Міністерство освіти і науки України
Національний технічний університет України “Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки Кафедра інформаційних систем та технологій

Лабораторна робота №6

Технології розроблення програмного забезпечення

Тема: «Патерни проектування.»

Виконала:
Студентка групи ІА-34
Сизоненко А.О
Перевірив:
Мягкий Михайло Юрійович

Київ-2025

Мета: Вивчити структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчитися застосовувати їх в реалізації програмної системи.

Тема роботи: Текстовий редактор (strategy, command, observer, template method, flyweight, SOA)

Текстовий редактор повинен вміти розпізнавати текстові файли в будь-якій кодуванні, мати розширені функції редагування: макроси, сніппети, підказки, закладки, перехід на рядок / сторінку, підсвічування синтаксису (для однієї мови програмування або розмітки на розсуд студента).

Теоретичні відомості

Abstract-Factory

Дозволяє створювати сімейства взаємопов’язаних об’єктів без вказівки їх конкретних класів. Наприклад, у генерації кімнат для гри створюються стіни, двері, підлога і меблі одного стилю. Переваги: об’єкти одного стилю узгоджені, код структурований, легко додавати нові стилі. Недоліки: складність коду, додавання нового типу продукту потребує змін у багатьох місцях.

Factory-Method

Визначає інтерфейс для створення об’єктів базового типу, дозволяючи підставляти власні підкласи без зміни базової поведінки. Наприклад, створення TcpPacket та UdpPacket через відповідні фабрики. Переваги: відсутність прив’язки до конкретних класів, централізоване створення продуктів, легке додавання нових продуктів. Недоліки: може створювати велику кількість ієрархій класів.

Memento

Дозволяє зберігати і відновлювати стан об’єкта без порушення інкапсуляції. Часто використовується разом з патерном Command для реалізації undo. Переваги: не порушує інкапсуляцію, спрощує структуру об’єкта. Недоліки: потребує багато пам’яті при частому створенні знімків, необхідно контролювати звільнення застарілих станів.

Observer

Призначення: Шаблон визначає залежність «один-до-багатьох» таким чином, що коли один об'єкт змінює власний стан, усі інші об'єкти отримують про це сповіщення і мають можливість змінити власний стан також.

Цей шаблон дуже широко поширений в шаблоні MVVM і механізмі «прив'язок» (bindings) в WPF і частково в WinForms. Інша назва шаблону – підписка/розсилка. Кожен з оглядачів власноручно підписується на зміни конкретного об'єкту, а об'єкти зобов'язані сповіщати своїх передплатників про усі свої зміни (на даний момент конкретних механізмів автоматичного сповіщення про зміну стану в .NET мовах не існує).

Переваги та недоліки:

+ Можливість паралельної та асинхронної обробки повідомлень про оновлення.

+ Спостерігачів можна добавляти та видаляти в будь-який момент часу.

+ Спостерігач і суб’єкт можуть працювати в різних потоках.

+ Реалізує принцип слабкого зв’язку між об’єктами.

- Послідовність розсилки повідомлень підписникам не підтримується.

Decorator

Дозволяє динамічно додавати функціональні можливості об’єкту без зміни його коду, обертаючи його в новий клас. Наприклад, додавання смуги прокрутки до візуальних елементів UI. Переваги: декомпозиція на дрібні об’єкти, динамічне додавання обов’язків, гнучкість. Недоліки: велика кількість дрібних класів, складно конфігурувати об’єкти з кількома обгортками одночасно.

Хід роботи

1. Діаграма класів для реалізації шаблону спостерігач “observer”.

