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 «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Виконала:
студентка групи ІА-33
Самойленко Анастасія

Перевірив:
асистент кафедри ІСТ
Мягкий Михайло Юрійович

Київ 2025

Тема: Патерни проектування.
Мета: Вивчити структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчитися застосовувати їх в реалізації програмної системи.
Тема роботи:
27. Особиста бухгалтерія (state, prototype, decorator, bridge, flyweight, SOA)
Програма повинна бути наочним засобом для ведення особистих фінансів: витрат і прибутку; з можливістю встановлення періодичних витрат / прибутку (зарплата і орендна плата); введення сканованих чеків з відповідними статтями витрат; побудова статистики; експорт/імпорт в Excel, реляційні джерела даних; різні рахунки; ведення єдиного фонду на всі рахунки (всією сім'єю) – на особливі потреби (ремонт, автомобіль, відпустка); можливість введення вкладів / кредитів для контролю банківських рахунків (звірка нарахованих відсотків з необхідними і т.д.).

Вступ

Під час розробки складних програмних систем важливо забезпечити гнучкість, масштабованість і можливість повторного використання коду. Для цього застосовуються патерни проектування — перевірені рішення типових завдань об’єктно-орієнтованого програмування. Вони спрощують процес розробки, підвищують зрозумілість структури коду та полегшують його подальшу підтримку.

У межах даної лабораторної роботи розглядаються структурні та поведінкові шаблони, такі як Abstract Factory, Factory Method, Memento, Observer і Decorator. Вивчення їхньої структури, принципів роботи та реалізація на практиці дозволяє глибше зрозуміти механізми взаємодії об’єктів у програмних системах і вдосконалити навички застосування патернів у власних проєктах.

Метою роботи є вивчення структури зазначених шаблонів та набуття практичних навичок їх використання під час реалізації програмних систем. Результатом стане розуміння способів застосування цих шаблонів для побудови більш ефективного, масштабованого та зручного у підтримці програмного забезпечення.

Теоретичні відомості

Шаблон (патерн) проєктування — це формалізований опис типового рішення, яке багаторазово зустрічається під час розроблення інформаційних систем. Він містить узагальнений спосіб вирішення певної задачі проєктування, рекомендації щодо його застосування та має загальновживану назву.

Головна мета використання шаблонів — повторне застосування вже знайдених ефективних рішень у нових ситуаціях. Важливою умовою є коректне моделювання предметної області, що дозволяє правильно визначити задачу та обрати відповідний патерн.

Застосування шаблонів проєктування забезпечує низку переваг:

- створення структурованої та логічної моделі системи;
- підвищення наочності та спрощення вивчення архітектури;
- глибше опрацювання структури програмного продукту;
- підвищення стійкості системи до змін вимог;
- спрощення подальшого доопрацювання та розширення;
- полегшення інтеграції систем і взаєморозуміння між розробниками завдяки спільному “словнику проєктування”.

Шаблон «Abstract Factory»

Шаблон «Абстрактна фабрика» використовується для створення сімейств об'єктів без вказівки їх конкретних класів. Дозволяє легко замінювати одне сімейство продуктів іншим.

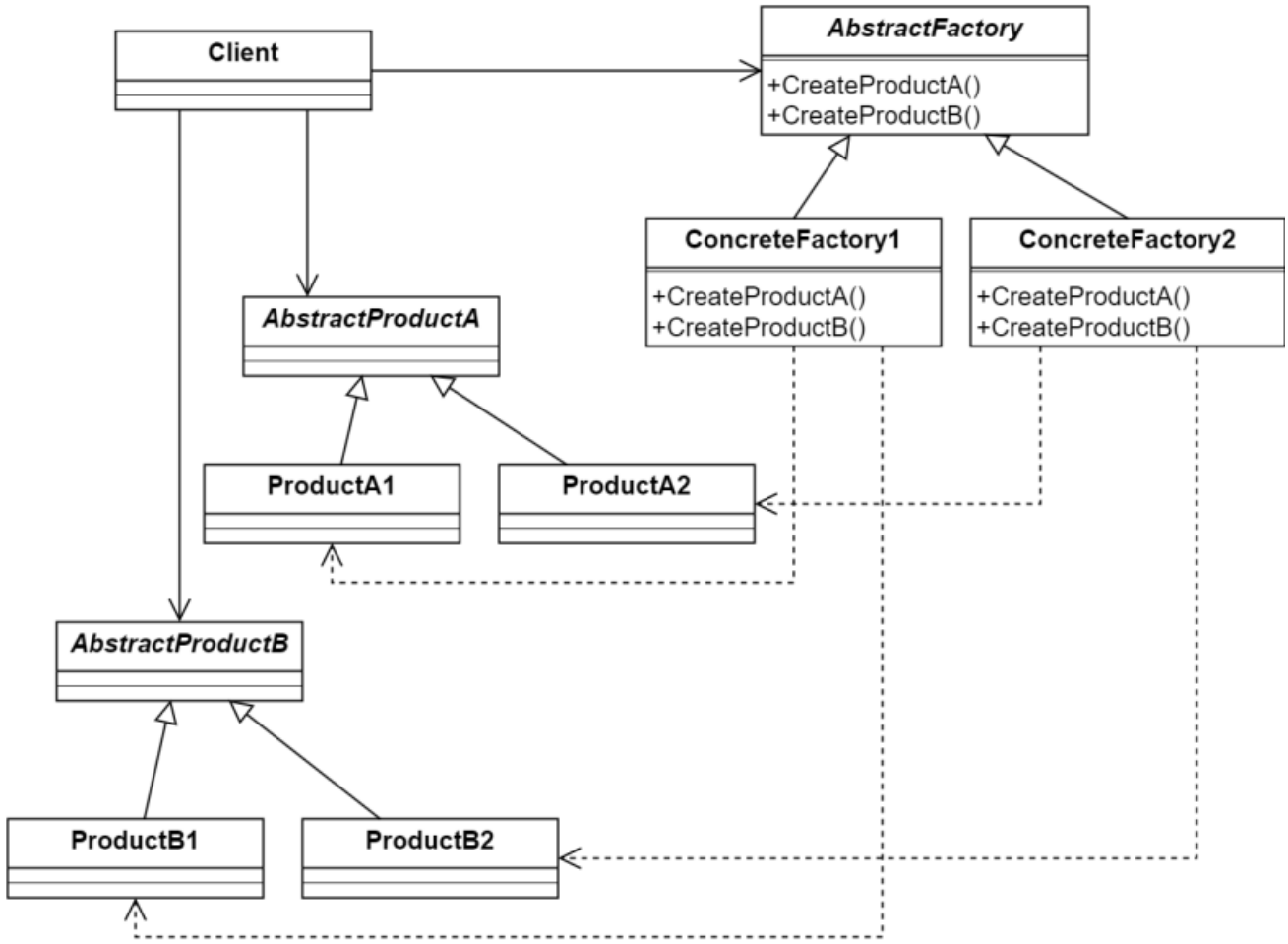
Мета: Визначається загальний інтерфейс фабрики (AbstractFactory), а його реалізації створюють об'єкти, що належать до одного стилю/сімейства (наприклад, фабрика для створення всіх елементів кімнати в стилі "Хай-тек": стін, дверей, меблів тощо).

