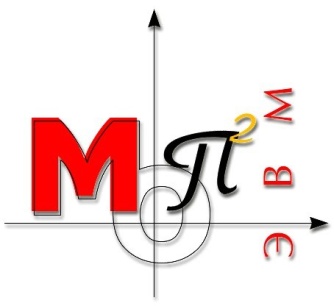
Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждения высшего образования   
«Южный федеральный университет»

Инженерно-технологическая академия

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**

Лабораторная работа № 3

по дисциплине

"ООП"

на тему

"Фабрика в С++"

**Выполнил:**

студент группы КТбо2-1

Сафронов Н. А.

**Проверил:**

Таганрог, 2021

**Вариант №7**

**Задание:**

Создать класс Solution (решение) с методами вычисления корней уравнения. На его основе реализовать классы Linear (линейное уравнение) и Square (квадратное уравнение). Для создания массивов решений уравнений (не систем) с различными аргументами создать дополнительный класс Series.

**Листинг кода:**

**Main.cpp**

#include <iostream>

#include "Interface.h"

//#include <vld.H>

int main()

{

setlocale(LC\_ALL, "Russian");

Interface interface;

interface.Go();

return 0;

}

**Interface.h**

#pragma once

#include "LinearEquation.h"

#include "QuadraticEquation.h"

#include "EquationFactory.h"

class Interface

{

public:

void Go();

~Interface();

private:

void printMenuText();

void dialog();

void dialogEquationCount();

void createEmptyArr();

void createEquation();

int figureID() const;

int \_count = 0;

Solution\*\* \_equation = nullptr;

};

**Interface.cpp**

#pragma once

#include "Interface.h"

#include "LinearEquation.h"

#include "QuadraticEquation.h"

#include <iostream>

using std::cin;

using std::cout;

using std::endl;

Interface::~Interface()

{

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

if (\_equation[i] != nullptr) {

delete \_equation[i];

}

}

delete[] \_equation;

}

void Interface::Go()

{

try {

dialogEquationCount();

createEmptyArr();

createEquation();

dialog();

}

catch (const std::exception& exept) {

cout << exept.what() << endl;

}

}

void Interface::dialogEquationCount()

{

do {

cout << "How many equations will there be?: ";

cin >> \_count;

} while (\_count <= 0);

}

void Interface::createEmptyArr()

{

\_equation = new Solution \* [\_count];

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

\_equation[i] = nullptr;

}

}

void Interface::createEquation()

{

char T;

std::vector<double> params(3);

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

cout << "\n---Equation #" << i + 1 << "---" << endl;

cout << "What equation to create? | (L) - linear equation, (Q) - quadratic equation" << std::endl;

cin >> T;

if (T == 'L')

{

cout << "Enter coefficients a, b (a \* x - b)" << endl;

cin >> params[0] >> params[1];

}

if (T == 'Q')

{

cout << "Enter coefficients a, b, c (a \* x - b \* x + c + 0)" << endl;

cin >> params[0] >> params[1] >> params[2];

}

\_equation[i] = EquationFactory().createEquation(T, params);

}

}

void Interface::dialog()

{

bool running = true;

while (running)

{

printMenuText();

int input;

cin >> input;

try

{

switch (input)

{

case 0:

{

running = false;

cout << "GoodBuy\n";

}

break;

case 1:

{

auto result = \_equation[figureID()]->CalculateXResult();

for (double i : result)

{

cout << i << endl;

}

}

break;

default:

{

running = false;

}

break;

}

}

catch (const std::logic\_error& error)

{

cout << error.what() << endl;

}

}

}

int Interface::figureID() const

{

int figure\_id;

do {

cout << "Enter Equation ID: " << endl;

cin >> figure\_id;

} while (figure\_id < 1 || figure\_id > \_count);

return figure\_id - 1;

}

void Interface::printMenuText()

{

cout << "\n-----------------------------" << endl;

cout << "1 - Print the answer to the equation by ID" << endl;

cout << "0 - Exit" << endl;

cout << "-----------------------------" << endl;

}

**EquationFactory.h**

#pragma once

#include "Solution.h"

#include "LinearEquation.h"

#include "QuadraticEquation.h"

class EquationFactory {

public:

Solution\* createEquation(char T, const std::vector<double>& params);

};

**EquationFactory.cpp**

#include "EquationFactory.h"

Solution\* EquationFactory::createEquation(char T, const std::vector<double>& params) {

switch (T) {

case 'L':

{

return new LinearEquation(params[0], params[1]);

}

case 'Q':

{

return new QuadraticEquation(params[0], params[1], params[2]);

}

default:

throw std::exception("Incorrect equation type!");

}

}

**Solution.h**

class Solution

{

public:

Solution(const std::vector<double>& ratioVector);

virtual std::vector<double> CalculateXResult() = 0;

virtual ~Solution() = default;

protected:

double GetRatio(int i) const;

private:

std::vector<double> \_ratio;

};

**Solution.cpp**

#include "Solution.h"

Solution::Solution(const std::vector<double>& ratioVector)

: \_ratio(ratioVector)

{

}

double Solution::GetRatio(int i) const

{

return \_ratio[i];

}

**LinearEquation.h**

#pragma once

#include "Solution.h"

class LinearEquation : public Solution {

public:

LinearEquation(double a, double b);

virtual std::vector<double> CalculateXResult() override;

};

**LinearEquation.cpp**

#include "LinearEquation.h"

LinearEquation::LinearEquation(double a, double b)

: Solution(std::vector<double>({a, b}))

{

}

std::vector<double> LinearEquation::CalculateXResult()

{

return std::vector<double>(1, -GetRatio(0) / GetRatio(1));

}

**QuadraticEquation.h**

#pragma once

#include "Solution.h"

class QuadraticEquation :public Solution {

public:

QuadraticEquation(double a, double b, double c);

virtual std::vector<double> CalculateXResult() override;

private:

void CheckDisscriminant();

};

**QuadraticEquation.cpp**

#include "QuadraticEquation.h"

#include <stdexcept>

QuadraticEquation::QuadraticEquation(double a, double b, double c)

: Solution(std::vector<double>({ a, b, c }))

{

}

void QuadraticEquation::CheckDisscriminant()

{

if ((pow(GetRatio(0), 2) - (4 \* GetRatio(1) \* GetRatio(2))) < 0)

{

throw std::logic\_error("Discriminant < 0");

}

}

std::vector<double> QuadraticEquation::CalculateXResult()

{

CheckDisscriminant();

std::vector<double> result(0);

result.push\_back((-GetRatio(0) + sqrt(pow(GetRatio(0), 2) -

4 \* GetRatio(1) \* GetRatio(2))) / (2 \* GetRatio(1)));

result.push\_back((-GetRatio(0) - sqrt(pow(GetRatio(0), 2)

- 4 \* GetRatio(1) \* GetRatio(2))) / (2 \* GetRatio(1)));

return result;

}