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 КТбо2-7

Безрукавая А.С.

Проверил:

Тарасов С. А.

Оценка

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

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

**1 Формулировка задания**

# Создать класс Body. На его основе реализовать классы Parallelepiped (прямоугольный параллелепипед), Cone (конус) и Ball (шар) с возможностью вычисления площади поверхности и объема. **2 Спецификация классов**

**class Body**

{

public:

Body();

Body(double lenghtBase);//когда lenghtBase==height

Body(double lenghtbase, double height);

double GetHeight() const;

double GetLenght() const;

void SetHeight(double const newHeight);

void SetLenght(double const newLenght);

virtual int Type() = 0;

virtual void ShowBody() {};

virtual double AreaSurface();

virtual double Volume();

virtual ~Body()=default;

private:

double \_lenghtBase;//длина основания (== радиус)

double \_height; //для шара=0; радиус для конуса

};

**class Ball** :public Body

{

public:

Ball();

Ball(double radius);//lenghtBase = radius

double GetRadius() const;

friend std::istream& operator>> (std::istream& is, Ball& date);

friend std::ostream& operator<< (std::ostream& os, Ball& date);

void SetRadius(double const newRadius);

int Type() override;

void ShowBody() override;

double AreaSurface() override;

double Volume() override;

private:

const double \_pi = 3.1415;

};

**class Cone** :public Body

{

public:

Cone();

Cone(double radius, double height);//lenghtBase = radius

double GetInclined() const;

friend std::istream& operator>> (std::istream& is, Cone& date);

friend std::ostream& operator<< (std::ostream& os, Cone& date);

void ShowBody() override;

int Type() override;

double AreaSurface() override;

double Volume() override;

private:

double \_inclined;

const double \_pi = 3.1415;

double Sqrt();

};

**class Parallelepiped** :public Body

{

public:

Parallelepiped();

Parallelepiped(double height, double lenght, double width);

double GetWight()const;

void SetWight(double const newWidth);

friend std::istream& operator>> (std::istream& is, Parallelepiped& date);

friend std::ostream& operator<< (std::ostream& os, Parallelepiped& date);

void ShowBody() override;

int Type() override;

double AreaSurface() override;

double Volume() override;

private:

double \_width;

};

**class Parser**

{

public:

virtual Body\* BodyCreate();

private:

};

**class BallParser** : public Parser

{

Body\* BodyCreate() override

{

Ball\* a = new Ball();

std::cin >> (\*a);

std::cout << (\*a);

return a;

}

};

**class ConeParser**:public Parser

{

public:

Body\* BodyCreate() override

{

Cone\* a = new Cone();

std::cin >> (\*a);

std::cout << (\*a);

return a;

}

};

**class ParallelepipedParser** :public Parser

{

Body\* BodyCreate() override

{

Parallelepiped\* a = new Parallelepiped();

std::cin >> (\*a);

std::cout << (\*a);

return a;

}

};

**class IBodyParser**

{

public:

Body\* Parse(int const typeBody) const;

Body\* ChangeBody(Body\* body)const;

private:

Parser\* CuboidEnter()const;

Parser\* ConeEnter()const;

Parser\* BallEnter()const;

};

**class Interface**

{

public:

void Run();

private:

void Info() const;

void InfoBody(double const surface, double const volume) const;

void ShowList()const;

void Change();

void DeleteBody();

int EnterBodyNumber()const;

void IncreaseSpace();

int \_count = -1; // -1 - тел нет

int \_freeSpace = 10;

Body\*\* \_bodies = nullptr;

};

**3** **Используемые математические зависимости и**

**алгоритмы**

При выполнении лабораторной работы использовались следующие математические зависимости:

При работе с прямоугольным параллелепипеда:

S = 2 (ab + bc + ac);

V = abc, где a, b, c – это длина, ширина и высота прямоугольного параллелепипеда, а S и V – это его площадь поверхности и объём.