Приклад: У Roguelike грі генератор кімнат може використовувати різні конкретні фабрики (HighTechFabric, ModernFabric) для забезпечення того, що всі елементи в кімнаті (стіни, двері, підлога) будуть узгодженого стилю.

Переваги: Узгодженість створюваних об'єктів, відділення створення від використання, легке додавання нових сімейств.

Недоліки: Складно додавати новий тип продукту (потрібні зміни в інтерфейсі та всіх конкретних фабриках).

Структура патерну Абстрактна фабрика на рівні об’єктів:



Шаблон «Factory Method»

Шаблон «Фабричний метод» визначає інтерфейс для створення об'єктів певного базового типу. Це зручно, коли хочеться додати можливість створення об'єктів не базового типу, а деякого дочірнього. Фабричний метод у такому разі є зачіпкою для впровадження власного конструктора об'єктів. Основна ідея полягає саме в заміні об'єктів їх підтипами, що при цьому зберігає ту ж функціональність; інша частина поведінки об'єктів

не є інтерфейсною (AnOperation) і дозволяє взаємодіяти із створеними об'єктами як з об'єктами базового типу. Тому шаблон «Фабричний метод» носить ще назву «Віртуальний конструктор»

Базовий клас містить метод створення об’єктів, але конкретні підкласи перевизначають його для створення потрібних екземплярів. Наприклад, клас PacketCreator має метод створення мережевих пакетів, а підкласи TcpCreator та UdpCreator реалізують його для відповідних протоколів.

Переваги:

Позбавляє необхідності жорсткої прив’язки до конкретних класів.

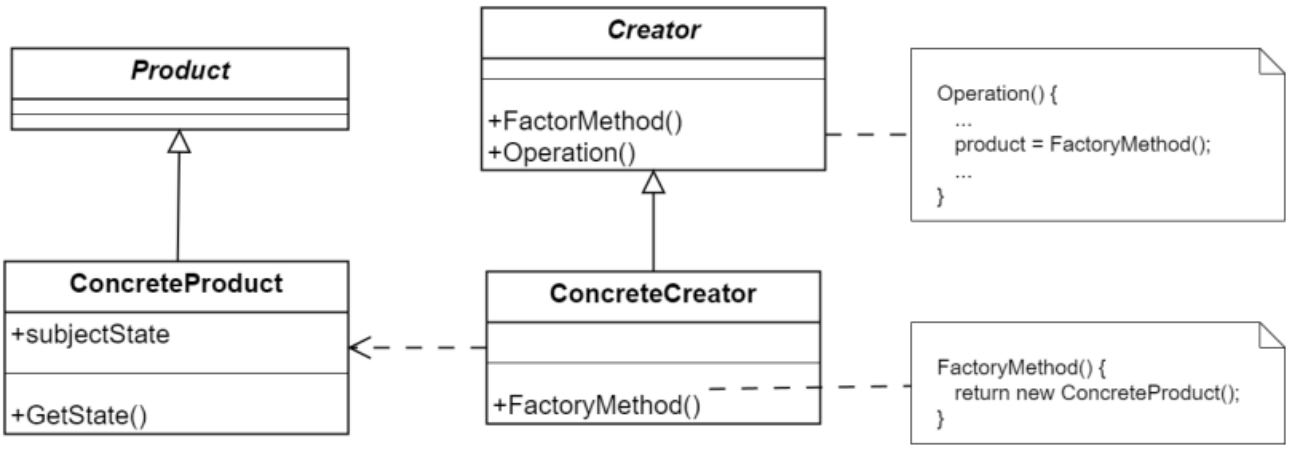
Спрощує розширення системи новими типами об’єктів.

Полегшує підтримку та тестування.

Недоліки:

– Може призвести до складної ієрархії класів при великій кількості підтипів.

Структура патерну Фабричний Метод:



Шаблон «Memento»

Шаблон використовується для збереження і відновлення стану об'єктів без порушення інкапсуляції [6]. 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.

Створюється спеціальний об’єкт Memento, який зберігає стан початкового об’єкта (Originator). Caretaker відповідає за передачу та зберігання цих знімків, не маючи доступу до їхнього вмісту.