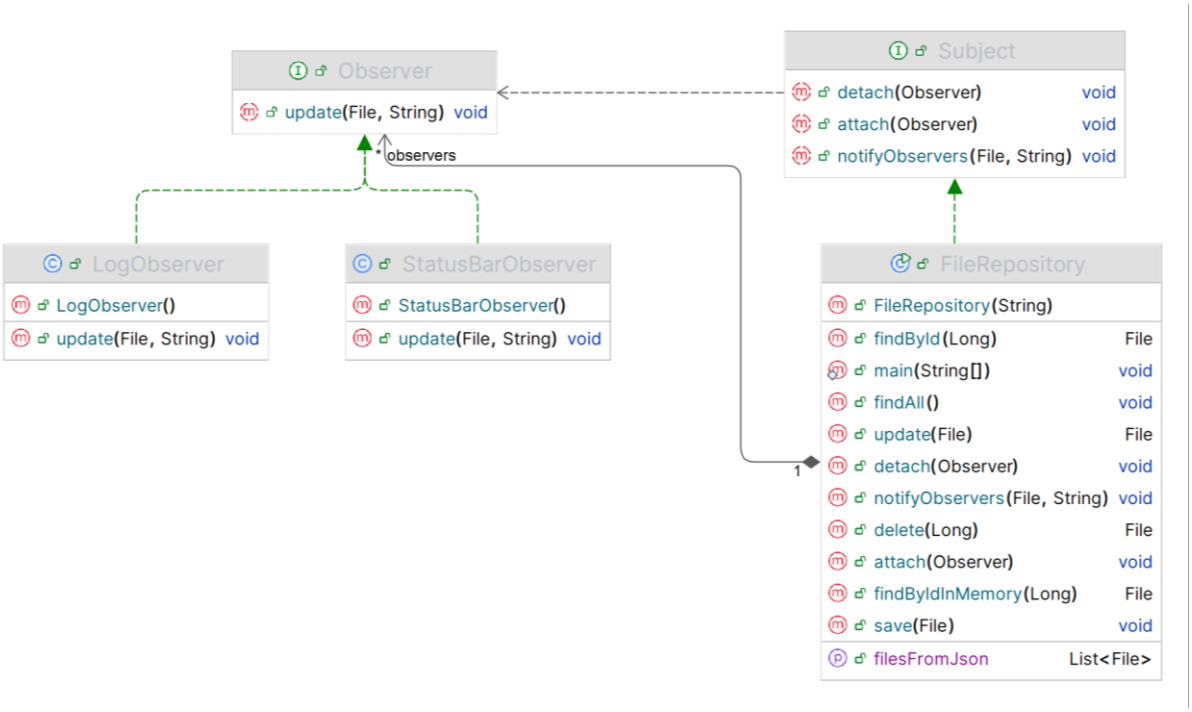


Рисунок 1 – Діаграма класів

Опис діаграми класів

1. Subject – це «підписник», який зберігає список Observer’ів і повідомляє їх про зміни. Має в собі методи:
   attach(Observer) — підписка нового Observer.
   detach(Observer) — відписка Observer.
   notifyObservers(File, String) — повідомлення всіх підписників про зміну файлу (наприклад, зміну контенту або статусу).
2. Observer – інтерфейс, що визначає метод:
   update(File, String) — кожен клас-наступник реалізує свій спосіб реагування на оновлення файлу.
3. ConcreteObserver (в даному випадку LogObserver, StatusBarObserver) — конкретні підписники:
   LogObserver — записує зміни файлу у лог.
   StatusBarObserver — оновлює статус-бар у UI (виводить повідомлення про дію над файлом та кількість символів у контенті).
   Всі реалізують метод update(File, String) з інтерфейсу Observer.
4. FileRepository – Concrete Subject, реалізує інтерфейс Subject і відповідає за збереження/видалення файлів, керування їхнім станом та повідомлення підписників про будь-які зміни.

Приклад реалізації шаблону:

1. Спочатку Client створює об’єкти, які беруть участь у патерні:
   - Створюється FileRepository — Concrete Subject, який зберігає файли та повідомляє підписників про зміни.
   - Створюються спостерігачі: LogObserver, StatusBarObserver.
   - спостерігачі підписуються на FileRepository через attach().
2. Далі Client виконує дії над файлом. Наприклад зберігає, видаляє або оновлює файл.
3. Після кожної дії FileRepository повідомляє всіх підписників через notifyObservers(file, "action"):
   - LogObserver.update(file, "saved") → записує зміни у лог.
   - StatusBarObserver.update(file, "saved") → оновлює статус-бар, відображаючи дію над файлом та стан його контенту.

