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 №6

з дисципліни: «Інженерія програмного забезпечення»

за темою: «Патерн проектування Adapter»

Виконав:
студент 3 курсу
Владислав Краковський

Перевірив:
викладач
Пенко В.Г.

Одеса 2025

Зміст

1 ЗАВДАННЯ РОБОТИ

Мета роботи: Ознайомитися з патерном проєктування “Адаптер” (Adapter), навчитися реалізовувати його на мові програмування C++, зрозуміти основне призначення, переваги й недоліки цього патерну, а також сфери його практичного застосування.

Умова роботи:

Приклад – зрізи в C#.

У C# в класі String є метод Substring з параметрами start (початок підрядка) та length (довжина підрядка). Але досить часто зручно виділяти піврядки, вказавши номер початкового та кінцевого символу у вихідному рядку. Реалізувати цю модифіковану функціональність за допомогою патерна Adagter.

Рисунок 1.1 - Умова

Варіанти самостійних завдань

1. Модифікуйте попередній додаток, щоб можна було витягати з рядка «перевернуті» підрядки.

Рисунок 1.2 - Варіант задачі

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 Adapter:

Adapter (Адаптер) — це структурний патерн проєктування, який дозволяє об’єктам з несумісними інтерфейсами працювати разом.
Патерн обгортає один клас в інший спеціальний клас (адаптер), який надає потрібний клієнту інтерфейс і переводить (адаптує) виклики з одного інтерфейсу до іншого.

Як реалізується патерн Adapter?

1. Створюється інтерфейс, який потрібен клієнту.
2. Створюється адаптер, який реалізує цей інтерфейс та містить посилання на адаптований (існуючий) об’єкт.
3. Адаптер перетворює виклики клієнта до відповідних викликів адаптованого об’єкта, якщо потрібно — з перетворенням даних.
4. Усі звернення клієнта йдуть через адаптер, не напряму до несумісного класу.

У C++ це означає створення класу-адаптера, який містить як публічні методи інтерфейс клієнта, так і приватне посилання на існуючий об’єкт.

Для чого використовується патерн Adapter?

- Коли потрібно використовувати існуючий клас, але його інтерфейс не відповідає вимогам клієнта.

- Для інтеграції стороннього коду (наприклад, бібліотеки) у власну систему без переписування коду.
- При поетапному переході на новий інтерфейс без зміни існуючих класів.

Типові приклади:

- Драйвери пристроїв, які забезпечують однаковий інтерфейс для різних апаратних платформ.
- Використання старих бібліотек у нових програмах (legacy code).
- Обгортання API бібліотек для підходу “своїм” інтерфейсом.

Переваги патерну Adapter

- Можливість використання існуючого коду без змін.
- Гнучкість інтеграції різних компонентів (навіть якщо їхні інтерфейси відрізняються).
- Дотримання принципу єдиного обов’язку (Single Responsibility Principle) — адаптація винесена в окремий клас.

Недоліки патерну Adapter

− Додає новий шар абстракції, що може ускладнити розуміння коду.
− Може знизити продуктивність через додаткові виклики (не критично для більшості застосувань).
− Не вирішує глибинних конфліктів логіки — лише інтерфейсну несумісність.

Запишемо розв’язок поставленої задачі з використання патерну Adapter:

```
#include <iostream>
#include <string>
#include <memory>
using namespace std;

// === Целевой интерфейс ===
class Sliceable {
public:
    virtual string Slice(int start, int finish) const = 0;
    virtual ~Sliceable() = default;
};

// === Адаптируемый класс ===
class MyString {
private:
    string innerString;

public:
    MyString(const string& s) : innerString(s) {}

    string Substring(int start, int length) const {
        return innerString.substr(start, length);
    }

    int Length() const {
        return innerString.length();
    }
};

// === Адаптер ===
```

```
class StringAdapter : public Sliceable {
private:
    shared_ptr<MyString> adaptee;

public:
    StringAdapter(shared_ptr<MyString> ms) : adaptee(ms) {}

    string Slice(int start, int finish) const override {
        if ((start >= 0) && (finish < adaptee->Length())) {
            if (start <= finish) {
                return adaptee->Substring(start, finish - start +
1);
            }
            else {
                // Перевернутая подстрока
                string reversed;
                for (int i = start; i >= finish; --i) {
                    reversed += adaptee->Substring(i, 1);
                }
                return reversed;
            }
        }
        else {
            throw runtime_error("Illegal call of Slice method");
        }
    }
};

// === main ===
int main() {
    string s = "Hello, World!";
    shared_ptr<MyString> ms = make_shared<MyString>(s);
```

```
    shared_ptr<Sliceable> adapter =
make_shared<StringAdapter>(ms);


    cout << "Normal:   " << adapter->Slice(2, 8) << endl;    //
llo, Wo
    cout << "Reversed: " << adapter->Slice(8, 2) << endl;    //
oW ,oll


    return 0;
}
```

Подивимось скріншот з результатом на ілюстрації 2.1:

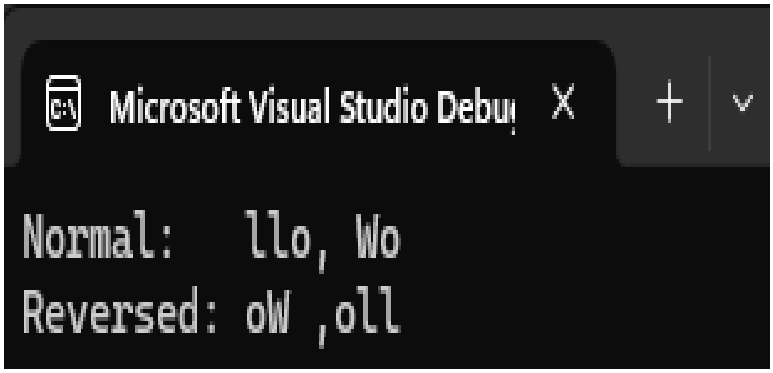

Рисунок 2.1 — Результат роботи програми

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

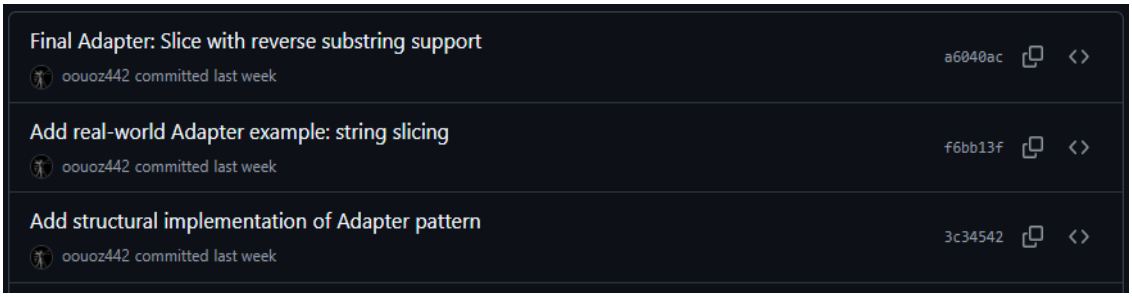

Рисунок 2.2 - GitHub

Висновок

Патерн “Adapter” є корисним для інтеграції сторонніх і застарілих компонентів у нові системи без зміни їхнього коду. Він забезпечує сумісність між різними інтерфейсами, підвищуючи гнучкість і підтримуваність програмного забезпечення. Проте Adapter додає нову абстракцію і може ускладнити структуру проекту, тому застосовувати його варто тоді, коли реально існує потреба у “стикуванні” різних інтерфейсів.