Це дає можливість повертати систему до попереднього стану — наприклад, для реалізації функції «Скасувати дію».

Переваги:

Не порушує інкапсуляцію об’єкта.

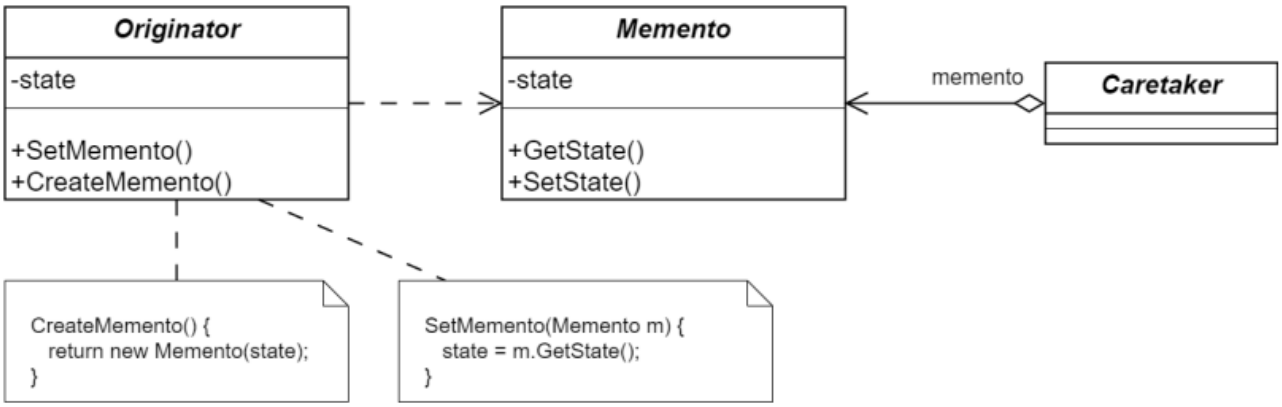
Спрощує структуру об’єкта, оскільки не потрібно зберігати історію змін усередині нього.

Недоліки:

– Часте створення знімків потребує багато пам’яті.

– Можливі втрати пам’яті, якщо старі знімки не очищуються.

Структура шаблону Знімок:



Шаблон «Observer»

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 .

Об’єкт Суб’єкт підтримує список спостерігачів, які підписані на його зміни. При зміні стану суб’єкт надсилає всім підписникам повідомлення.

Наприклад, у банківській системі після зміни балансу всі клієнти, що спостерігають за рахунком, отримують оновлені дані.

Переваги:

Підтримка асинхронних оновлень.

Можна динамічно додавати або видаляти спостерігачів.

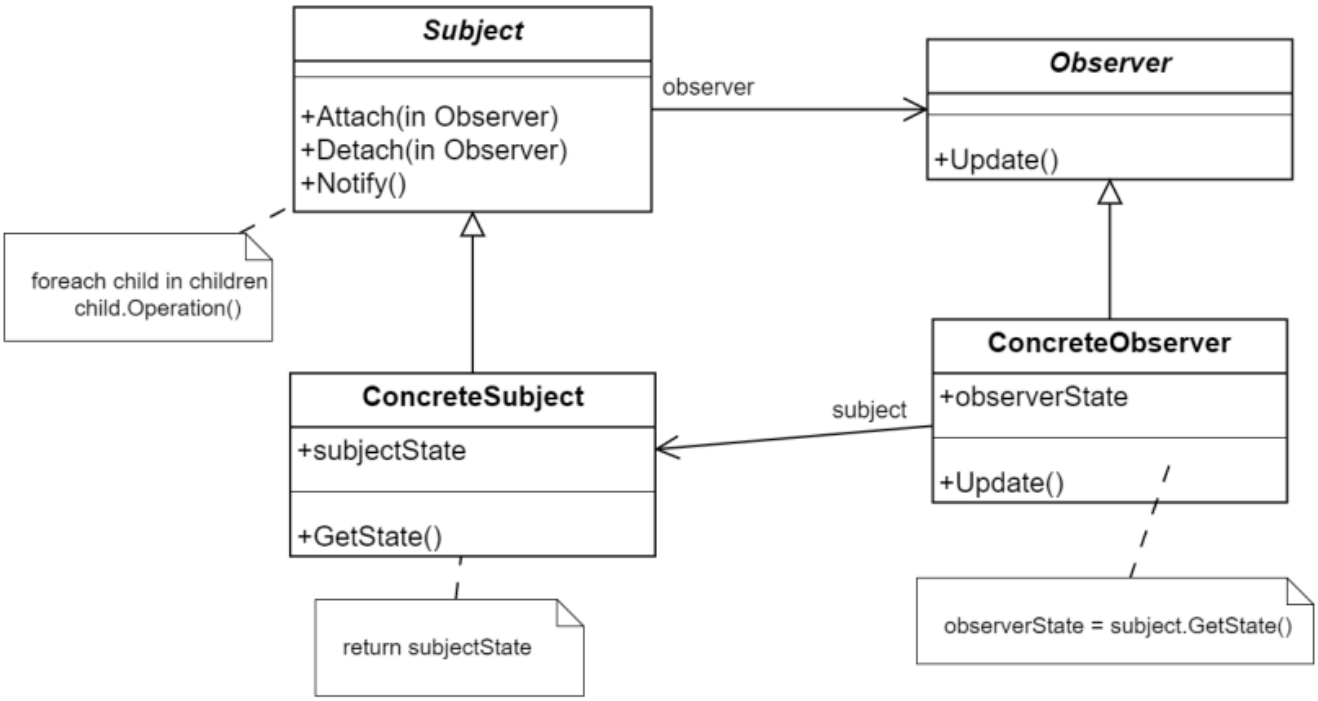
Знижує зв’язність між об’єктами.