При работе с конусом:

S = πrl + πr2;

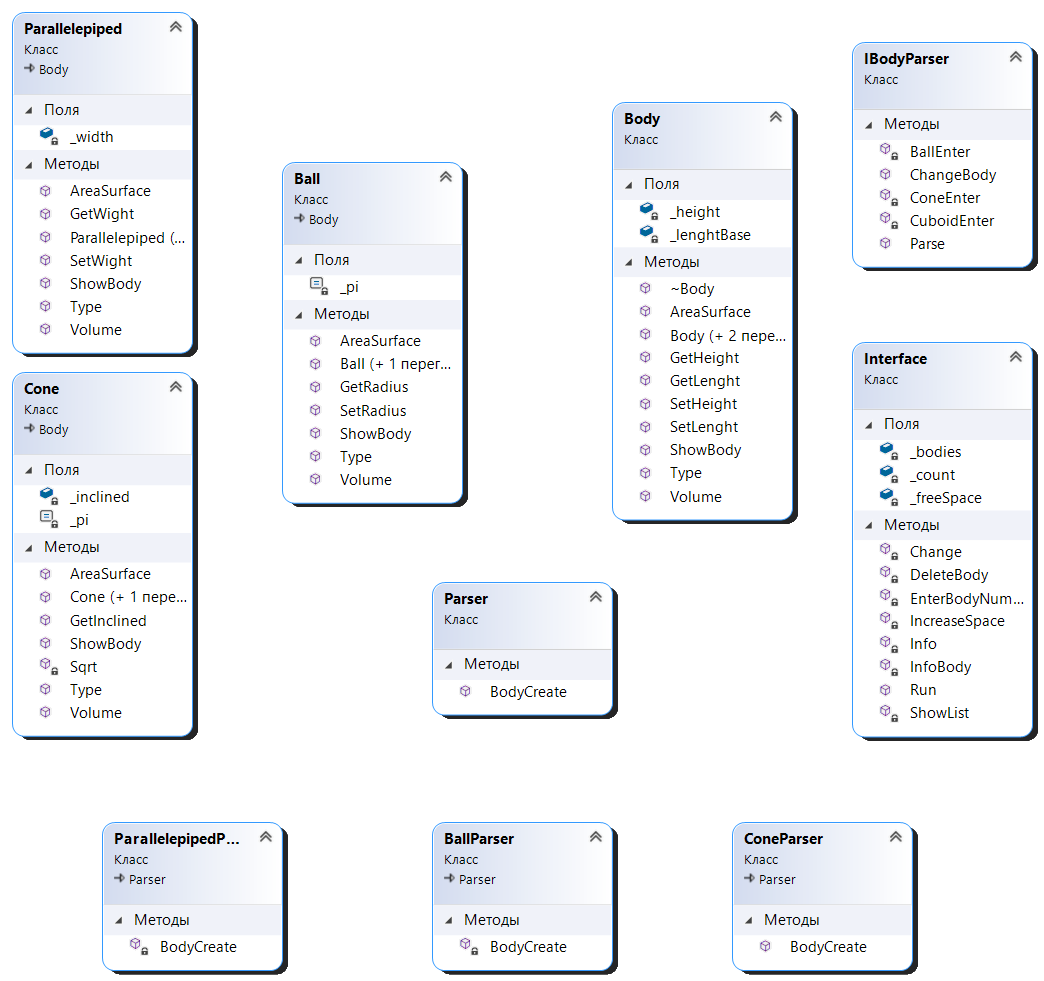
V = πr2h, где r – это радиус основания, h – высота, l – наклонная конуса, а S и V – это его площадь поверхности и объём.

При работе с шаром:

S = πR3;

V = 4 πR2, где R – это радиус шара, а S и V – это его площадь поверхности и объём.

**4 Диаграмма классов**

Ниже на рисунке 1 приведена UML˗диаграмма классов:

**Листинг программы**

**//Ball.h**

#pragma once

#include "Body.h"

#include <istream>

class Ball :public Body

{

public:

Ball();

Ball(double radius);//lenghtBase = radius

double GetRadius() const;

friend std::istream& operator>> (std::istream& is, Ball& date);

friend std::ostream& operator<< (std::ostream& os, Ball& date);

void SetRadius(double const newRadius);

int Type() override;

void ShowBody() override;

double AreaSurface() override;

double Volume() override;

private:

const double \_pi = 3.1415;

};

**//BallParser.h**

#pragma once

#include "Parser.h"

#include "Body.h"

#include <iostream>

class IBodyParser

{

public:

Body\* Parse(int const typeBody) const;

Body\* ChangeBody(Body\* body)const;

private:

Parser\* CuboidEnter()const;

Parser\* ConeEnter()const;

Parser\* BallEnter()const;

