Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 6

з дисципліни «Теорія розробки прогрманого забезпечення»

Тема: «Патерни проектування.»

Варіант «Powershell terminal»

Виконав:

студент групи ІА-32
Бечке Олексій Ігорович

Перевірив:

Мягкий Михайло Юрійович

Київ 2025

Зміст

Вступ

Тема: Патерни проектування.

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

Тема проєкту: Powershell terminal (strategy, command, abstract factory, bridge, interpreter, client-server). Термінал для powershell повинен нагадувати типовий термінал з можливістю налаштування кольорів синтаксичних конструкцій, розміру вікна, фону вікна, а також виконання команд powershell і виконуваних файлів, а також працювати в декількох вікнах терміналу (у вкладках або одночасно шляхом розділення вікна).

Теоретичні відомості

Шаблон «Abstract Factory»

Призначення патерну: Шаблон «Абстрактна фабрика» використовується для створення сімейств об'єктів без вказівки їх конкретних класів [6]. Для цього виноситься загальний інтерфейс фабрики (AbstractFactory) і створюються його реалізації для різних сімейств продуктів. Хорошим прикладом використання абстрактної фабрики є ADO.NET: існує загальний клас DbProviderFactory, здатний створювати об'єкти типів DbConnection, DbDataReader, DbAdapter та ін.; існують реалізації цих фабрик і об'єктів – SqlProviderFactory, SqlConnection, SqlDataReader, SqlAdapter і так далі. Відповідно, якщо додатку необхідно працювати з різними базами даних (чи потрібна така можливість), то досить використати базові реалізації (Db.) і підставити відповідну фабрику у момент ініціалізації фабрики (Factory = new SqlProviderFactory()).

Цей шаблон передусім структурує знання про схожі об'єкти (що називаються сімействами, як класи для доступу до БД) і створює можливість взаємозаміни різних сімейств (робота з Oracle ведеться також, як і робота з SQL Server). Проте, при використанні такої схеми украй незручно розширювати фабрику – для додавання нового методу у фабрику необхідно додати його в усіх фабриках і створити відповідні класи, що створюються цим методом.