Недоліки:

– Не контролюється порядок надсилання повідомлень.

– Може виникати велика кількість оновлень при багатьох підписниках.

Структура патерна Спостерігач:



Шаблон «Decorator»

Шаблон призначений для динамічного додавання функціональних можливостей об'єкту під час роботи програми [6]. 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.

Декоратор «обгортає» базовий об’єкт, зберігаючи його початкові властивості, але додає нову поведінку. Наприклад, до елемента інтерфейсу можна додати смугу прокрутки або тінь без зміни основного класу.

Переваги:

Гнучке додавання нових функцій «на льоту».

Не потребує зміни базового класу.

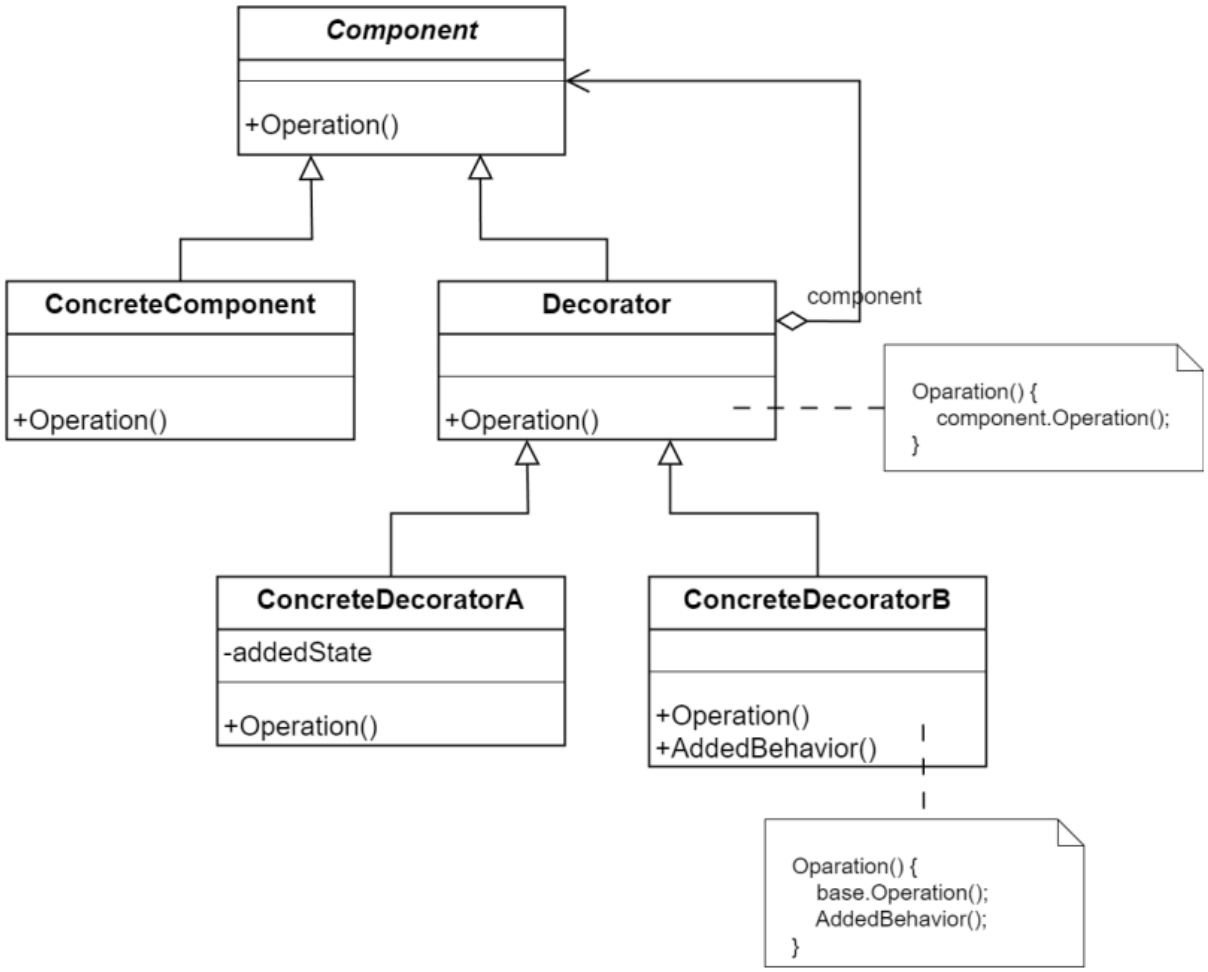
Дозволяє комбінувати різні декоратори.

Недоліки:

– Збільшує кількість дрібних класів.

– Ускладнює розуміння коду при багатьох рівнях обгортання.