};

**//Body.h**

#pragma once

#include <istream>

class Body

{

public:

Body();

Body(double lenghtBase);//когда lenghtBase==height

Body(double lenghtbase, double height);

double GetHeight() const;

double GetLenght() const;

void SetHeight(double const newHeight);

void SetLenght(double const newLenght);

virtual int Type() = 0;

virtual void ShowBody() {};

virtual double AreaSurface();

virtual double Volume();

virtual ~Body()=default;

private:

double \_lenghtBase;//длина основания (== радиус)

double \_height; //для шара=0; радиус для конуса

};

**//Cone.h**

#pragma once

#include "Body.h"

#include <istream>

class Cone :public Body

{

public:

Cone();

Cone(double radius, double height);//lenghtBase = radius

double GetInclined() const;

friend std::istream& operator>> (std::istream& is, Cone& date);

friend std::ostream& operator<< (std::ostream& os, Cone& date);

void ShowBody() override;

int Type() override;

double AreaSurface() override;

double Volume() override;

private:

double \_inclined;

const double \_pi = 3.1415;

double Sqrt();

};

**//ConeParser.h**

#pragma once

#include "Parser.h"

#include "Cone.h"

#include "Body.h"

#include <iostream>

class ConeParser:public Parser

{

public:

Body\* BodyCreate() override

{

Cone\* a = new Cone();

std::cin >> (\*a);

std::cout << (\*a);

return a;

}

};

**//IBodyParser.h**

#pragma once

#include "Parser.h"

#include "Body.h"

#include <iostream>

class IBodyParser

{

public:

Body\* Parse(int const typeBody) const;

Body\* ChangeBody(Body\* body)const;

private:

Parser\* CuboidEnter()const;

Parser\* ConeEnter()const;

Parser\* BallEnter()const;

};

**//Interface.h**

#pragma once

#include "Body.h"

class Interface

{

public:

void Run();

private:

void Info() const;

void InfoBody(double const surface, double const volume) const;

void ShowList()const;

void Change();

void DeleteBody();

int EnterBodyNumber()const;

void IncreaseSpace();

int \_count = -1; // -1 - тел нет

int \_freeSpace = 10;

Body\*\* \_bodies = nullptr;

};

**//Parallelepiped.h**

#pragma once

#include "Body.h"

class Interface

{

public:

void Run();

private:

void Info() const;

void InfoBody(double const surface, double const volume) const;

void ShowList()const;

void Change();

void DeleteBody();

int EnterBodyNumber()const;

void IncreaseSpace();

int \_count = -1; // -1 - тел нет

int \_freeSpace = 10;

Body\*\* \_bodies = nullptr;

};

**//ParallelepipedParser.h**

#pragma once

#include "Body.h"

class Interface

{

public:

void Run();

private:

void Info() const;

void InfoBody(double const surface, double const volume) const;

void ShowList()const;

void Change();

void DeleteBody();

int EnterBodyNumber()const;

void IncreaseSpace();

int \_count = -1; // -1 - тел нет

int \_freeSpace = 10;

Body\*\* \_bodies = nullptr;

};

**//Parser.h**

#pragma once

#include "Body.h"

class Interface

{

public:

void Run();

private:

void Info() const;

void InfoBody(double const surface, double const volume) const;

void ShowList()const;

void Change();

void DeleteBody();

int EnterBodyNumber()const;

void IncreaseSpace();

int \_count = -1; // -1 - тел нет

int \_freeSpace = 10;

Body\*\* \_bodies = nullptr;

};

**//Ball.cpp**

#include "Ball.h"

#include <istream>

#include <iostream>

Ball::Ball():Body()

{

}

Ball::Ball(double radius) :Body(radius)

{

}

double Ball::GetRadius() const

{

return GetLenght();

}

void Ball::SetRadius(double const newRadius)

{

SetLenght(newRadius);

}

int Ball::Type()

{

return 3;

}

void Ball::ShowBody()

{

std::cout << (\*this);

}

double Ball::AreaSurface()

{

return 4 \* GetLenght() \*\_pi;

}

double Ball::Volume()

{

double V3 = GetLenght() \* GetLenght() \* GetLenght();

return 4\*(V3\*\_pi)/3.0;

}

std::istream& operator>>(std::istream& is, Ball& date)