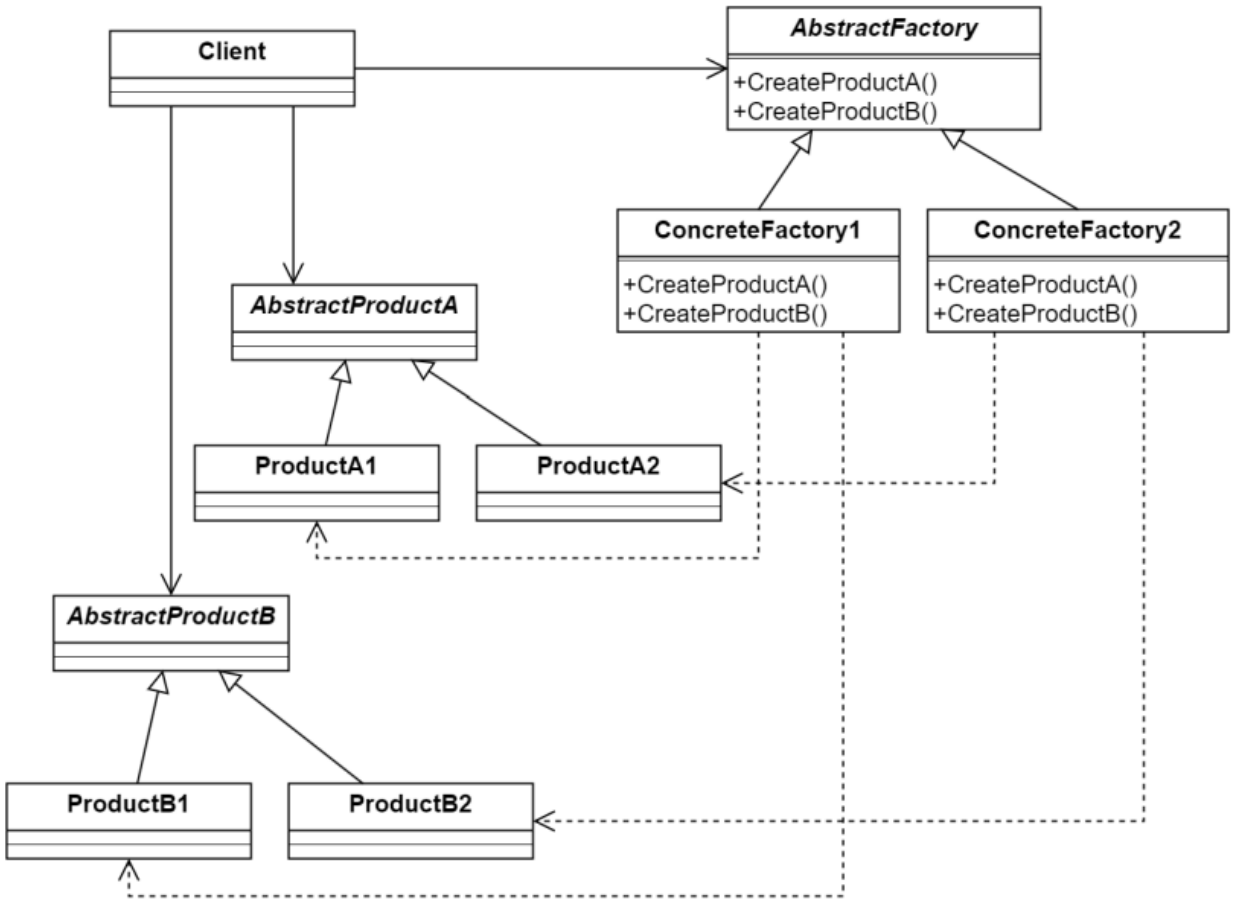


Рисунок 6.1. Структура патерну «Абстрактна фабрика»

Проблема: Ви розробляєте для Roguelike гри модуль автоматичної генерації кімнат. Модуль генерації кімнат, в процесі генерації конкретної кімнати, створює стіни, двері, вікна, підлогу та меблі в кімнаті. Модуль повинен підтримувати генерацію кімнат у різних стилях, таких як стиль хайтек, модерн, класичний офіс. Також критично є щоб всі елементи в одній згенерованій кімнаті відносилися до одного стилю.

Рішення: Для вирішення поставленої задачі дуже добре підходить використання патерну «Абстрактна фабрика».

Для кожного типу продукту створюємо свій інтерфейс продукту який об'являє функції доступні для використання з цим продуктом. Наприклад, можна виділити інтерфейси для стін, вікон, дверей, підлоги, а також окремі інтерфейси для меблів, такі як стіл, шафа, крісло та інші. Від кожного інтерфейсу наслідуємо і реалізуємо продукти різних типів, наприклад, для стола: інтерфейс ITable та реалізації продуктів HighTechTable, ModernTable та інші.

Далі визначаємо інтерфейс для абстрактної фабрики, який буде містити методи для створення кожного типу продукту. Створюємо класи конкретних фабрик під кожен стиль: HighTechFabric, ModernFabric та інші. Кожна конкретна фабрика буде створювати всі типи продуктів, але всі вони будуть відноситися до одного стилю.

Далі в алгоритм генерації кімнати будемо передавати конкретну фабрику і алгоритм при створенні продуктів тільки через фабрику завжди буде отримувати продукти одного стилю і працювати з продуктами тільки через інтерфейс продукта.

Таким чином ми вирішуємо проблему узгодженості стилів всіх елементів в кімнаті, а за рахунок використання різних конкретних фабрик ми зможемо генерувати кімнати різних стилів. Якщо нам потрібно буде добавити ще один стиль, то достатньо буде реалізувати нові дочірні класи для кожного елемента кімнати, а також нову конкретну фабрику під цей стиль.

Переваги та недоліки:

+ Спрощує створення об’єктів і код стає легшим для розуміння.

+ Об’єкти створені однією фабрикою добре узгоджуються один з одним і зменшується кількість помилок взаємодії між ними.

+ Відокремлення створення об’єктів від їх використання, за рахунок чого, код стає біль структурованим.

+ Додавання нових сімейств продуктів виконується без зміни існуючого коду.

- Збільшується складність коду, особливо для простих проєктів.