Структура патерну Декоратор:



Хід роботи

1) Діаграма класів частини функціоналу робочої програми. (рис. 1)

Рисунок 1 – Діаграма класів

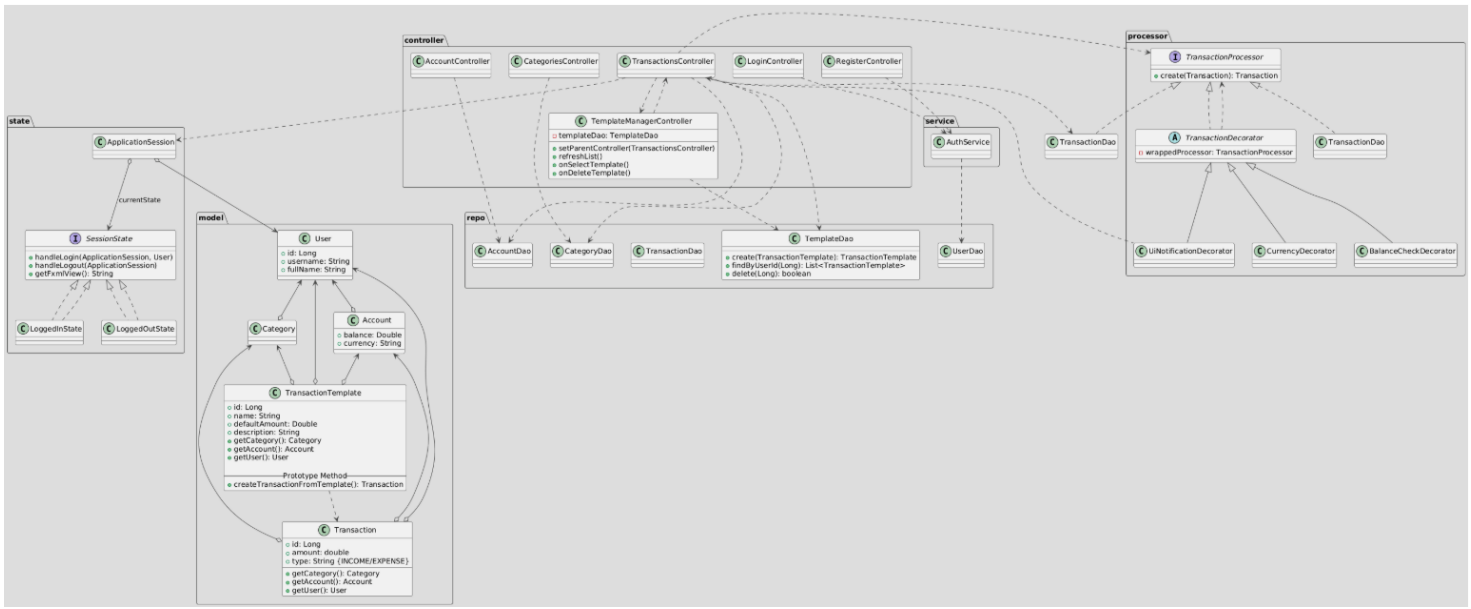


Рисунок 2 – Діаграма класів з патерном Decorator

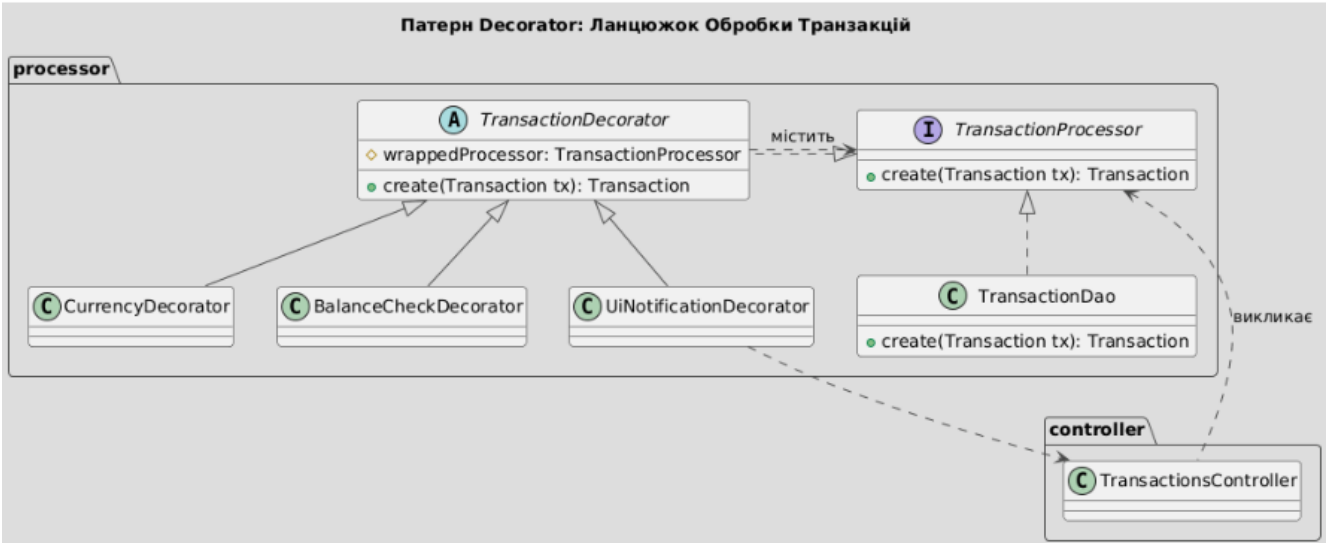


Рисунок 3 – Представлення самого шаблону на діаграмі

2) Реалізація Шаблону «Decorator»

Патерн Декоратор (Decorator) належить до структурних патернів проєктування. Його основна мета — динамічно додавати нові обов'язки об'єкту, обгортаючи його. Це дозволяє розширювати функціональність класу без зміни його структури.

У проєкті управління особистими фінансами патерн Декоратор був застосований для створення гнучкого ланцюжка послідовної обробки об'єкта Transaction перед його фінальним збереженням у базі даних.

Реалізація патерну Декоратор у системі обробки транзакцій базується на спільному інтерфейсі та створенні ієрархії класів для динамічного додавання функціоналу:

1. Компонент (Component)

Фундаментом архітектури є інтерфейс TransactionProcessor. Цей інтерфейс виступає в ролі Компонента, який визначає єдиний контракт — метод create(Transaction tx). Всі класи, що беруть участь у ланцюжку обробки (як базовий компонент, так і всі декоратори), повинні реалізовувати цей інтерфейс, забезпечуючи взаємозамінність об'єктів.

