\_sizeP++] = Parallelepiped(side1, side2, height);

}

else {

cout << "No such figures.\n";

}

}

void Console::findVolume() const {

cout << "Enter index and figure(example: 4 ball):";

int index;

std::string ans;

cin >> index >> ans;

index--;

if (ans == "ball") {

if (index >= \_sizeB) cout << "Index more than size of balls.\n";

else cout << \_b[index].findVolume() << "\n";

}

else if (ans == "cone") {

if (index >= \_sizeC) cout << "Index more than size of cones.\n";

else cout << \_c[index].findVolume() << "\n";

}

else if (ans == "parallelepiped") {

if (index >= \_sizeP) cout << "Index more than size of parallelepipeds.\n";

else cout << \_p[index].findVolume() << "\n";

}

else {

cout << "No such figures.\n";

}

}

void Console::findSurfaceArea() const {

cout << "Enter index and figure(example: 4 ball):";

int index;

std::string ans;

cin >> index >> ans;

index--;

if (ans == "ball") {

if (index > \_sizeB) cout << "Index more than size of balls.\n";

else cout << \_b[index].findSurfaceArea() << "\n";

}

else if (ans == "cone") {

if (index > \_sizeC) cout << "Index more than size of cones.\n";

else cout << \_c[index].findSurfaceArea() << "\n";

}

else if (ans == "parallelepiped") {

if (index > \_sizeP) cout << "Index more than size of parallelepipeds.\n";

else cout << \_p[index].findSurfaceArea() << "\n";

}

else {

cout << "No such figures.\n";

}

}

void Console::run() {

commands();

int request = 1;

while (request) {

cout << "Enter command number: ";

cin >> request;

switch (request) {

case 0:

cout << "Goodbye.\n";

break;

case 1:

createFigure();

break;

case 2:

findVolume();

break;

case 3:

findSurfaceArea();

break;

case 4:

commands();

break;

default:

cout << "No such command.\n";

break;

}

}

}*// main.cpp*

#include "Console.h"

int main() {

Console console;

console.run();

return 0;

}

1. **Контрольные вопросы**
2. В чем заключается наследование одного класса другому? В чем разница в организации наследования полей и методов?
3. Опишите свою иерархию классов, используя конструкцию «…это есть..».
4. Определены 2 класса:

class Based{

protected: int x;

public:

void setX(int \_x){x=\_x;}

int getX() const {return x;}

};

class Derived:public Based

{};

Какие значения выводятся на консоль?

Based b1;

b1.setX(3);

cout<<b1.getX();

Derived d1;

cout<<d1.getX();

1. Удачной ли является иерархия классов, при которой некоторый класс Х является производным от большого количества классов с большим числом полей в каждом (A<-B <- C <-…<- X)? Какая существует альтернатива наследованию?
2. Есть ли ошибки в нижеследующих объявлениях, если класс Shape абстрактный:

Shape sh;

Shape \*psh;

Shape \*psh1=new Shape();

1. Что из себя представляет виртуальная функция и как она должна вызываться?