- Додавання нового типу продукту є складним і вимагає змін коду в багатьох місцях

Хід роботи

Завдання

- Ознайомитись з короткими теоретичними відомостями.
- Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
- Реалізувати один з розглянутих шаблонів за обраною темою.
- Реалізувати не менше 3-х класів відповідно до обраної теми.
- Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

Реалізація шаблону проєктування для майбутньої системи

Ми використовуємо патерн Abstract Factory для створення сімейств взаємопов'язаних об'єктів веб-інтерфейсу (таких як запрошення до вводу та кнопки виконання) без прив'язки до їхніх конкретних класів. Замість того, щоб жорстко прописувати HTML-розмітку або логіку стилізації безпосередньо в контролері, ми делегуємо процес створення цих елементів об'єкту-фабриці (WebUIFactory), конкретна реалізація якої (DarkUIFactory або LightUIFactory) обирається динамічно залежно від налаштувань теми. Це дозволяє відділити

клієнтський код (веб-контролер) від деталей реалізації візуального оформлення, гарантує узгодженість стилю всіх елементів інтерфейсу, а також дає можливість легко додавати нові візуальні теми без необхідності вносити зміни в основну логіку контролера.

```
public interface WebPrompt {
    String renderHtml();  1 us
}
```

Рис. 1 – Код інтерфейсу WebPrompt

```
public interface WebButton {
    String renderHtml();  1 us
}
```

Рис. 2 – Код інтерфейсу WebButton

```
public interface WebUIFactory
    WebPrompt createPrompt();
    WebButton createButton();
}
```

Рис. 3 – Код інтерфейсу WebUIFactory

```
public class DarkPrompt implements WebPrompt {  1 usage  Oleksiy B
    @Override  1 usage  Oleksiy Bechke *
    public String renderHtml() {
        return "<span style='color: lightgreen; " +
                "font-weight: bold;'>PS Dark User&gt;</span>";
    }
}
```

Рис. 4 – Код класу DarkPrompt

```
public class DarkButton implements WebButton {  1 usage
    @Override  1 usage  Oleksiy Bechke *
    public String renderHtml() {
        return "<button type='submit' " +
                "style='background-color: #333; " +
                "color: white; " +
                "border: 1px solid #555; " +
                "padding: 5px 10px; " +
                "cursor: pointer;'>Execute</button>";
    }
}
```

Рис. 5 – Код класу DarkButton

```
public class DarkUIFactory implements WebUIFactory {  2 usages  Oleksiy Bechke
    @Override  1 usage  Oleksiy Bechke
    public WebPrompt createPrompt() { return new DarkPrompt(); }

    @Override  1 usage  Oleksiy Bechke
    public WebButton createButton() { return new DarkButton(); }
}
```

Рис. 6 – Код класу DarkUIFactory

```
public class LightPrompt implements WebPrompt {  1 usage  Oleksiy Bechke
    @Override  1 usage  Oleksiy Bechke
    public String renderHtml() {
        return "<span style='color: blue; font-weight: bold;'>PowerShell (Light) $</span>";
    }
}
```

Рис. 7 – Код класу LightPrompt

```
public class LightButton implements WebButton {  1 usage  Oleksiy Bechke *
    @Override  1 usage  Oleksiy Bechke *
    public String renderHtml() {
        return "<button type='submit' " +
                "style='background-color: #f0f0f0; " +
                "color: black; border: 1px solid #ccc; " +
                "padding: 5px 10px; cursor: pointer;'>Run Script</button>";
    }
}
```

Рис. 8 – Код класу LightButton

```
public class LightUIFactory implements WebUIFactory {  2 usages  Ol
    @Override  1 usage  Oleksiy Bechke
    public WebPrompt createPrompt() { return new LightPrompt(); }

    @Override  1 usage  Oleksiy Bechke
    public WebButton createButton() { return new LightButton(); }
}
```

