Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.04 – «Программная инженерия»

Дисциплина: «Проектирование архитектуры программных систем»

Профиль: «Разработка программно-информационных систем»

Семестр 5

ОТЧЕТ

по лабораторной работе №2

Тема: «Паттерны проектирования»

Выполнил: студент группы РИС-22-1б

Поважный В.Е. \_\_\_\_\_\_\_\_\_\_\_

Проверил: доцент кафедры ИТАС

Викентьева О.Л. \_\_\_\_\_\_\_\_\_\_\_

Пермь, 2024

**Постановка задачи**

1. **Описание используемого паттерна**

**1.1 Название паттерна**

Паттерн "Стратегия" (Strategy)

**1.2 Контекст и условия применения**

**Стратегия** — это поведенческий паттерн проектирования, который определяет семейство алгоритмов, помещает их в отдельные классы и делает их взаимозаменяемыми. Это позволяет выбирать алгоритм поведения объекта во время выполнения.

**Основные свойства:**

1. Алгоритмы вынесены в отдельные классы.
2. Клиентский код не зависит от конкретной реализации алгоритма.
3. Стратегии взаимозаменяемы, их можно легко переключать.

**1.3 Решение**

В этой задаче нужно реализовать три метода численного решения уравнения f(x)=0f(x) = 0f(x)=0:

* **Метод бисекции.**
* **Метод хорд.**
* **Метод Ньютона.Диаграмма классов**

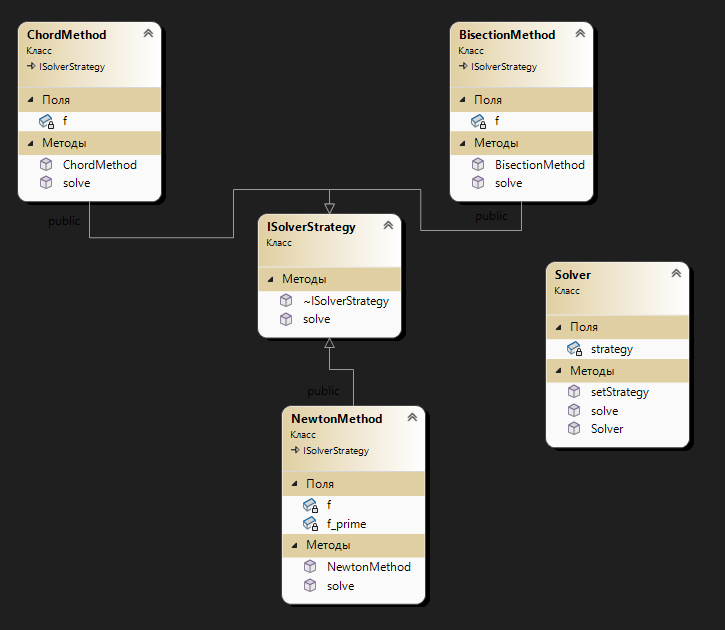


Рисунок 1 – Диаграмма классов

**Листинг**

#ifndef SOLVER\_H

#define SOLVER\_H

#include <iostream>

#include <cmath>

#include <functional>

class ISolverStrategy

{

public:

virtual double solve(double a, double b, double tolerance) = 0;

virtual ~ISolverStrategy() = default;

};

#endif

#ifndef SOLVERB\_H

#define SOLVERB\_H

#include "ISolverStrategy.h"

class BisectionMethod : public ISolverStrategy

{

private:

std::function<double(double)> f;

public:

BisectionMethod(std::function<double(double)> f) : f(f) {}

double solve(double a, double b, double tolerance) override

{

double mid;

while ((b - a) / 2 > tolerance)

{

mid = (a + b) / 2;

if (f(mid) == 0)

{

return mid;

}

else if (f(a) \* f(mid) < 0)

{

b = mid;

}

else

{

a = mid;

}

}

return (a + b) / 2;

}

};

#endif

#ifndef SOLVERC\_H

#define SOLVERC\_H

#include "ISolverStrategy.h"

class ChordMethod : public ISolverStrategy

{

private:

std::function<double(double)> f;

public:

ChordMethod(std::function<double(double)> f) : f(f) {}

double solve(double a, double b, double tolerance) override

{

double x = b;

while (std::abs(f(x)) > tolerance)

{

double fa = f(a);

double fb = f(b);

if (fb == fa)

throw std::runtime\_error("Division by zero in Chord Method.");

x = b - fb \* (b - a) / (fb - fa);

if (f(x) \* f(b) < 0)

a = b;

b = x;

}

return x;

}

};

#endif

#ifndef SOLVERN\_H

#define SOLVERN\_H

#include "ISolverStrategy.h"

class NewtonMethod : public ISolverStrategy

{

private:

std::function<double(double)> f;

std::function<double(double)> f\_prime;

public:

NewtonMethod(std::function<double(double)> f, std::function<double(double)> f\_prime)

: f(f), f\_prime(f\_prime) {}

double solve(double a, double b, double tolerance) override

{

double x = b;

while (std::abs(f(x)) > tolerance)

{

double derivative = f\_prime(x);

if (derivative == 0)

throw std::runtime\_error("Derivative is zero in Newton's Method.");

x = x - f(x) / derivative;

}

return x;

}

};

#endif

#ifndef SOLVERR\_H

#define SOLVERR\_H

#include "ISolverStrategy.h"

class Solver

{

private:

ISolverStrategy\* strategy;

public:

Solver(ISolverStrategy\* strategy) : strategy(strategy) {}

void setStrategy(ISolverStrategy\* newStrategy)

{

strategy = newStrategy;

}

double solve(double a, double b, double tolerance)

{

return strategy->solve(a, b, tolerance);

}

};

#endif

#include "BisectionMethod.h"

#include "ChordMethod.h"

#include "NewtonMethod.h"

#include "Solver.h"

int main()

{

setlocale(LC\_ALL, "rus");

auto f = [](double x) { return x \* x - 2; };

auto f\_prime = [](double x) { return 2 \* x; };

double a = 0, b = 2, tolerance = 1e-6;

try

{

// Метод бисекции

BisectionMethod bisection(f);

Solver solver(&bisection);

std::cout << "Метод бисекции: корень = " << solver.solve(a, b, tolerance) << std::endl;

// Метод хорд

ChordMethod chord(f);

solver.setStrategy(&chord);

std::cout << "Метод хорд: корень = " << solver.solve(a, b, tolerance) << std::endl;

// Метод Ньютона

NewtonMethod newton(f, f\_prime);

solver.setStrategy(&newton);

std::cout << "Метод Ньютона: корень = " << solver.solve(a, b, tolerance) << std::endl;

}

catch (const std::exception& e)

{

std::cerr << "Ошибка: " << e.what() << std::endl;

}

system("pause");

return 0;

}