МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа №5**

по дисциплине: Объектно-ориентированное программирование Тема: Классы, виды отношений. Наследование.

Выполнил: студент группы ПВ-223

Дмитриев А.А.

Проверил:

Черников С.В.

Белгород 2024 г.

**Цель работы:** Получение теоретических знаний в области разработки классов, получение практических навыков реализаций классов и отношений между ними.

**Задание:** Программа решения заданных произвольных уравнений.

1. Создать абстрактный класс Function с методом вычисления значения

функции y=f(x) в заданной точке.

2. Создать производные классы:

Line (y=ax+b),

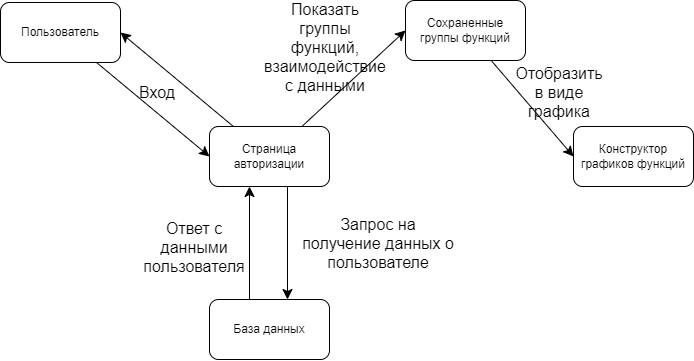
Parabola (y=ax 2 +bx +c),

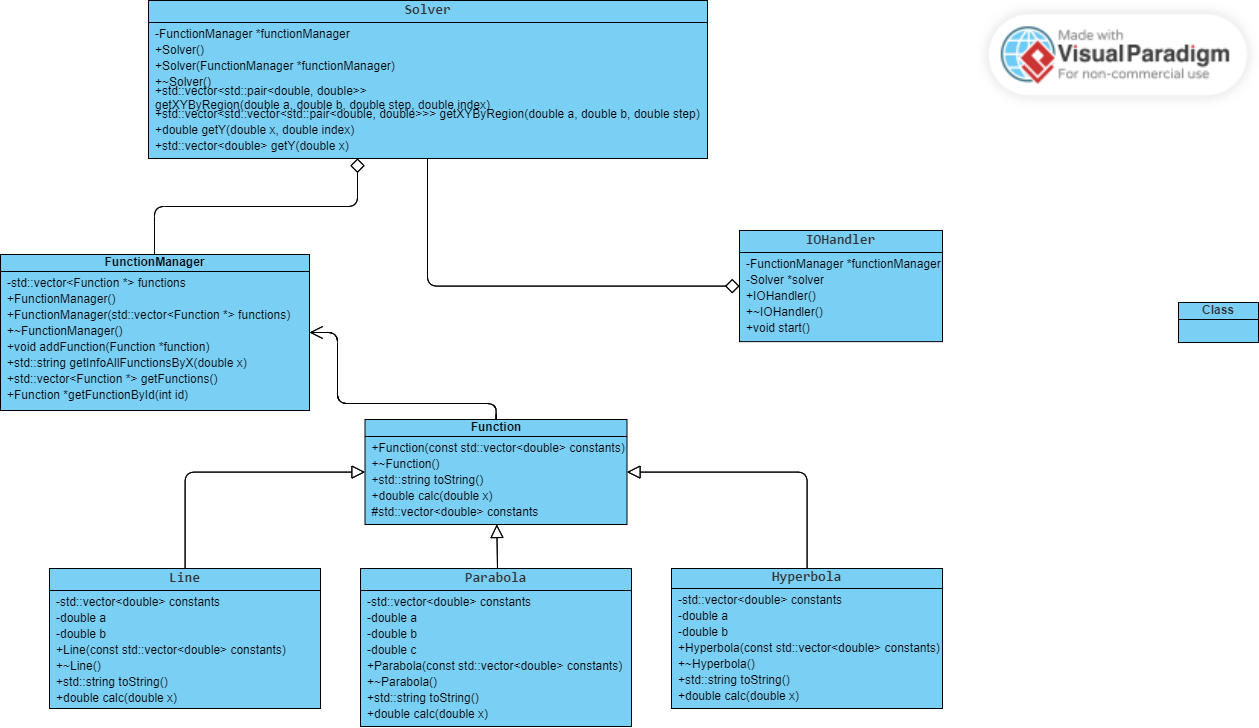
Hyperbola (y=a/x+b)

со своими методами вычисления значения в заданной точке.