Рис. 9 – Код класу LightUIFactory

Зображення структури шаблону

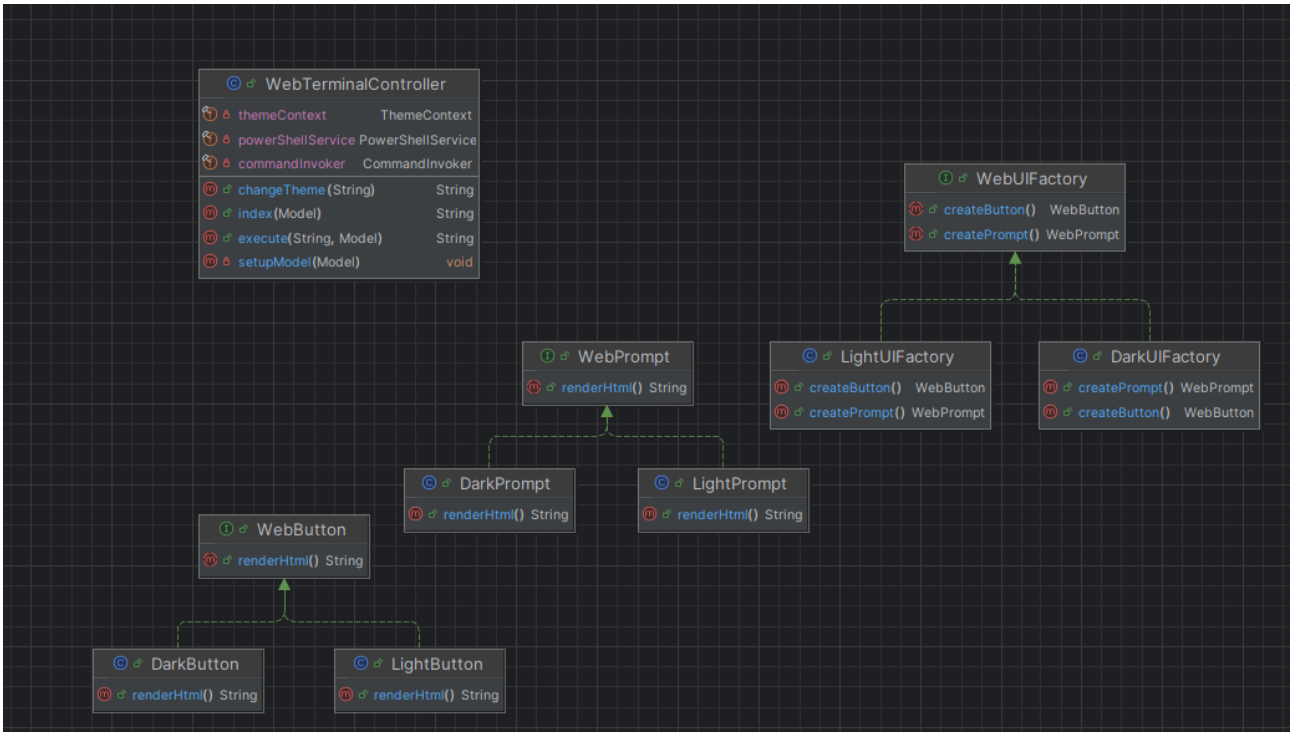


Рис. 10 – Структура шаблону Abstract Factory

Структура реалізації:

- WebUIFactory (Abstract Factory) – інтерфейс, що оголошує методи для створення сімейства взаємопов’язаних об’єктів інтерфейсу (запрошення до вводу та кнопки виконання) без прив’язки до конкретних класів реалізації.
- DarkUIFactory / LightUIFactory (Concrete Factories) – конкретні реалізації фабрики, які відповідають за створення компонентів, стилізованих під певну тему (темну або світлу), гарантуючи їх візуальну сумісність та узгодженість.
- WebPrompt / WebButton (Abstract Products) – інтерфейси для окремих типів елементів інтерфейсу, які визначають загальний контракт (метод renderHtml()) для отримання їхнього HTML-представлення.

- DarkPrompt / LightPrompt та DarkButton / LightButton (Concrete Products) – конкретні класи компонентів, що реалізують відповідні інтерфейси та повертають специфічний HTML-код із CSS-стилями, характерними для обраної теми.
- WebTerminalController (Client) – клієнтський клас, який використовує абстрактну фабрику для генерації необхідних елементів інтерфейсу, не знаючи про деталі їхньої реалізації чи конкретні класи продуктів.Такий підхід дозволяє інкапсулювати запити до системи у вигляді окремих об'єктів, відокремлюючи об'єкт, що ініціює запит (веб-контролер), від об'єкта, що знає, як його виконати (сервіс). Це дає змогу централізовано керувати історією операцій, легко додавати нові типи команд та змінювати механізм їх обробки без впливу на клієнтський код.

