МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І. І. МЕЧНИКОВА
ФАКУЛЬТЕТ МАТЕМАТИКИ, ФІЗИКИ ТА ІНФОРМАЦІЙНИЇ ТЕХНОЛОГІЙ
КАФЕДРА МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

Лабораторна робота №10

з дисципліни: «Інженерія програмного забезпечення»

за темою: «Патерн проектування Proxy»

Виконав:
студент 3 курсу
Владислав Краковський

Перевірив:
викладач
Пенко В.Г.

Одеса 2025

Зміст

1 ЗАВДАННЯ РОБОТИ

Мета роботи: Ознайомитися з патерном проєктування “Проксі” (Proxy), навчитися реалізовувати його на мові програмування C++, зрозуміти сфери застосування, основні переваги й недоліки цього патерну.

Умова роботи:

Приклад - обчислення чисел Фібоначчі.

Традиційний спосіб обчислення чисел Фібоначчі – рекурсія, що ґрунтується на математичному визначенні чисел Фібоначчі. Однак, якщо клієнт запросив число Фібоначчі, яке вже обчислювалося раніше, його можна просто запам'ятати та видавати на вимогу клієнта.

Рисунок 1.1 - Умова

Варіанти самостійних завдань

1. Модифікуйте попередній додаток, щоб він показував витрати часу при отриманні нових чисел Фібоначчі та раніше запам'ятаних.

Рисунок 1.2 - Самостійне завдання

Система контролю версій Git та рефакторинг

Варіанти самостійних завдань

У наведеному вище викладі кожного патерну наводяться структурний та реальний приклад. Реалізувати структурний приклад та зафіксувати його в системі контролю версій. Далі здійснити серію фіксацій, яка трансформує структурний приклад у реальний. Деякі з проміжних фіксацій повинні полягати у застосуванні того чи іншого прийому рефакторингу.

Рисунок 1.3 - Завдання з GitHub

2 ХІД РОБОТИ

Опис патерну Proxy:

Proxy (Проксі, Замісник) — це структурний патерн проєктування, який дозволяє створити об’єкт-замісник (проксі), що контролює доступ до іншого об’єкта.
Проксі обгортає реальний об’єкт і “перехоплює” всі звернення до нього, надаючи додаткову логіку: кешування, захист, відкладене створення, ведення журналу тощо.

Як реалізується патерн Proxy?

1. Створюється загальний інтерфейс для реального об’єкта та проксі (наприклад, Subject).
2. Реальний об’єкт (RealSubject) реалізує цей інтерфейс і містить основну логіку.
3. Клас Proxy також реалізує інтерфейс, але містить посилання на реальний об’єкт і додає власну поведінку (перевірка прав, кеш, логування, ліниве створення…).
4. Клієнт взаємодіє лише з інтерфейсом — не розрізняючи, працює він із проксі чи з реальним об’єктом.

У C++ це означає створення базового інтерфейсу, двох похідних класів (реальний і проксі), і взаємодію клієнта тільки через інтерфейс.

Для чого використовується патерн Proxy?

- Для контролю доступу до об’єкта (авторизація, права доступу).
- Для організації кешування результатів, які дорого обчислювати.
- Для оптимізації роботи з “важкими” об’єктами (відкладене створення).
- Для ведення журналу доступу, статистики.
- Для розподілених систем (віддалений проксі).

Типові приклади:

- Віртуальні проксі для завантаження великих зображень.
- “Розумні” вказівники (smart pointers) у C++.
- Мережевий проксі для передачі запитів у віддалені системи.

Переваги патерну Proxy

- Гнучкість: додатковий рівень контролю, без зміни коду реального об’єкта.
- Зниження навантаження (наприклад, через кешування або відкладене створення).
- Безпека: можна організувати перевірку прав доступу.
- Прозорість для клієнта: клієнт працює з інтерфейсом, не знаючи про проксі.

Недоліки патерну Proxy

− Ускладнення архітектури: додається ще один клас і рівень абстракції.
− Можливе зниження продуктивності через додаткові перевірки або перенаправлення викликів.
− Необхідність синхронізації (для потокобезпечності у складних випадках).