У такій реалізації патерн Observer дозволяє Client керувати файлами та отримувати інформацію про зміни через підписників, не знаючи деталей їх обробки, а Concrete Subject (FileRepository) відповідає за збереження файлів і повідомлення підписників. Observer’и забезпечують реакцію на зміни і відображення стану.
Демонстрація роботи шаблону Observer

```
public class App {
    public static void main(String[] args) {
        FileRepository repository = new FileRepository( jsonFilePath: "");
        CommandInvoker invoker = new CommandInvoker();
        repository.attach(new LogObserver());
        repository.attach(new StatusBarObserver());

        File fileJson = new File();
        fileJson.setId(1L);
        fileJson.setFileName("file_json1");
        fileJson.setFilePath("C:\\Users\\AHOTA\\IdeaProjects\\trpz\\json");
        fileJson.setContent("\"text\": \"Hello JSON\"");

        File fileTxt = new File();
        fileTxt.setId(2L);
        fileTxt.setFileName("file_txt");
        fileTxt.setFilePath("C:\\Users\\AHOTA\\IdeaProjects\\trpz\\txt");
        fileTxt.setContent("\"text\": \"afqk\"");


        Command save1 = new SaveFileCommand(repository, fileJson);
        Command save2 = new SaveFileCommand(repository, fileTxt);
        Command delete = new DeleteFileCommand(repository, fileId: 1L);
        InsertCharCommand insertChar = new InsertCharCommand(repository, fileId: 2L, character: '>', position: 5);

        invoker.executeCommand(save1);
        invoker.executeCommand(save2);
        invoker.executeCommand(insertChar);
        invoker.executeCommand(delete);
    }
```

Рисунок 2 – код класу App

```java
@Override
public void save(File file) {
    filesInMemory.add(file);
    FileSaver saver = new FileSaver();
    String path = file.getFilePath().toLowerCase();
    if (path.endsWith("json")) {
        saver.setStrategy(new SaveAsJson());
    } else if (path.endsWith("xml")) {
        saver.setStrategy(new SaveAsXml());
    } else if (path.endsWith("txt")) {
        saver.setStrategy(new SaveAsTxt());
    }
    saver.save(file);
    notifyObservers(file, message: "saved");
}
```

Рисунок 3 – зміни в репозиторії на прикладі збереження

Результат виконання програми:

```
Файл збережено як JSON: C:\Users\АНЮТА\IdeaProjects\trpz\json\file_json1.json
[LOG] File 'file_json1' event: saved
[StatusBar] Action: saved | Symbols in content: 20
File saved: file_json1
Збережено як TXT: C:\Users\АНЮТА\IdeaProjects\trpz\txt\file_txt.txt
[LOG] File 'file_txt' event: saved
[StatusBar] Action: saved | Symbols in content: 14
File saved: file_txt
Збережено як TXT: C:\Users\АНЮТА\IdeaProjects\trpz\txt\file_txt.txt
[LOG] File 'file_txt' event: updated
[StatusBar] Action: updated | Symbols in content: 15
Char `>` inserted in File: file_txt
[LOG] File 'file_json1' event: deleted
[StatusBar] Action: deleted | Symbols in content: 20
```

Рисунок 4 – вивід повідомлення в консолі

Переваги використання шаблону Observer для моєї теми проекту:

1. Шаблон Observer дозволяє об’єктам ефективно повідомляти інших про зміни, не знаючи деталей їхньої обробки.
2. Зміни у файлі одночасно можуть оновлювати лог, статус-бар та інші UI-компоненти — користувач миттєво бачить результат своїх дій, наприклад підтвердження збереження файлу, оновлення кількості символів або історію змін.
3. Розширюваність системи
   Підписників можна додавати або видаляти в будь-який момент, а сама система легко розширюється новими реакціями на події, наприклад функцією сповіщення про помилки або статистику редагування, без зміни коду FileRepository.

4. Єдине уніфіковане представлення для різних дій
   Всі спостерігачі реалізують один інтерфейс Observer, що дозволяє Concrete Subject повідомляти їх однаково, незалежно від того, чи це логування, оновлення статус-бару або будь-яка інша реакція на зміну файлу.

Недоліки використання шаблону Command у текстовому редакторі:

1. Велика кількість спостерігачів і часті оновлення файлів можуть знижувати продуктивність — при активному редагуванні великих файлів інтерфейс може трохи «гальмувати».
2. Асинхронна обробка повідомлень ускладнює відстеження стану файлів і налагодження взаємодії між компонентами