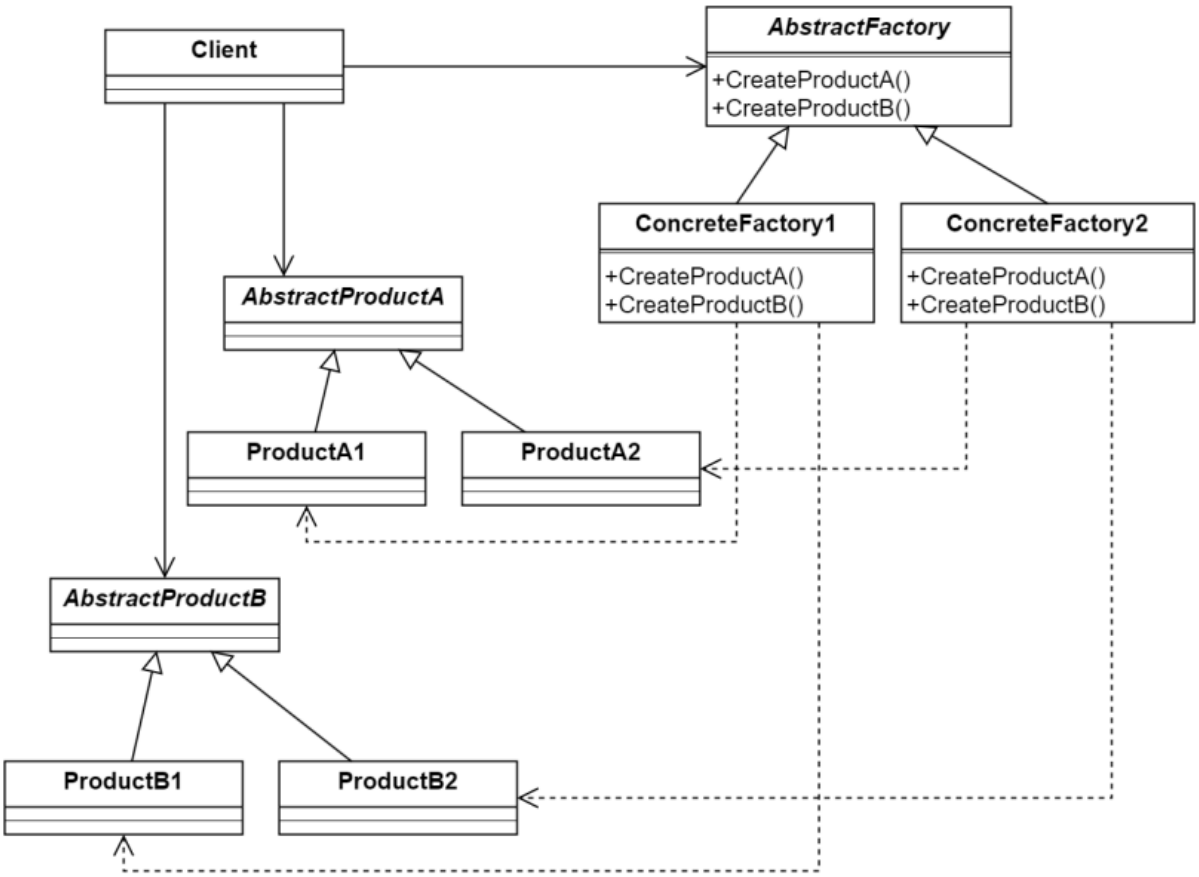
Такий підхід дозволяє створювати сімейства взаємопов'язаних об'єктів веб-інтерфейсу, відокремлюючи логіку їх створення від клієнтського коду контролера. Це забезпечує легку заміну візуальних тем та спрощує додавання нових стилів у майбутньому без необхідності змінювати основну бізнес-логіку програми, гарантуючи при цьому, що всі елементи інтерфейсу завжди відповідатимуть одному стилю.

Питання до лабораторної роботи

1. Яке призначення шаблону «Абстрактна фабрика»?

Шаблон «Абстрактна фабрика» використовується для створення сімейств об'єктів без вказівки їх конкретних класів.

