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 №4

з дисципліни: «Інженерія програмного забезпечення»

за темою: «Патерн проектування Factory Method»

Виконав:
студент 3 курсу
Владислав Краковський

Перевірив:
викладач
Пенко В.Г.

Одеса 2025

Зміст

1 ЗАВДАННЯ РОБОТИ

Мета роботи: Ознайомитися з патерном проєктування “Фабричний метод” (Factory Method), навчитися його реалізовувати мовою програмування C++, зрозуміти, де він застосовується, а також розглянути його переваги й недоліки.

Умова роботи:

Реалістичний приклад

Конкретний приклад демонструє Factory Method, що забезпечує гнучкість при створенні різних документів. Похідні класи Report(Звіт) та Resume(Резюме) створюють розширені версії об'єкта Document(Документ). Тут Factory Method викликається у конструкторі базового класу Document.

Рисунок 1.1 - Умова

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 1.2 - Завдання з GitHub

2 ХІД РОБОТИ

Опис патерну Decorator:

Factory Method (Фабричний метод) — це породжуючий патерн проєктування, який визначає загальний інтерфейс для створення об’єктів у суперкласі, дозволяючи підкласам змінювати тип створюваного об’єкта. Інакше кажучи, цей патерн делегує створення об’єктів дочірнім класам.

Як реалізується патерн Factory Method?

1. Створюється абстрактний базовий клас або інтерфейс з фабричним методом (наприклад, CreateProduct()).
2. Для кожного конкретного типу продукту створюється власний клас, що реалізує цей інтерфейс.
3. Для кожної конкретної фабрики створюється підклас, який перевизначає фабричний метод і повертає об'єкт потрібного типу.
4. Клієнт працює тільки з базовим інтерфейсом фабрики, не знаючи наперед, який саме клас буде створено.

У C++ це означає створення ієрархії базового продукту, підкласів-продуктів і фабрик, що їх породжують.

Для чого використовується патерн Factory Method?

- Коли потрібно делегувати створення об’єктів підкласам.
- Коли код повинен працювати з об'єктами певного інтерфейсу, але конкретний клас заздалегідь невідомий.

- Коли необхідно уникнути жорсткої прив’язки коду до конкретних класів (зменшення залежностей).

Типові приклади:

- Створення різних документів у додатку (наприклад, Word, Excel, PDF).
- Підключення до різних баз даних.
- Відображення різних UI-елементів для різних платформ.

Переваги патерну Factory Method

- Послаблення зв’язку між класами: клієнт не залежить від конкретних класів продуктів.
- Гнучкість розширення: можна легко додавати нові типи продуктів, не змінюючи клієнтський код.
- Інкапсуляція створення об’єктів: централізовано керує створенням екземплярів.

Недоліки патерну Factory Method

− Ускладнення структури проекту: потрібно створювати додаткові класи для кожного продукту й фабрики.
− Початкове налаштування може займати більше часу, ніж у випадку прямого створення об’єктів.
− Може призвести до надмірної кількості класів у великих проектах.

Запишемо розв’язок поставленої задачі з використання патерну Factory Method:

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

// === Продукт ===
class Document {
public:
    virtual void Open() const = 0;
    virtual ~Document() = default;
};

// === Конкретные документы ===
class MyDoc : public Document {
public:
    void Open() const override {
        cout << "MyDoc opened\n";
    }
};

class ReportDoc : public Document {
public:
    void Open() const override {
        cout << "ReportDoc opened\n";
    }
};

// === Создатель ===
class Application {
protected:
```

```
    vector<shared_ptr<Document>> docs;

public:
    void NewDocument() {
        auto doc = CreateDocument();
        docs.push_back(doc);
        doc->Open();
    }

    virtual shared_ptr<Document> CreateDocument() const = 0;

    virtual ~Application() = default;
};

// === Конкретные приложения ===
class MyApp : public Application {
public:
    shared_ptr<Document> CreateDocument() const override {
        return make_shared<MyDoc>();
    }
};

class ReportApp : public Application {
public:
    shared_ptr<Document> CreateDocument() const override {
        return make_shared<ReportDoc>();
    }
};

// === main() ===
int main() {
    shared_ptr<Application> app1 = make_shared<MyApp>();
```

```
    app1->NewDocument();  // MyDoc opened

    shared_ptr<Application> app2 = make_shared<ReportApp>();
    app2->NewDocument();  // ReportDoc opened

    return 0;
}
```

Подивимось скріншот з результатом на ілюстрації 2.1:

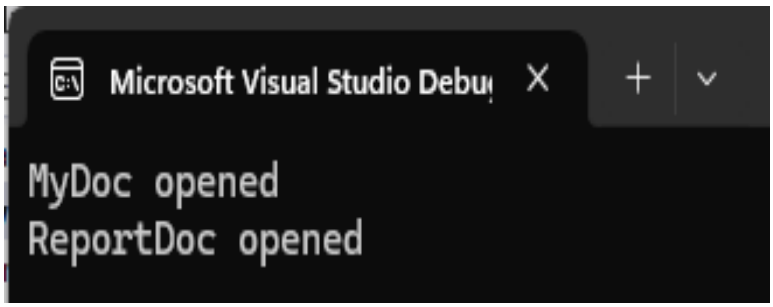

Рисунок 2.1 — Результат роботи програми

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

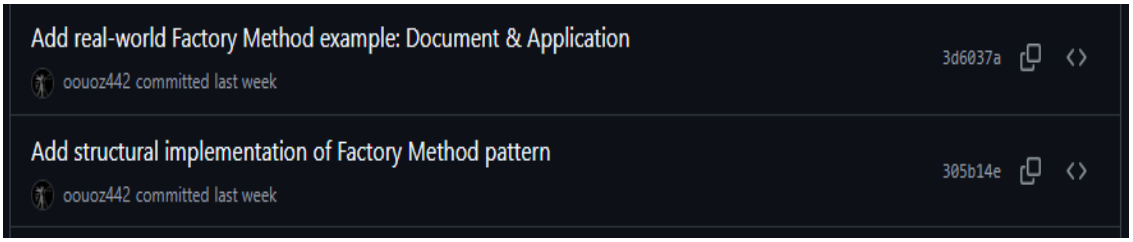

Рисунок 2.2 - GitHub

Висновок

Патерн “Factory Method” дозволяє централізовано і гнучко управляти створенням об’єктів у програмних системах, полегшує розширюваність і модифікацію коду. Його варто застосовувати у випадках, коли клас не може знати наперед, який саме тип об’єкта буде потрібен, або коли створення об’єктів має бути делеговано підкласам. Проте слід враховувати, що Factory Method збільшує кількість класів і може ускладнювати структуру проекту.