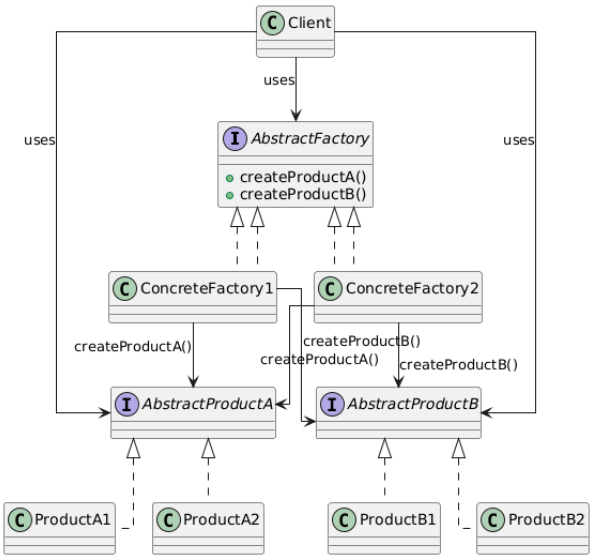
Висновок: У роботі реалізовано патерн Observer для реагування на зміни файлів у текстовому редакторі. Concrete Subject (FileRepository) відповідає за збереження, оновлення та видалення файлів, а підписники (Observer'и), такі як LogObserver та StatusBarObserver, отримують повідомлення про зміни і відповідно виконують свої дії — ведуть лог або оновлюють статус-бар. Це дозволяє відокремити збереження файлів від реакцій на їх зміну та забезпечує гнучкість у керуванні сповіщеннями. Було створено діаграму класів, яка наочно демонструє взаємодію компонентів системи: інтерфейс Observer визначає метод update, конкретні спостерігачі реалізують цю дію, а Concrete Subject (FileRepository) повідомляє всіх підписників про зміни.

Відповіді на контрольні питання

1. Яке призначення шаблону «Абстрактна фабрика»?

Дозволяє створювати сімейства взаємопов'язаних об'єктів без вказівки їх конкретних класів. Забезпечує узгодженість об'єктів одного стилю та відокремлює процес їх створення від використання.

2. Нарисуйте структуру шаблону «Абстрактна фабрика».



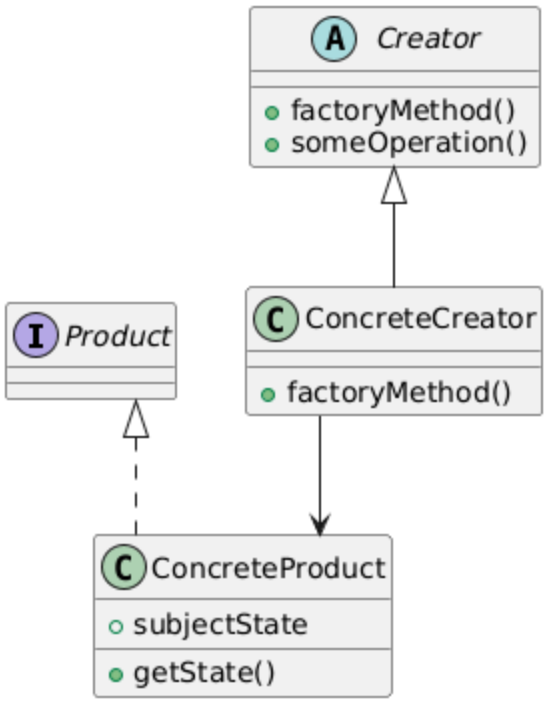
3. Які класи входять в шаблон «Абстрактна фабрика», та яка між ними взаємодія?

- AbstractFactory — інтерфейс для створення продуктів.
- ConcreteFactory — реалізація конкретної фабрики, створює конкретні продукти.
- AbstractProduct — інтерфейс продукту.
- ConcreteProduct — реалізація конкретного продукту.
  Взаємодія: клієнт використовує фабрику через AbstractFactory для створення продуктів одного стилю.

4. Яке призначення шаблону «Фабричний метод»?

Визначає інтерфейс для створення об’єктів певного базового типу і дозволяє підставляти підкласи без зміни коду клієнта.

5. Нарисуйте структуру шаблону «Фабричний метод».



6. Які класи входять в шаблон «Фабричний метод», та яка між ними взаємодія?

- Creator — базовий клас, що визначає фабричний метод.
- ConcreteCreator — реалізація фабричного методу, створює конкретний продукт.
- Product — базовий інтерфейс продукту.
- ConcreteProduct — конкретна реалізація продукту.
  Взаємодія: клієнт використовує Creator для створення об’єкта через фабричний метод, не знаючи конкретного підкласу.

7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?

