

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

**Лабораторна робота №2**  
**з дисципліни: «Інженерія програмного забезпечення»**  
**за темою: «Патерн проектування Observer»**

**Виконав:**  
студент 3 курсу  
Владислав Краковський

**Перевірив:**  
викладач  
Пенко В.Г.

## **Зміст**

1 ЗАВДАННЯ РОБОТИ.....	3
2 ХІД РОБОТИ.....	4
Висновок.....	11

## 1 ЗАВДАННЯ РОБОТИ

**Мета роботи:** Ознайомитися з патерном проєктування “Спостерігач” (Observer), навчитись реалізовувати його на мові програмування C++, зрозуміти сфери застосування цього патерну, а також його основні переваги й недоліки.

### Умова роботи:

#### Приклад – друкуємо, перевіряємо та перекладаємо текст

Промодельюємо з допомогою патерну Observer процес, у якому циклічно виробляється отримання чергового слова. За це відповідає клас WordWriter, що грає роль Subject. Зацікавлені у цьому процесі спостерігачі – перекладачі одержуваних слів різними мовами та коректором слів. При виникненні нового слова вони автоматично виконують переклад та коригування.

*Рисунок 1.1 - Умова*

#### Варіанти самостійних завдань

1. Введіть у функціональність перекладача виправлення на основі відстані Дамерау-Левенштейна.

*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 Observer:

Патерн Observer (Спостерігач) — це поведінковий патерн проектування, який визначає залежність типу “один-до-багатьох” між об’єктами так, що при зміні стану одного об’єкта всі залежні від нього об’єкти сповіщаються та автоматично оновлюються.

Цей патерн ще часто називають "Підписка-сповіщення" (Publish-Subscribe).

### Як реалізується патерн Observer?

1. Створюється інтерфейс для “спостерігачів” (Observer), який містить, як правило, один метод для отримання сповіщення про зміну.
2. Створюється клас “суб’єкта” (Subject), який зберігає список підписаних спостерігачів і надає методи для підписки/відписки.
3. Коли стан суб’єкта змінюється — він викликає метод сповіщення для всіх своїх спостерігачів.
4. Спостерігачі реалізують метод реакції на сповіщення, який оновлює їхній стан згідно з новими даними суб’єкта.

*У C++ це зазвичай означає наявність абстрактного класу Observer, класу Subject зі списком Observer, та реалізації цих інтерфейсів у власних класах.\**

## Для чого використовується патерн Observer?

- Коли треба автоматично повідомляти багатьом об'єктам про зміну стану одного об'єкта.
- Для організації “реактивних” систем, де зміна однієї частини системи має викликати зміну в інших частинах.
- Часто використовується у графічних інтерфейсах (GUI), у системах обробки подій, у моделюванні тощо.

Типові приклади:

- Оновлення віджетів інтерфейсу при зміні даних.
- Підписка на оновлення даних у месенджерах, новинах.
- Реалізація “живої” стрічки повідомлень.

## Переваги патерну Observer

- Ослаблене зв'язування між об'єктами: суб'єкт не знає деталей реалізації спостерігачів.
- Автоматична синхронізація стану: усі спостерігачі отримують актуальні дані.
- Легке додавання нових спостерігачів без зміни коду суб'єкта.

## Недоліки патерну Observer

- Можливість “ланцюгових” оновлень: якщо об'єкти взаємно спостерігають один за одним, можна отримати нескінченний цикл оновлень.
- Важко відстежувати залежності у великих системах — що від чого залежить.
- Потенційна витрата ресурсів: велика кількість спостерігачів може призвести до затримок при сповіщеннях.