2. Нарисуйте структуру шаблону «Абстрактна фабрика».



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

AbstractFactory (Абстрактна фабрика): Оголошує інтерфейс для операцій, що створюють абстрактні об'єкти-продукти (наприклад, CreateProductA(), CreateProductB()).

ConcreteFactory (Конкретна фабрика): Реалізує операції абстрактної фабрики для створення конкретних об'єктів-продуктів (наприклад, ConcreteFactory1, ConcreteFactory2). Кожна конкретна фабрика відповідає за створення продуктів певної сім'ї.

AbstractProduct (Абстрактний продукт): Оголошує інтерфейс для типу об'єкта-продукту (наприклад, AbstractProductA, AbstractProductB).

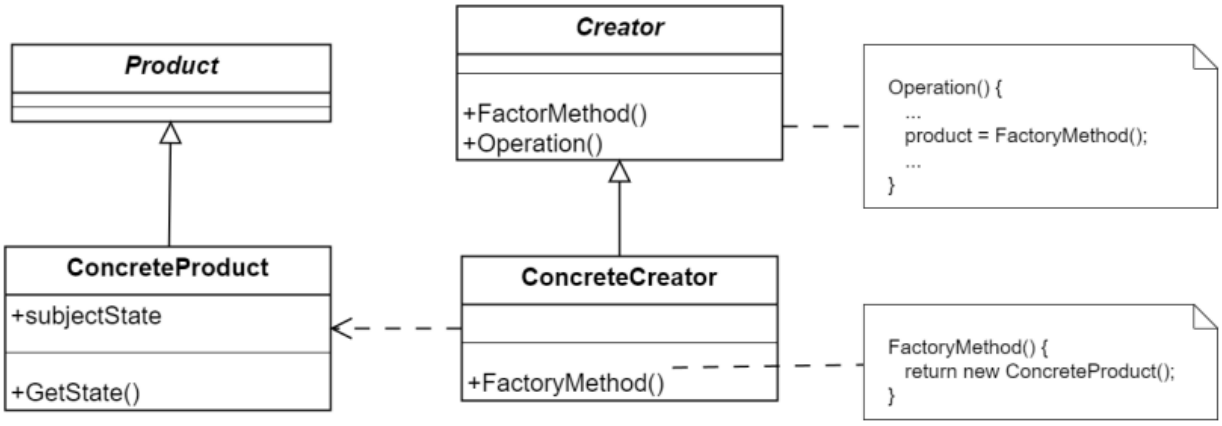
ConcreteProduct (Конкретний продукт): Визначає об'єкт-продукт, що створюється відповідною конкретною фабрикою, та реалізує інтерфейс AbstractProduct. Приклади: ProductA1, ProductB1 (створюються ConcreteFactory1) та ProductA2, ProductB2 (створюються ConcreteFactory2).

Client (Клієнт): Використовує виключно інтерфейси, оголошені класами AbstractFactory та AbstractProduct.

4. Яке призначення шаблону «Фабричний метод»?

Шаблон «Фабричний метод» визначає інтерфейс для створення об'єктів певного базового типу.

5. Нарисуйте структуру шаблону «Фабричний метод».



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

Product (Продукт): Визначає інтерфейс об'єктів, які створює фабричний метод.

ConcreteProduct (Конкретний продукт): Реалізує інтерфейс Product. Це конкретний клас об'єкта, який створюється.

Creator (Творець): Оголошує фабричний метод, який повертає об'єкт типу Product. Також може містити реалізацію методів, що оперують продуктами (наприклад, Operation()), викликаючи фабричний метод для їх створення.

ConcreteCreator (Конкретний творець): Перевизначає (реалізує) фабричний метод, щоб повернути екземпляр ConcreteProduct.