2. Конкретний Компонент (Concrete Component)

Клас TransactionDao є Конкретним Компонентом і представляє ядро функціональності. Він не додає жодної бізнес-логіки; його єдиний обов'язок — це постійне збереження об'єкта Transaction у базі даних. Цей об'єкт завжди знаходиться в кінці ланцюжка обробки.

3. Абстрактний Декоратор (Decorator)

Клас TransactionDecorator є абстрактним класом, який реалізує інтерфейс TransactionProcessor. Він слугує базою для всіх конкретних декораторів. Ключовим елементом є його поле (захищена змінна) для зберігання посилання на обгорнутий об'єкт (wrappedProcessor), що дозволяє делегувати виклик далі по ланцюжку через super.create(tx).

4. Конкретні Декоратори (Concrete Decorators)

Це класи, які обгортають попередні об'єкти, додаючи нові, незалежні обов'язки:

- CurrencyDecorator: Додає логіку фінансової модифікації. Він відповідає за конвертацію суми транзакції з її початкової валюти у базову валюту системи (UAH) перед тим, як передати транзакцію далі.
- BalanceCheckDecorator: Додає логіку бізнес-валідації. Він перевіряє, чи достатньо коштів на рахунку (Account.balance) для покриття суми витрати, і кидає виняток, якщо ліміт перевищено.
- UiNotificationDecorator: Додає логіку керування інтерфейсом. Цей декоратор є зовнішнім шаром, який обгортає весь ланцюжок у блок try-catch, щоб перехоплювати результат (успіх або помилку) та відображати відповідне спливаюче повідомлення користувачеві.

Під час застосування патерну «Декоратор» вдалося досягти покращення структури програми, підвищити її гнучкість, розширюваність і зручність у подальшій підтримці.

Основні переваги використання цього шаблону:

Відкритість для розширення, закритість для змін

Система не потребує змін у вже реалізованих класах. Якщо в майбутньому потрібно додати нову функціональність (наприклад, TaxCalculationDecorator або FraudCheckDecorator), достатньо створити новий клас-декоратор і додати його до ланцюжка. Це дозволяє легко розширювати функціональність без ризику порушення роботи наявного коду.

Чітке розділення обов’язків між компонентами

Кожен клас виконує одну конкретну функцію:

- CurrencyDecorator — відповідає за конвертацію валют;
- BalanceCheckDecorator — перевіряє наявність достатнього балансу перед здійсненням операції;
- TransactionDao — займається безпосередньою взаємодією з базою даних.

Такий підхід робить код більш зрозумілим і легким у супроводі.

Зручне керування помилками та взаємодією з користувачем

UiNotificationDecorator розташований на початку ланцюжка обробки і перехоплює всі можливі помилки, що можуть виникнути в процесі виконання нижчих декораторів. Наприклад, якщо BalanceCheckDecorator виявляє нестачу коштів, то UiNotificationDecorator перетворює цю технічну помилку на зрозуміле повідомлення користувачу та не дозволяє зберегти некоректну транзакцію.

Гнучке налаштування послідовності виконання операцій

Патерн дозволяє визначати логічну послідовність дій у ланцюжку декораторів. Наприклад, спочатку відбувається конвертація валют, а потім — перевірка балансу. Це забезпечує правильний порядок виконання дій і узгодженість результатів.

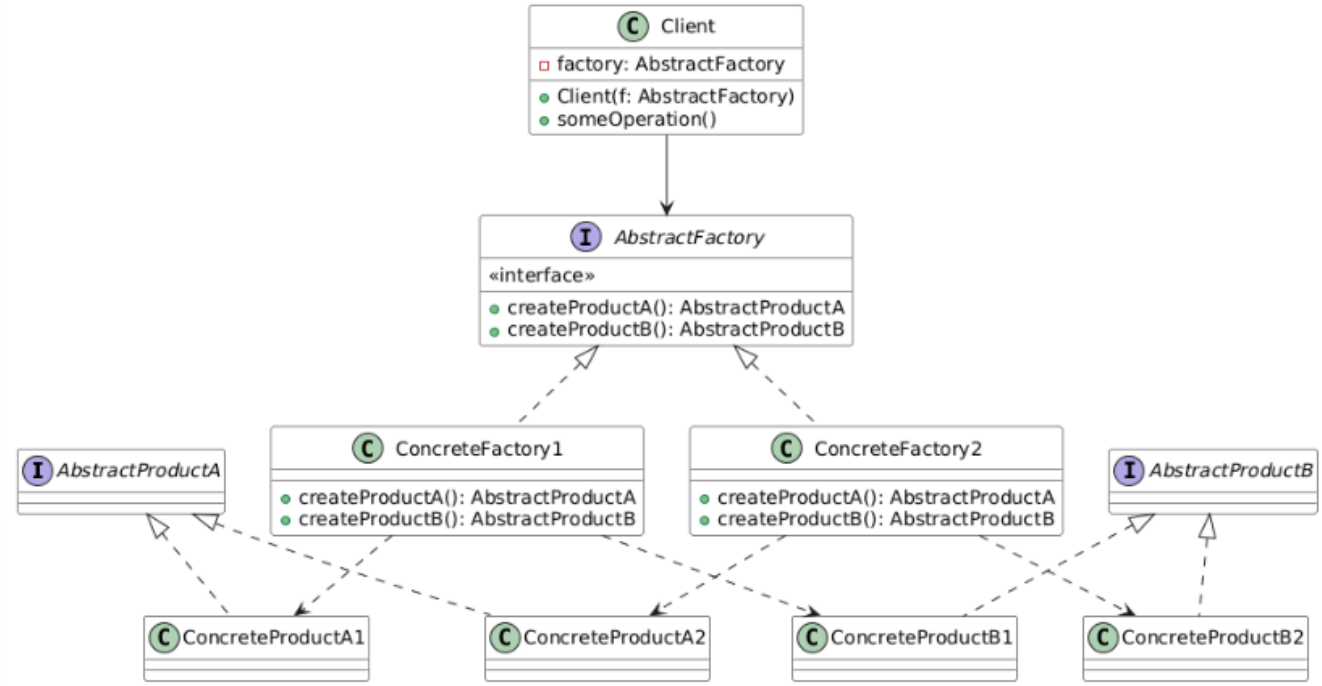
Таким чином, застосування шаблону «Декоратор» забезпечує чисту архітектуру програми, можливість легкого розширення функціональності без втручання у вже реалізований код та підвищує стабільність і зручність роботи користувача.