{

double radius;

is >> radius;

date.SetLenght(radius);

if (radius == 0)

throw std::exception("incorrect data");

return is;

}

std::ostream& operator<<(std::ostream& os, Ball& date)

{

os << "Ball";

os << "(" << date.GetRadius() << ")\n";

return os;

}

**//Body.cpp**

#include "Body.h"

Body::Body()

{

\_lenghtBase = 0;

\_height = 0;

}

Body::Body(double lenghtBase)

{

\_lenghtBase = lenghtBase;

\_height = 0;//не используется

}

Body::Body(double lenghtbase, double height)

{

\_lenghtBase = lenghtbase;

\_height = height;

}

double Body::GetHeight() const

{

return \_height;

}

double Body::GetLenght() const

{

return \_lenghtBase;

}

void Body::SetHeight(double const newHeight)

{

\_height = newHeight;

}

void Body::SetLenght(double const newLenght)

{

\_lenghtBase = newLenght;

}

double Body::AreaSurface()

{

return 0;

}

double Body::Volume()

{

return 0;

}

**//Cone.cpp**

#include "Cone.h"

#include <math.h>

#include <istream>

#include <iostream>

Cone::Cone() :Body()

{

\_inclined = 0;

}

Cone::Cone(double radius, double height):Body(radius, height)

{

\_inclined = Sqrt();

}

double Cone::GetInclined() const

{

return \_inclined;

}

void Cone::ShowBody()

{

std::cout << (\*this);

}

int Cone::Type()

{

return 2;

}

double Cone::AreaSurface()

{

return \_pi \* GetLenght() \* (\_inclined + GetLenght());

}

double Cone::Volume()

{

return \_pi \* GetLenght() \* GetLenght() \* GetHeight() / 3;

}

double Cone::Sqrt()

{

return sqrt((GetLenght() \* GetLenght() + GetHeight() \* GetHeight()));

}

std::istream& operator>>(std::istream& is, Cone& date)

{

double lenght, height;

is >> lenght >> height;

date.SetLenght(lenght);

date.SetHeight(height);

if (height == 0 || lenght == 0)

throw std::exception("incorrect data");

return is;

}

std::ostream& operator<<(std::ostream& os, Cone& date)

{

os << "Cone";

os << "(" << date.GetLenght() << ")";

os << "(" << date.GetHeight() << "), ";

os << "forming a cone - " << date.GetInclined() << "\n";

return os;

}

**//IBodyParser.cpp**

#include "IBodyParser.h"

#include "BallParser.h"

#include "ConeParser.h"

#include "ParallelepipedParser.h"

Body\* IBodyParser::Parse(int const typeBody) const

{

std::cout << "enter the parameters of the figure in one value:\n";

int flag = 1;

while (flag)

{

try

{

Parser\* parser = nullptr;

switch (typeBody)

{

case 1: parser = CuboidEnter();

break;

case 2: parser = ConeEnter();

break;

case 3: parser = BallEnter();

break;

default: throw std::exception("Bad body type");

}

Body\* result = parser->BodyCreate();

free(parser);

return result;

}

catch (const std::exception& errorMessage)

{

std::cerr << errorMessage.what() << std::endl;

return nullptr;

}

}

}

Body\* IBodyParser::ChangeBody(Body\* body) const

{

int type = body->Type();

delete(body);

return Parse(type);

}

Parser\* IBodyParser::CuboidEnter()const

{

std::cout << "height, lenght, wight\n";

return new ParallelepipedParser();

}

Parser\* IBodyParser::ConeEnter()const

{

std::cout << "radius\n";

return new BallParser();

}

Parser\* IBodyParser::BallEnter() const

{

std::cout << "the radius of the base, height\n";

return new ConeParser();

}

**//Interface.cpp**

#include "Interface.h"

#include "Body.h"

#include "IBodyParser.h"

#include <iostream>

using std::cin;

using std::cout;

void Interface::Run()

{

\_bodies = new Body \* [10];

int choice =1;

while (choice)

{

Info();

cin >> choice;

if (choice < 4 && choice > 0)

{

IBodyParser\* parse = new IBodyParser();

\_count++;

\_bodies[\_count] = parse->Parse(choice);

if (\_bodies[\_count] == nullptr)

\_count--;

if ((\_count+1) == \_freeSpace)

IncreaseSpace();

