МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«Пермский национальный исследовательский**

**политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЕТ

По лабораторной работе № 1

Дисциплина: «Проектирование архитектуры программных систем»

Выполнил:

студент 4 курса, гр. РИС-21-2б

Уржумов В.И.

Руководитель:

Кандидат технических наук

Викентьева О.Л.

Пермь, 2025

## Задача «Логгер с различным поведением (стратегия)», сложность 3

**1. Реализовать журналирование (класс логгер). Примерный интерфейс:**

**class** **LoggerStrategy** {

**public**:

**virtual** void write(**const** std::string &message) = 0;

};

**2. Реализовать 3 типа поведения (стратегий)**

* ConsoleLogger - выводит сообщение в консоль
* SimpleFileLogger - выводит сообщение в файл
* TimedFileLogger - выводит текущее время + сообщение в файл

**3. Реализовать класс, который будет использовать одну из стратегий**

**class** **Logger** {

**public**:

void set\_strategy(LoggerStrategy& strategy);

void log\_message(**const** std::string &message);

};

Паттерн «Стратегия» (Strategy) относится к поведенческим паттернам проектирования. Его суть заключается в том, чтобы выделить различные алгоритмы (или поведения) в отдельные классы, которые могут быть взаимозаменяемыми.

Это позволяет клиенту выбирать алгоритм в процессе выполнения программы, не изменяя структуру самого объекта. В результате мы получаем более гибкую и расширяемую архитектуру, где новые алгоритмы можно добавлять без изменений в существующий код.

РЕШЕНИЕ

Класс LogStrategy- абстрактный интерфейс для стратегий логирования. Данный чисто виртуальный класс определяет метод write (), принимающий контейнер string, который будет реализован в конкретных стратегиях.

class LoggerStrategy {

public:

virtual void write(const std::string& message) = 0;

};

Реализация стратегий

Стратегия ConsoleLogger реализует простой вывод сообщения в консоль

class ConsoleLogger : public LoggerStrategy

{

public:

void write(const std::string& message) override {

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

}

};

Стратегия SimpleFileLogger реализует простой вывод сообщения в текстовый файл

class SimpleFileLogger : public LoggerStrategy

{

void write(const std::string& message) override {

std::ofstream LogFile;

LogFile.open("Log.txt", std::ios::app);

if (LogFile.is\_open())

LogFile << message << "\n";

LogFile.close();

}

};

Стратегия TimedFileLogger реализует вывод сообщения и текущей даты/времени в текстовый файл

class TimedFileLogger : public LoggerStrategy

{

void write(const std::string& message) override {

const auto now = std::chrono::system\_clock::now();

const std::time\_t t\_c = std::chrono::system\_clock::to\_time\_t(now);

std::ofstream LogFile;

LogFile.open("Log.txt", std::ios::app);

if (LogFile.is\_open())

LogFile <<message << " DateTime: " << std::ctime(&t\_c) << "\n";

LogFile.close();

}

};

Класс LoggerContext хранит указатель на выбранную стратегию логирования и использует его для нужного типа вывода сообщения.

class LogContext {

private:

LoggerStrategy\* strategy;

public:

void setStrategy(LoggerStrategy\* strategy) {

this->strategy = strategy;

}

void executeStrategy(std::string& message) {

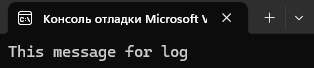
strategy->write(message);

}

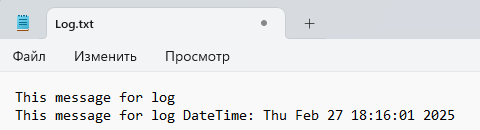
};

Метод setStartegy- позволяет менять стратегию на необходимую. Метод executeStartegy- вызывает вывод сообщения, делегирую это выбранной стратегии.

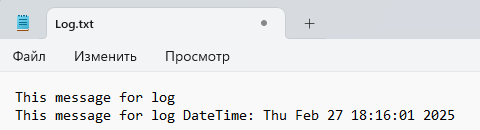
Результаты



Сообщение выводится в консоль



Сообщение выводится в текстовый файл



Сообщение и текущая дата/время выводится в текстовый файл

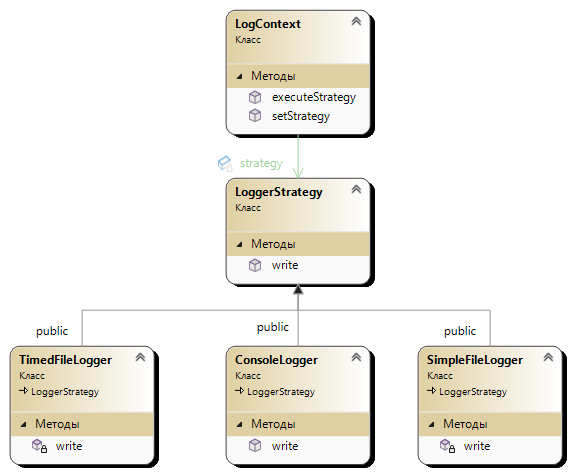


Диаграмма классов

Листинг

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include<fstream>

#include<chrono>

class LoggerStrategy {

public:

virtual void write(const std::string& message) = 0;

};

class LogContext {

private:

LoggerStrategy\* strategy;

public:

void setStrategy(LoggerStrategy\* strategy) {

this->strategy = strategy;

}

void executeStrategy(std::string& message) {

strategy->write(message);

}

};

class ConsoleLogger : public LoggerStrategy

{

public:

void write(const std::string& message) override {

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

}

};

class SimpleFileLogger : public LoggerStrategy

{

void write(const std::string& message) override {

std::ofstream LogFile;

LogFile.open("Log.txt", std::ios::app);

if (LogFile.is\_open())

LogFile << message << "\n";

LogFile.close();

}

};

class TimedFileLogger : public LoggerStrategy

{

void write(const std::string& message) override {

const auto now = std::chrono::system\_clock::now();

const std::time\_t t\_c = std::chrono::system\_clock::to\_time\_t(now);

std::ofstream LogFile;

LogFile.open("Log.txt", std::ios::app);

if (LogFile.is\_open())

LogFile <<message << " DateTime: " << std::ctime(&t\_c) << "\n";

LogFile.close();

}

};

int main()

{

std::string message = "This message for log";

LogContext context;

ConsoleLogger logConsole;

SimpleFileLogger logSimpleFile;

TimedFileLogger logTimedFile;

context.setStrategy(&logConsole);

context.executeStrategy(message);

context.setStrategy(&logSimpleFile);

context.executeStrategy(message);

context.setStrategy(&logTimedFile);

context.executeStrategy(message);

return 0;

}