Відповіді на контрольні запитання:

1. Яке призначення шаблону «Абстрактна фабрика»?

Абстрактна фабрика — це породжувальний патерн проектування, що дає змогу створювати сімейства пов'язаних об'єктів, не прив'язуючись до конкретних класів створюваних об'єктів.

2. Нарисуйте структуру шаблону «Абстрактна фабрика».



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

1. **Абстрактна фабрика (AbstractFactory)**
   • Оголошує інтерфейси створення кожного продукту сімейства.
   • Взаємодія: клієнт працює з фабрикою через цей інтерфейс.
2. **Конкретна фабрика (ConcreteFactory)**
   • Реалізує створення конкретних продуктів певного сімейства.
   • Взаємодія: повертає об'єкти конкретних класів продуктів.
3. **Абстрактний продукт (AbstractProduct)**
   • Спільний інтерфейс для продуктів, що можуть бути створені фабрикою.
   • Взаємодія: клієнт працює через абстракції продуктів.
4. **Конкретний продукт (ConcreteProduct)**
   • Конкретна реалізація продукту.
   • Взаємодія: створюється конкретною фабрикою.
5. **Клієнт (Client)**
   - Використовує тільки інтерфейси, оголошені AbstractFactory та AbstractProduct. Він не знає, які конкретні класи продуктів створюються.

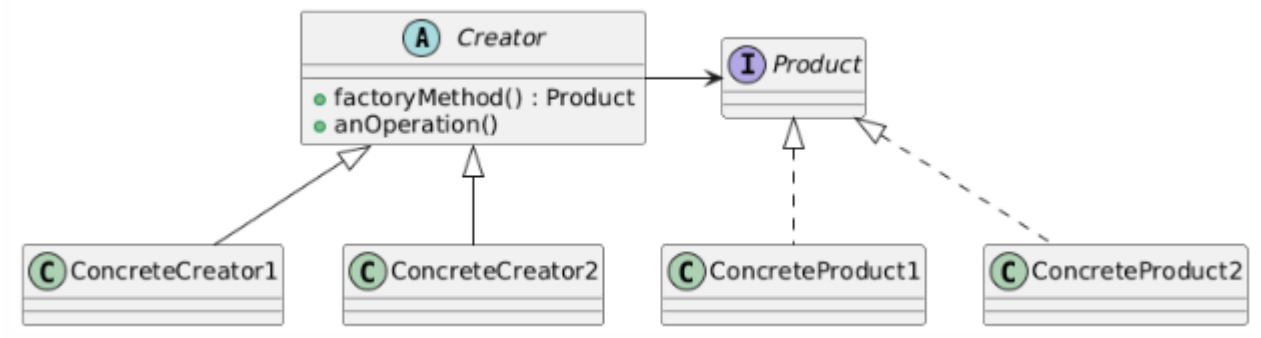
- Частина програми, яка викликає методи фабрики, наприклад, factory.createButton(), а потім викликає методи на отриманому об'єкті-кнопці, наприклад, button.render().

Взаємодія між цими класами забезпечує створення узгоджених сімейств об’єктів, не розкриваючи їх реалізацій.

4. Яке призначення шаблону «Фабричний метод»?

Фабричний метод — це породжувальний патерн проектування, який визначає загальний інтерфейс для створення об’єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об’єктів.

5. Нарисуйте структуру шаблону «Фабричний метод».



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

1. **Творець (Creator)**
   • Містить фабричний метод для створення об’єктів.
   • Може мати базову реалізацію або бути абстрактним.
2. **Конкретний творець (ConcreteCreator)**
   • Перевизначає фабричний метод.
   • Визначає, який конкретний продукт створити.
3. **Продукт (Product / AbstractProduct)**
   • Описує інтерфейс продукту, який створюється фабричним методом.
4. **Конкретний продукт (ConcreteProduct)**
   • Реалізує інтерфейс продукту.

Взаємодія: Creator викликає фабричний метод, ConcreteCreator повертає потрібний ConcreteProduct.

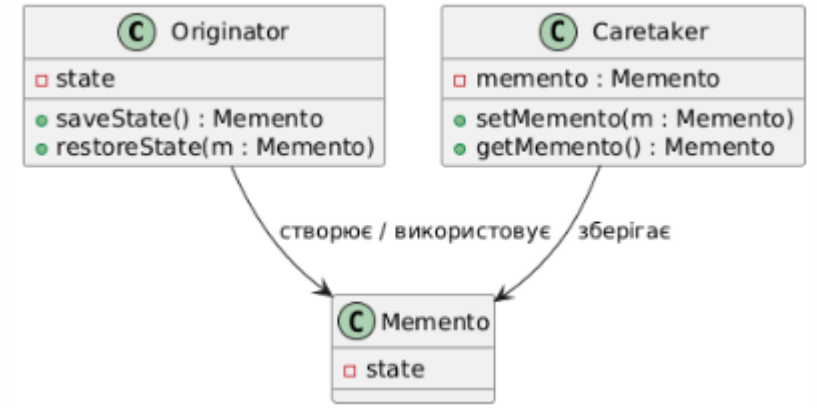
7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?