3. Создать массив n функций и вывести полную информацию о значении

данных функций в точке x.





class Function

{

private:

public:

    Function(const std::vector<double> constants);

    ~Function();

    virtual std::string toString();

    virtual double calc(double x) = 0;

protected:

    std::vector<double> constants;

};

class Hyperbola : public Function

{

private:

    std::vector<double> constants;

    double a;

    double b;

public:

    Hyperbola(std::vector<double> constants);

    ~Hyperbola();

    double calc(double x) override;

    std::string toString() override;

};

class Line : public Function

{

private:

    std::vector<double> constants;

public:

    Line(std::vector<double> constants);

    ~Line();

    double calc(double x) override;

    std::string toString() override;

protected:

    double a;

    double b;

};

class Parabola : public Function

{

private:

    std::vector<double> constants;

public:

    Parabola(std::vector<double> constants);

    ~Parabola();

    double calc(double x) override;

    std::string toString() override;

protected:

    double a;

    double b;

    double c;

};

class FunctionManager

{

private:

    std::vector<Function \*> functions;

public:

    FunctionManager();

    FunctionManager(std::vector<Function \*> functions);

    ~FunctionManager();

    Function \*getFunctionById(int id);

    std::vector<Function \*> getFunctions();

    void addFunction(Function \*function);

    size\_t size();

};

class IOHandler

{

private:

    FunctionManager \*functionManager;

    Solver \*solver;

public:

    IOHandler();

    ~IOHandler();

    void start();

};

class Solver

{

private:

    FunctionManager \*functionManager;

public:

    Solver();

    Solver(FunctionManager \*functionManager);

    ~Solver();

    std::vector<std::pair<double, double>> getXYByRegion(double a, double b, double step, double index);

    std::vector<std::vector<std::pair<double, double>>> getXYByRegion(double a, double b, double step);

    double getY(double x, double index);

    std::vector<double> getY(double x);

};

Function::Function(std::vector<double> constants) {

    this->constants = constants;

}

Function::~Function() {

    delete this;

}

std::string Function::toString() {

    std::string res;

    for (double constant : constants)

        res += std::to\_string(constant) + " ";

    return res;

}

Hyperbola::Hyperbola(std::vector<double> constants) : Function::Function(constants)

{

    if (constants.size() != 2)

        throw std::invalid\_argument("num of constants not equal 2");

    a = constants[0];

    b = constants[1];

}

Hyperbola::~Hyperbola() { delete this; }

double Hyperbola::calc(double x)

{

    return a / x + b;

}

std::string Hyperbola::toString() {

    return std::to\_string(a) + " / x + " + std::to\_string(b);

}

Line::Line(std::vector<double> constants) : Function::Function(constants)

{

    if (constants.size() != 2)

        throw std::invalid\_argument("num of constants not equal 2");

    a = constants[0];

    b = constants[1];

}

Line::~Line() {

    delete this;

}

double Line::calc(double x)

{

    return a \* x + b;

}

std::string Line::toString() {

    return std::to\_string(a) + " \* x + " + std::to\_string(b);

}

Parabola::Parabola(std::vector<double> constants) : Function::Function(constants)

{

    if (constants.size() != 3)

        throw std::invalid\_argument("num of constants not equal 3");

    a = constants[0];

    b = constants[1];

    c = constants[2];

}

Parabola::~Parabola()

{

    delete this;

}

double Parabola::calc(double x)

{

    return a \* x \* x + b \* x + c;

}

std::string Parabola::toString() {

    return std::to\_string(a) + " \* x2 + " + std::to\_string(b) + " x + " + std::to\_string(c);

}

FunctionManager::FunctionManager()

{

}

FunctionManager::FunctionManager(std::vector<Function \*> functions)

{

    this->functions = functions;

}

FunctionManager::~FunctionManager()

{

    delete this;

}

void FunctionManager::addFunction(Function \*function)

{

    functions.push\_back(function);

}

std::vector<Function \*> FunctionManager::getFunctions()

{

    return functions;

}

Function \*FunctionManager::getFunctionById(int id)

{

    return functions[id];

}

size\_t FunctionManager::size()

{

    return functions.size();

}

#include "../../headers/function/IOHandler.h"

#include "../../headers/function/Solver.h"

#include "../../headers/function/specific/Line.h"

#include "../../headers/function/specific/Hyperbola.h"

#include "../../headers/function/specific/Parabola.h"

#include <iostream>

IOHandler::IOHandler()

{

    this->functionManager = new FunctionManager();

    this->solver = new Solver(functionManager);

}

IOHandler::~IOHandler()

{

    delete this;

}

void IOHandler::start()

{

    int command\_index;

    while (true)

    {

        std::cout << "1- add function\n2- calc functions\n";

        std::cin >> command\_index;

        switch (command\_index)

        {

        case 1:

            std::cout << "\nChange type of function\n1- Line\n2- Hyperbola\n3- Parabola\n";

            std::cin >> command\_index;

            double a;

            std::cout << "input a: ";

            std::cin >> a;

            double b;

            std::cout << "input b: ";

            std::cin >> b;

            switch (command\_index)

            {

            case 1:

                functionManager->addFunction(new Line({a, b}));

                break;

            case 2:

                functionManager->addFunction(new Hyperbola({a, b}));

                break;

            case 3:

                double c;

                std::cout << "input c: ";

                std::cin >> c;

                functionManager->addFunction(new Parabola({a, b, c}));

                break;

            default:

                std::cerr << "Incorrect command index\n";

                break;

            }

            break;

        case 2:

            std::cout << "\nHow much function wou want calculate\n1- One\n2- Many\n";

            std::cin >> command\_index;

            switch (command\_index)

            {

            case 1:

                for (size\_t i = 0; i < functionManager->size(); i++)