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

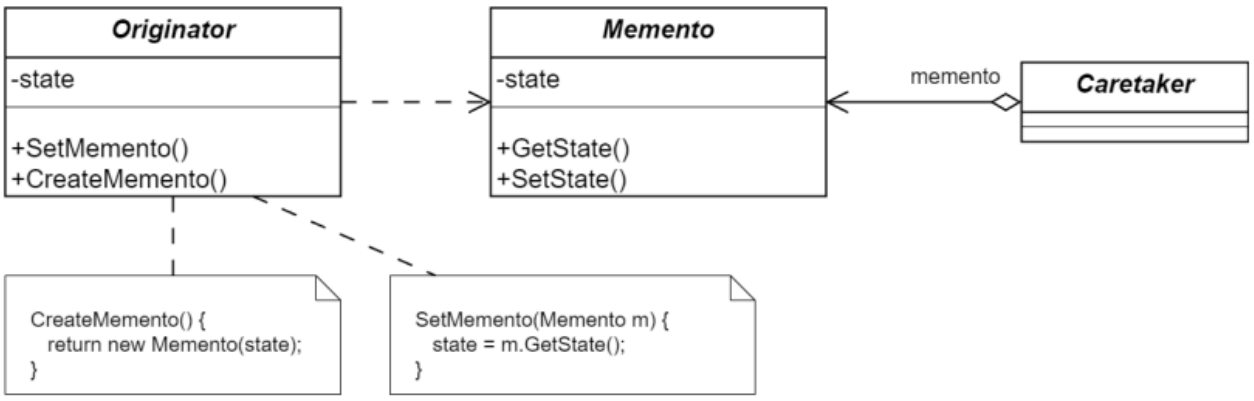
Основна відмінність полягає в масштабі та способі реалізації: «Абстрактна фабрика» створює цілі сімейства взаємопов'язаних об'єктів, використовуючи для цього окремий об'єкт-фабрику, тоді як «Фабричний метод» відповідає за створення лише одного продукту і реалізується через успадкування, де підкласи перевизначають метод створення. Крім того, «Фабричний метод» дозволяє легко додавати нові продукти через створення нових підкласів, на відміну від «Абстрактної фабрики», де додавання нового типу продукту є складним процесом, що вимагає змін у всіх існуючих фабриках.

8. Яке призначення шаблону «Знімок»?

Шаблон використовується для збереження і відновлення стану об'єктів без порушення інкапсуляції. Об'єкт «Memento» служить виключно для збереження змін над початковим об'єктом (Originator). Лише початковий об'єкт має можливість зберігати і отримувати стан об'єкту «Memento» для власних цілей,

цей об'єкт є «порожнім» для кого-небудь ще. Об'єкт «Caretaker» використовується для передачі і зберігання мементо об'єктів в системі.

9. Нарисуйте структуру шаблону «Знімок».



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

Originator (Початковий об'єкт / Хазяїн): Об'єкт, стан якого необхідно зберігати. Він створює об'єкт Memento, що містить знімок його поточного внутрішнього стану, і використовує його для відновлення свого стану.