Абстрактна фабрика створює сімейства продуктів.
Фабричний метод створює один продукт через наслідування.
Абстрактна фабрика використовує композицію, фабричний метод — перевизначення методів.

8. Яке призначення шаблону «Знімок»?

Знімок — це поведінковий патерн проектування, що дає змогу зберігати та відновлювати минулий стан об’єктів, не розкриваючи подробиць їхньої реалізації.

9. Нарисуйте структуру шаблону «Знімок».



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

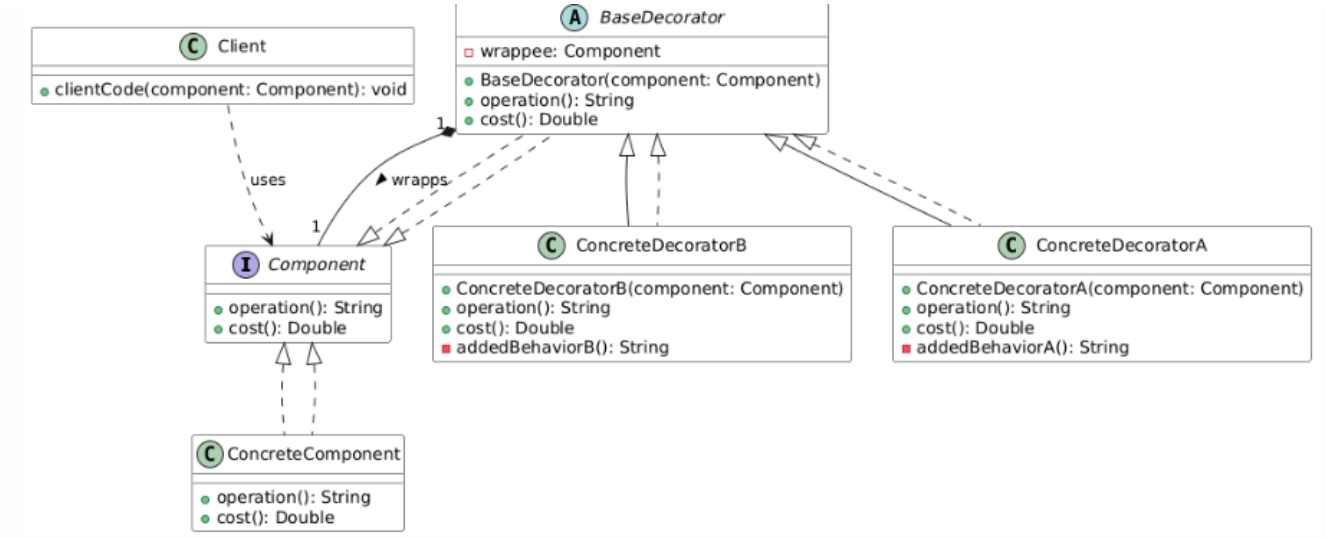
1. **Творець стану (Originator)**
   • Створює об’єкт Memento із власним станом.
   • Відновлює свій стан із Memento.
2. **Знімок (Memento)**
   • Зберігає внутрішній стан Originator.
   • Не розкриває цей стан іншим об’єктам.
3. **Опікун (Caretaker)**
   • Зберігає історію знімків.
   • Не має доступу до внутрішнього стану.

Взаємодія: Опікун керує Знімками, але лише Творець може створювати їх та відновлювати свій стан із них, забезпечуючи інкапсуляцію.

11. Яке призначення шаблону «Декоратор»?

Декоратор — це структурний патерн проектування, що дає змогу динамічно додавати об’єктам нову функціональність, загортаючи їх у корисні «обгортки».

12. Нарисуйте структуру шаблону «Декоратор».



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

1. **Компонент (Component)**
   • Базовий інтерфейс для об’єктів, до яких можна додавати функціональність.
2. **Конкретний компонент (ConcreteComponent)**
   • Об’єкт, що отримує нові можливості.
3. **Декоратор (Decorator)**
   • Містить посилання на компонент.
   • Делегує виклики базовому компоненту.
4. **Конкретний декоратор (ConcreteDecorator)**
   • Додає додаткову поведінку.

Взаємодія: декоратори обгортають компонент і викликають його методи, додаючи власну логіку.

14. Які є обмеження використання шаблону «декоратор»?

• Велика кількість вкладених декораторів ускладнює структуру.
• Діагностика та відлагодження стають складнішими.
• Порядок обгортань може змінювати поведінку об’єкта.
• Не підходить, якщо потрібна жорстка структура об’єкта.

Висновок:

У ході лабораторної роботи було розглянуто структуру та принципи функціонування шаблонів проєктування Abstract Factory, Factory Method, Memento, Observer та Decorator. Опрацьовано їх призначення, ключові компоненти та взаємодію між класами, а також особливості застосування цих патернів у різних програмних ситуаціях. Крім того, набуті практичні навички побудови UML-діаграм та роботи з породжувальними, поведінковими й структурними патернами.

У практичній частині було реалізовано патерн Decorator, що дозволив динамічно розширити функціональність об’єктів без зміни їх базового класу. Використання декораторів дало змогу гнучко додавати нові властивості та поведінку, поєднуючи різні обгортування залежно від потреб. Такий підхід підвищив модульність, розширюваність та зручність розробки програмного забезпечення.