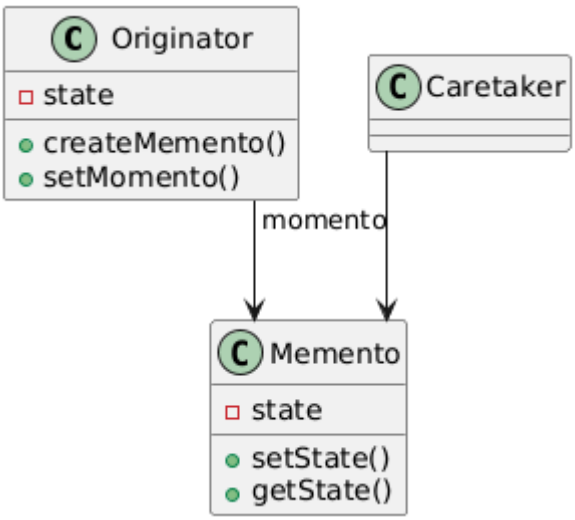
Abstract Factory створює сімейства взаємопов’язаних об’єктів (кілька типів продуктів), узгоджених між собою.

Factory Method створює один тип об’єкта і дозволяє підставляти підкласи без зміни коду клієнта.

8. Яке призначення шаблону «Знімок»?

Дозволяє зберігати та відновлювати стан об’єкта без порушення інкапсуляції. Використовується для реалізації undo/redo.

9. Нарисуйте структуру шаблону «Знімок».



10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія?

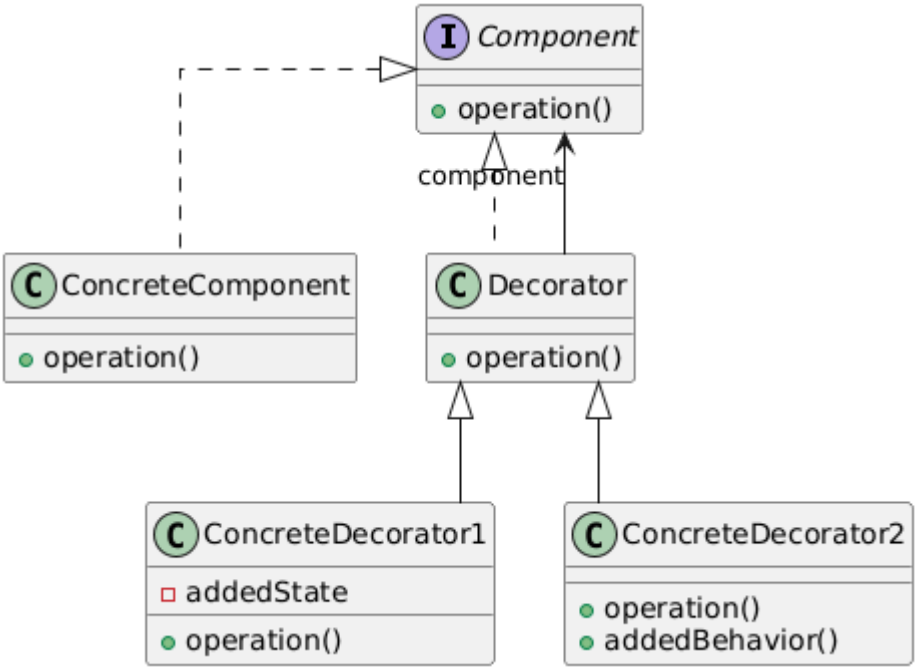
- Originator — об’єкт, стан якого потрібно зберегти.
- Memento — об’єкт для зберігання стану.
- Caretaker — управляє збереженими станами.

Взаємодія: Originator створює Memento, Caretaker зберігає його, а пізніше передає назад для відновлення стану.

11. Яке призначення шаблону «Декоратор»?

Дозволяє динамічно додавати функціональні можливості об’єкту без зміни його коду, обертаючи його в новий клас.

12. Нарисуйте структуру шаблону «Декоратор».



13. Які класи входять в шаблон «Декоратор», та яка між ними взаємодія?

- Component — базовий інтерфейс або клас об’єкта.
- ConcreteComponent — конкретний об’єкт, який можна декорувати.
- Decorator — базовий клас для декораторів, містить посилання на Component.
- ConcreteDecorator — додає нові обов’язки до об’єкта.
  Взаємодія: Decorator обертає Component і виконує додаткові дії під час виклику operation().

14. Які є обмеження використання шаблону «декоратор»?

Велика кількість дрібних класів ускладнює систему.

Складно конфігурувати об’єкти, які обгорнуті в декілька обгорток одночасно.