Запишемо розв'язок поставленої задачі з використання патерну Observer:

```
#include <iostream>
#include <vector>
#include <string>
#include <memory>
#include <algorithm>
#include <windows.h>

using namespace std;

// === Інтерфейс Observer ===
class Observer {
public:
    virtual void Update(const string& word) = 0;
    virtual ~Observer() = default;
};

// === Інтерфейс Subject ===
class Subject {
public:
    virtual void Attach(shared_ptr<Observer> observer) = 0;
    virtual void Detach(shared_ptr<Observer> observer) = 0;
    virtual void Notify(const string& word) = 0;
    virtual ~Subject() = default;
};

// === Конкретний Subject: WordWriter ===
class WordWriter : public Subject {
private:
    vector<shared_ptr<Observer>> observers;

public:
```

```

void Attach(shared_ptr<Observer> observer) override {
    observers.push_back(observer);
}

void Detach(shared_ptr<Observer> observer) override {
    observers.erase(remove(observers.begin(), observers.end(),
observer), observers.end());
}

void Notify(const string& word) override {
    for (auto& obs : observers) {
        obs->Update(word);
    }
}

void TypeWord(const string& word) {
    cout << "\nTyping word: " << word << endl;
    Notify(word);
}
};

// === Переводчики (Observers) ===
class EngToUkrTranslator : public Observer {
public:
    void Update(const string& word) override {
        cout << "[EN→UA] Translation of '" << word << "' →
'Переклад'" << endl;
    }
};

class EngToRusTranslator : public Observer {
public:
    void Update(const string& word) override {

```

```

        cout << "[EN→RU] Translation of '" << word << "' →
'Перевод'" << endl;
    }
};

class CorrectorWithFix : public Observer {
private:
    vector<string> dictionary = { "hello", "world", "test", "the",
"word" };

    int DamerauLevenshtein(const string& s1, const string& s2) {
        int len1 = s1.size();
        int len2 = s2.size();
        vector<vector<int>> d(len1 + 1, vector<int>(len2 + 1));

        for (int i = 0; i <= len1; ++i) d[i][0] = i;
        for (int j = 0; j <= len2; ++j) d[0][j] = j;

        for (int i = 1; i <= len1; ++i) {
            for (int j = 1; j <= len2; ++j) {
                int cost = (s1[i - 1] == s2[j - 1]) ? 0 : 1;

                int del = d[i - 1][j] + 1;          // удаление
                int ins = d[i][j - 1] + 1;          // вставка
                int rep = d[i - 1][j - 1] + cost;    // замена

                d[i][j] = min(min(del, ins), rep);

                if (i > 1 && j > 1 &&
                    s1[i - 1] == s2[j - 2] &&
                    s1[i - 2] == s2[j - 1]) {
                    d[i][j] = min(d[i][j], d[i - 2][j - 2] +
cost); // транспозиция
                }
            }
        }
    }
};

```



```

        }
    }
}

return d[len1][len2];
}

public:
    void Update(const string& word) override {
        cout << "[Corrector] Checking word: " << word << endl;

        for (const string& correct : dictionary) {
            int dist = DamerauLevenshtein(word, correct);
            if (dist <= 1 && word != correct) {
                cout << "    Did you mean: '" << correct << "'?
(distance = " << dist << ")\n";
                return;
            }
        }

        cout << "    ✓ Word seems correct.\n";
    }
};

// === Пример ===
int main() {
    setlocale(LC_ALL, "");
    SetConsoleOutputCP(CP_UTF8);
    auto writer = make_shared<WordWriter>();

    auto uaTranslator = make_shared<EngToUkrTranslator>();
    auto ruTranslator = make_shared<EngToRusTranslator>();
    auto corrector = make_shared<CorrectorWithFix>();