Запишемо розв’язок поставленої задачі з використання патерну Proxy:

```
#include <iostream>
#include <vector>
#include <stdexcept>
#include <chrono>

// Абстрактный интерфейс
class Fibonacci {
public:
    virtual ~Fibonacci() {}
    virtual int GetFibonacci(int n) = 0;
};

// Класс-реализация с кэшем
class RetentiveFibonacci : public Fibonacci {
    std::vector<int> fibNumbers;
public:
    RetentiveFibonacci() {
        fibNumbers.push_back(1);
        fibNumbers.push_back(1);
    }
    int MaxIndex() const { return
static_cast<int>(fibNumbers.size()); }
    int Last() const { return fibNumbers.back(); }
    int Penultimate() const { return fibNumbers[fibNumbers.size()
- 2]; }

    void Add(const std::vector<int>& appendix) {
        fibNumbers.insert(fibNumbers.end(), appendix.begin(),
appendix.end());
    }
    int GetFibonacci(int n) override {
        if (n <= MaxIndex())
```

```
            return fibNumbers[n - 1];
        throw std::runtime_error("This Fibonacci number was not
yet calculated.");
    }
};

// Proxy (реальный объект с расширением)
class RealFibonacci : public Fibonacci {
    RetentiveFibonacci retentiveFibonacci;
public:
    int GetFibonacci(int n) override {
        if (n > retentiveFibonacci.MaxIndex()) {
            std::vector<int> appendix;
            appendix.push_back(retentiveFibonacci.Penultimate());
            appendix.push_back(retentiveFibonacci.Last());
            for (int i = retentiveFibonacci.MaxIndex(); i < n; +
+i)
                appendix.push_back(appendix[appendix.size() - 2] +
appendix[appendix.size() - 1]);
            // Удаляем первые два вспомогательных элемента
            appendix.erase(appendix.begin(), appendix.begin() +
2);
            retentiveFibonacci.Add(appendix);
        }
        return retentiveFibonacci.GetFibonacci(n);
    }
};

int main() {
    RealFibonacci proxy;
    // Тест: несколько чисел, среди них есть повторное вычисление
(уже закэшировано)
    int testValues[] = { 5, 20, 10, 20 };
    for (int n : testValues) {
```

```
        auto start = std::chrono::high_resolution_clock::now();
        int result = proxy.GetFibonacci(n);
        auto end = std::chrono::high_resolution_clock::now();
        std::chrono::duration<double, std::micro> elapsed = end -
start;
        std::cout << "Fibonacci(" << n << ") = " << result
            << " | Time: " << elapsed.count() << " microseconds"
<< std::endl;
    }
    return 0;
}
```

Подивимось скріншот з результатом на ілюстрації 2.1:

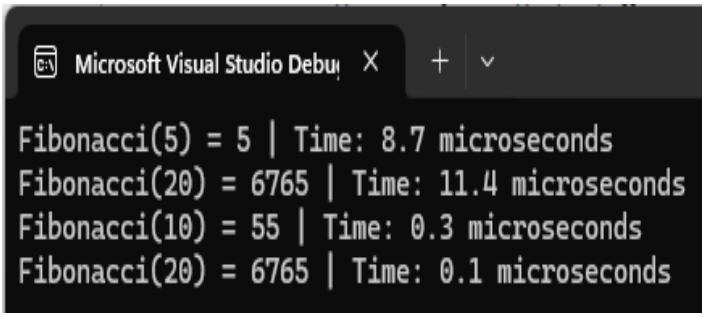

Рисунок 2.1 — Результат роботи програми

Програма була завантажена на GitHub згідно з завданням, ілюстрація 2.2:

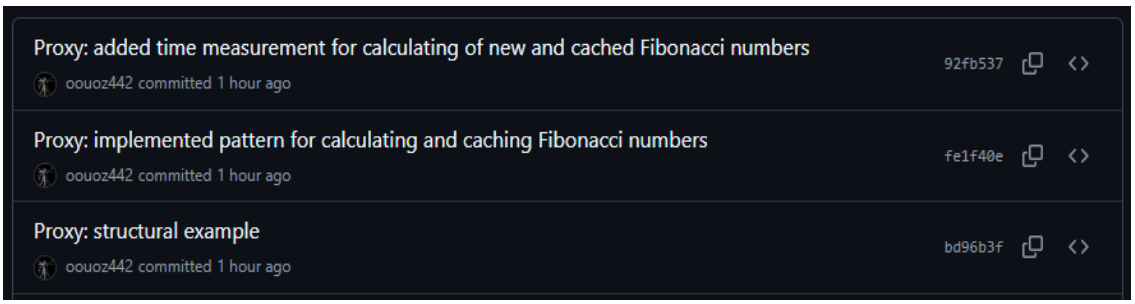

Рисунок 2.2 - GitHub

Висновок

Патерн “Proxy” — це ефективний інструмент для контролю доступу, кешування, оптимізації й розширення поведінки об’єктів без зміни їхнього коду. Він підвищує гнучкість і розширюваність систем, але вимагає уважності до зростання складності архітектури та можливої появи додаткових витрат на продуктивність. Proxy доцільно застосовувати там, де необхідний контроль або оптимізація взаємодії з об’єктами.