                {

                    std::cout << i + 1 << ' ' << functionManager->getFunctionById(i)->toString() << '\n';

                }

                int index;

                std::cout << "Change function:\n";

                std::cin >> index;

                std::cout << "1- By X\n2- By range\n";

                std::cin >> command\_index;

                switch (command\_index)

                {

                case 1:

                    double x;

                    std::cout << "Write X\n";

                    std::cin >> x;

                    std::cout << "Result: "

                              << "f(" << x << ") = " << solver->getY(x, index-1) << '\n';

                    break;

                case 2:

                    double a;

                    std::cout << "Write A\n";

                    std::cin >> a;

                    double b;

                    std::cout << "Write B\n";

                    std::cin >> b;

                    double step;

                    std::cout << "Write step\n";

                    std::cin >> step;

                    for (auto result : solver->getXYByRegion(a, b, step, index-1))

                        std::cout << result.first << "\t| " << result.second << '\n';

                    break;

                default:

                    std::cerr << "Incorrect command index\n";

                    break;

                }

                break;

            case 2:

                std::cout << "1- By X\n2- By range\n";

                std::cin >> command\_index;

                switch (command\_index)

                {

                case 1:

                    double x;

                    std::cout << "Write X\n";

                    std::cin >> x;

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

                    for (size\_t i = 0; i < functionManager->size(); i++)

                        std::cout << "f(" << x << ") = " << solver->getY(x, i) << '\n';

                    break;

                case 2:

                    double a;

                    std::cout << "Write A\n";

                    std::cin >> a;

                    double b;

                    std::cout << "Write B\n";

                    std::cin >> b;

                    double step;

                    std::cout << "Write step\n";

                    std::cin >> step;

                    for (auto table : solver->getXYByRegion(a, b, step))

                    {

                        for (auto result : table)

                            std::cout << result.first << " | " << result.second << '\n';

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

                    }

                    break;

                default:

                    std::cerr << "Incorrect command index\n";

                    break;

                }

                break;

            default:

                std::cerr << "Incorrect command index\n";

                break;

            }

        }

    }

}

Solver::Solver()

{

}

Solver::Solver(FunctionManager \*functionManager)

{

    this->functionManager = functionManager;

}

Solver::~Solver()

{

    delete this;

}

std::vector<std::pair<double, double>> Solver::getXYByRegion(double a, double b, double step, double index)

{

    auto res = std::vector<std::pair<double, double>>();

    while (a < b)

    {

        res.push\_back({a, getY(a, index)});

        a += step;

    }

    return res;

}

std::vector<std::vector<std::pair<double, double>>> Solver::getXYByRegion(double a, double b, double step)

{

    auto res = std::vector<std::vector<std::pair<double, double>>>();

    for (size\_t i = 0; i < functionManager->size(); i++)

    {

        res.push\_back(getXYByRegion(a, b, step, i));

    }

    return res;

}

double Solver::getY(double x, double index)

{

    return functionManager->getFunctionById(index)->calc(x);

}

std::vector<double> Solver::getY(double x)

{

    auto res = std::vector<double>();

    for (size\_t i = 0; i < functionManager->size(); i++)

    {

        res.push\_back(getY(x, i));

    }

    return res;

}

Пример работы:

int main()

{

    Line l1({2, 2});

    Line l2({2, 2});

    Parabola p1({3, 2, 1});

    Line l3({2, 2});

    Hyperbola h1({2, 2});

    FunctionManager fm({&l1, &l2, &l3});

    fm.addFunction(&p1);

    fm.addFunction(&h1);

    std::cout << fm.getInfoAllFunctionsByX(4);

    return 0;

}

Выходные данные:

1- add function

2- calc functions

1

Change type of function

1- Line

2- Hyperbola

3- Parabola

1

input a: 7

input b: 6

1- add function

2- calc functions

1

Change type of function

1- Line

2- Hyperbola

3- Parabola

3

input a: 4

input b: 5

input c: 6

1- add function

2- calc functions

2

How much function wou want calculate

1- One

2- Many

1

1 7.000000 \* x + 6.000000

2 4.000000 \* x2 + 5.000000 x + 6.000000

Change function:

1

1- By X

2- By range

1

Write X

8

Result: f(8) = 62

1- add function

2- calc functions

2

How much function wou want calculate

1- One

2- Many

2

1- By X

2- By range

1

Write X

4

Result:

f(4) = 34

f(4) = 90

1- add function

2- calc functions

2

How much function wou want calculate

1- One

2- Many

2

1- By X

2- By range

2

Write A

0

Write B

4

Write step

0.5

0 | 6

0.5 | 9.5

1 | 13

1.5 | 16.5

2 | 20

2.5 | 23.5

3 | 27

3.5 | 30.5

------

0 | 6

0.5 | 9.5

1 | 15

1.5 | 22.5

2 | 32

2.5 | 43.5

3 | 57

3.5 | 72.5

------

1- add function

2- calc functions

**Вывод:** В ходе лабораторной работы получили теоретические знания в области разработки классов, получили практические навыки реализации классов и отношений между ними.