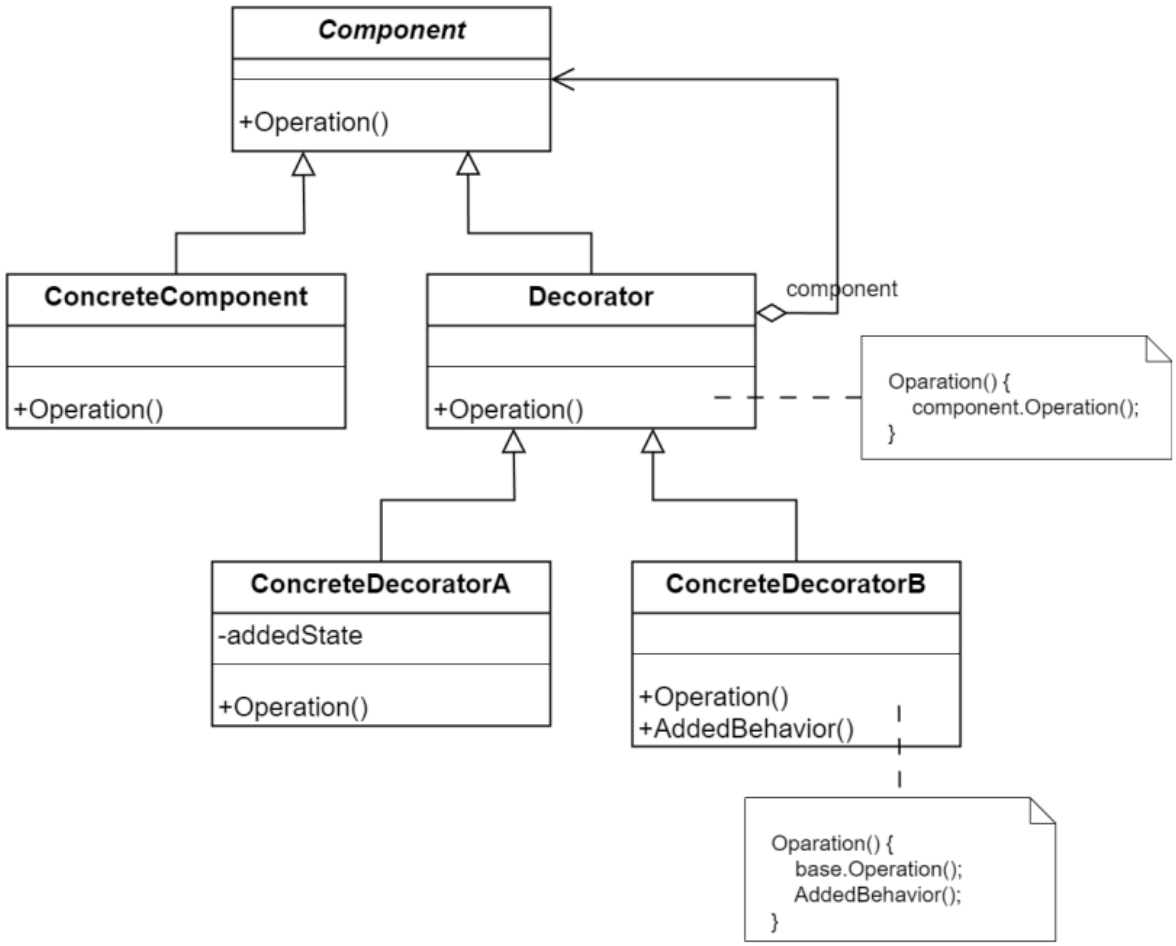
Memento (Знімок): Зберігає внутрішній стан об'єкта Originator. Він захищає цей стан від доступу будь-яких інших об'єктів, окрім того, що його створив (Originator), для якого він є «прозорим», тоді як для інших — «порожнім».

Caretaker (Опікун / Доглядач): Відповідає за зберігання та передачу об'єктів Memento, але не має права змінювати або читати їхній вміст.

11. Яке призначення шаблону «Декоратор»?

Шаблон призначений для динамічного додавання функціональних можливостей об'єкту під час роботи програми. Декоратор деяким чином «обертає» (за рахунок агрегації) початковий об'єкт зі збереженням його функцій, проте дозволяє додати додаткові дії. Такий шаблон надає гнучкіший спосіб зміни поведінки об'єкту чим просте спадкоємство, оскільки початкова функціональність зберігається в повному об'ємі. Більше того, таку поведінку можна застосовувати до окремих об'єктів, а не до усієї системи в цілому.

12. Нарисуйте структуру шаблону «Декоратор».



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

Component (Компонент): Визначає інтерфейс для об'єктів, на які можуть бути динамічно покладені додаткові обов'язки (наприклад, метод Operation()).

ConcreteComponent (Конкретний компонент): Визначає об'єкт, на який покладаються додаткові обов'язки.

Decorator (Декоратор): Зберігає посилання на об'єкт Component і визначає інтерфейс, що відповідає інтерфейсу Component. Тобто декоратор може виступати в ролі компонента.

ConcreteDecorator (Конкретний декоратор): Додає нові обов'язки (функціональність) до компонента (наприклад, ConcreteDecoratorA, ConcreteDecoratorB). Може додавати новий стан (addedState) або нову поведінку (AddedBehavior()).

14.Які є обмеження використання шаблону «декоратор»?

Велика кількість дрібних класів: Застосування цього шаблону часто призводить до створення значної кількості маленьких класів, що може ускладнити структуру проекту.

Складність конфігурації: Важко конфігурувати та керувати об'єктами, які загорнуті в декілька декораторів (обгорток) одночасно.

Висновок

Під час виконання лабораторної роботи я вивчив структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчитися застосовувати їх в реалізації програмної системи. Реалізував шаблон «Abstract Factory» у проєкті Power Shell термінал.