```

```

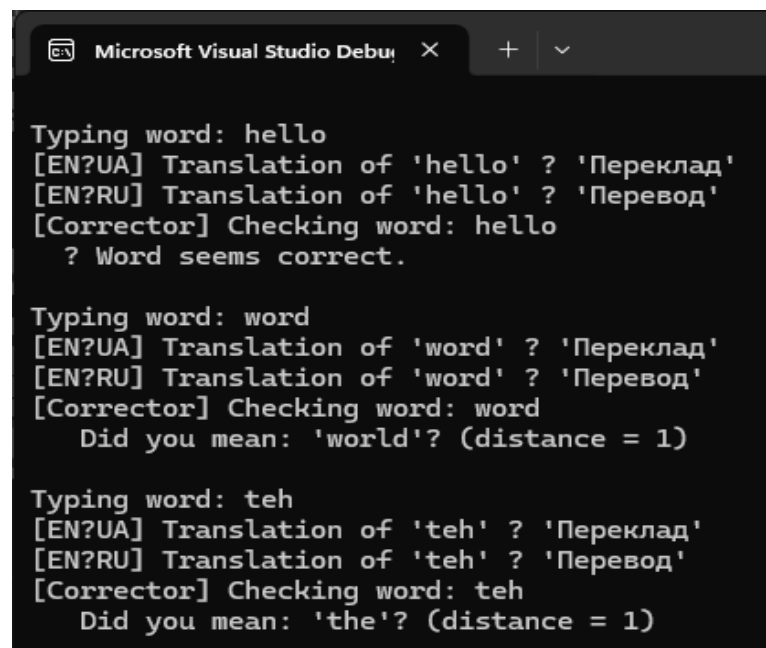
writer->Attach(uaTranslator);
writer->Attach(ruTranslator);
writer->Attach(corrector);

writer->TypeWord("hello");
writer->TypeWord("word");
writer->TypeWord("teh");

return 0;
}

```

Подивимось скріншот з результатом на ілюстрації 2.1:



```

Microsoft Visual Studio Debug Console
Typing word: hello
[EN?UA] Translation of 'hello' ? 'Переклад'
[EN?RU] Translation of 'hello' ? 'Перевод'
[Corrector] Checking word: hello
? Word seems correct.

Typing word: word
[EN?UA] Translation of 'word' ? 'Переклад'
[EN?RU] Translation of 'word' ? 'Перевод'
[Corrector] Checking word: word
Did you mean: 'world'? (distance = 1)

Typing word: teh
[EN?UA] Translation of 'teh' ? 'Переклад'
[EN?RU] Translation of 'teh' ? 'Перевод'
[Corrector] Checking word: teh
Did you mean: 'the'? (distance = 1)

```

Рисунок 2.1 — Результат роботи програми

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

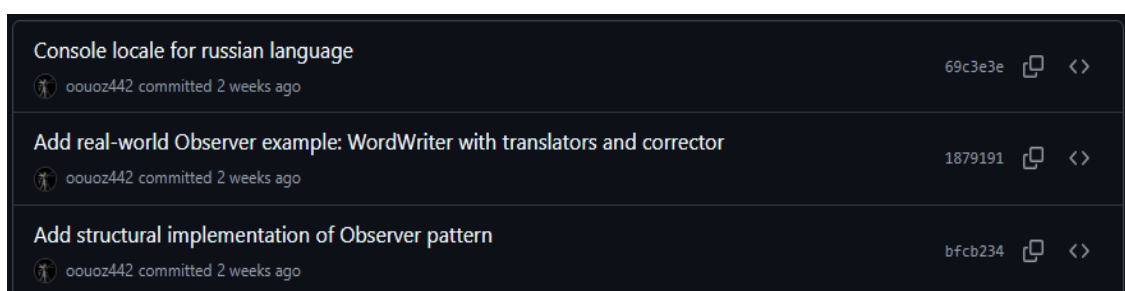


Рисунок 2.2 - GitHub

## **Висновок**

Патерн “Observer” — це потужний інструмент для організації зв’язаних змін між об’єктами у програмному забезпеченні. Він дозволяє автоматично повідомляти зацікавлені об’єкти про зміни стану, підвищує модульність і зменшує жорстке зв’язування між компонентами. Проте застосування Observer вимагає уважності до уникнення циклічних залежностей та контролю над числом підписників.