InfoBody(\_bodies[\_count]->AreaSurface(), \_bodies[\_count]->Volume());

delete(parse);

}

else

{

switch (choice)

{

case 0:

break;

case 4: ShowList();

break;

case 5: Change();

break;

case 6: DeleteBody();

break;

default:

cout << "incorrectly";

break;

}

}

}

for (int i = 0; i <= \_count; i++)

delete[] \_bodies [i];

delete[] \_bodies;

}

void Interface::Info() const

{

cout << "1 - new parallelepiped\n";

cout << "2 - new ball\n";

cout << "3 - new cone\n";

cout << "4 - to show the parameters of the existing bodies\n";

cout << "5 - change the parameters of an existing shape\n";

cout << "6 - delete the body (the last existing shape will be placed in the place of the deleted shape)\n";

cout << "0 - exit\n";

}

void Interface::InfoBody(double const surface, double const volume) const

{

cout << " >> total surface area : " << surface << "\n";

cout << " >> valume : " << volume << "\n";

}

void Interface::ShowList()const

{

cout << "total number of bodies: " << \_count+1 << "\n";

for (int i = 0; i <= \_count; i++)

{

cout << i+1 << " :\n";

\_bodies[i]->ShowBody();

InfoBody(\_bodies[i]->AreaSurface(), \_bodies[i]->Volume());

}

}

void Interface::Change()

{

int number = EnterBodyNumber();

if (number+1)

{

IBodyParser\* parse= new IBodyParser();

\_bodies[number] = parse->ChangeBody(\_bodies[number]);

delete(parse);

}

}

void Interface::DeleteBody()

{

int number = EnterBodyNumber();

if (number+1)

{

delete[]\_bodies[number];

\_bodies[number] = \_bodies[\_count];

\_bodies[\_count] = nullptr;

\_count--;

}

cout << "Complete!\n";

ShowList();

}

int Interface::EnterBodyNumber() const

{

int number;

cout << "enter one body number(" << \_count+1 << " in total)\n";

cin >> number;

number--;

if (number < 0 || number > \_count)

{

cout << "no correct number\n";

return -1;

}

return number;

}

void Interface::IncreaseSpace()

{

Body\*\* (copy) = new Body\* [\_freeSpace + 10];

for (int i = 0; i < \_freeSpace; i++)

{

copy[i] = \_bodies[i];

}

delete [] \_bodies;

\_bodies = copy;

\_freeSpace += 10;

}

**//Laba2.cpp**

// Создать класс Body. На его основе реализовать классы

//Parallelepiped(прямоугольный параллелепипед), Cone(конус) и

//Ball(шар) с возможностью вычисления площади поверхности и

//объема.

#include <vld.h>

#include "Interface.h"

int main()

{

Interface a;

a.Run();

return 0;

}

**//Parallelepiped.cpp**

#include "Parallelepiped.h"

#include "Body.h"

#include <istream>

#include <iostream>

Parallelepiped::Parallelepiped():Body()

{

\_width = 0;

}

Parallelepiped::Parallelepiped(double height, double lenght, double width):Body(lenght, height)

{

\_width = width;

}

double Parallelepiped::GetWight() const

{

return \_width;

}

void Parallelepiped::SetWight(double const newWidth)

{

\_width = newWidth;

}

void Parallelepiped::ShowBody()

{

std::cout << (\*this);

}

int Parallelepiped::Type()

{

return 1;

}

double Parallelepiped::AreaSurface()

{

return (2\*(\_width \* GetLenght() + GetLenght()\* GetHeight() + \_width \*GetHeight())) ;

}

double Parallelepiped::Volume()

{

return (GetLenght()\*\_width \*GetLenght());

}

std::istream& operator>>(std::istream& is, Parallelepiped& date)

{

double height, lenght;

is >> height >> lenght >> date.\_width;

date.SetHeight(height);

date.SetLenght(lenght);

if (height == 0 || lenght == 0 || date.\_width == 0)

throw std::exception("incorrect data");

return is;

}

std::ostream& operator<<(std::ostream& os, Parallelepiped& date)

{

os << "Parallelepiped";

os << "(" << date.GetHeight() << ")";

os << "(" << date.GetLenght() << ")";

os << "(" << date.GetWight() << ")\n";

return os;

}

**//Parser.cpp**

#include "Parser.h"

Body\* Parser::BodyCreate()

{

return nullptr;

}