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
«Патерни проектування»

Тема: Особиста бухгалтерія

Виконала:
студентка групи ІА-32
Коляда М. С.

Перевірила:
Мягкий М.Ю.

Київ 2025

Зміст

Тема: Патерни проектування.

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

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.

Теоретичні відомості

Шаблон «Decorator»

Призначення: 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. Простим прикладом є накладення смуги прокрутки до усіх візуальних елементів. Кожен об'єкт, який може прокручуватися, обертається в «прокручуваному» елементі, і при необхідності з'являється полоса прокрутки. Початкові функції елементу (наприклад, рядки статусу) залишаються незмінними.

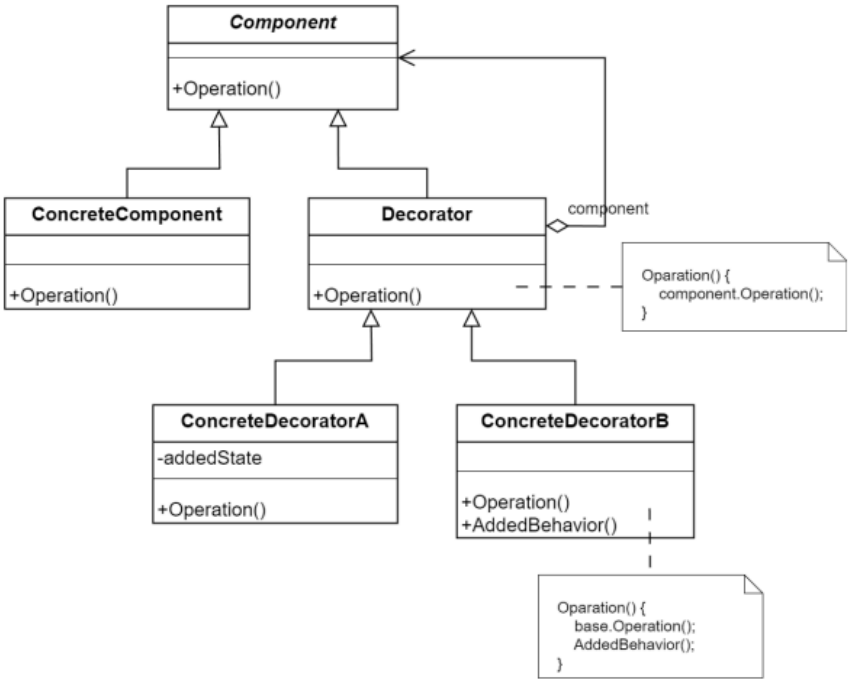


Рисунок 6.5. Структура патерну «Декоратор»

Переваги та недоліки:

+ Дозволяє мати кілька дрібних об’єктів, замість одного об’єкта «на всі випадки життя».

+ Дозволяє додавати обов’язки «на льоту».

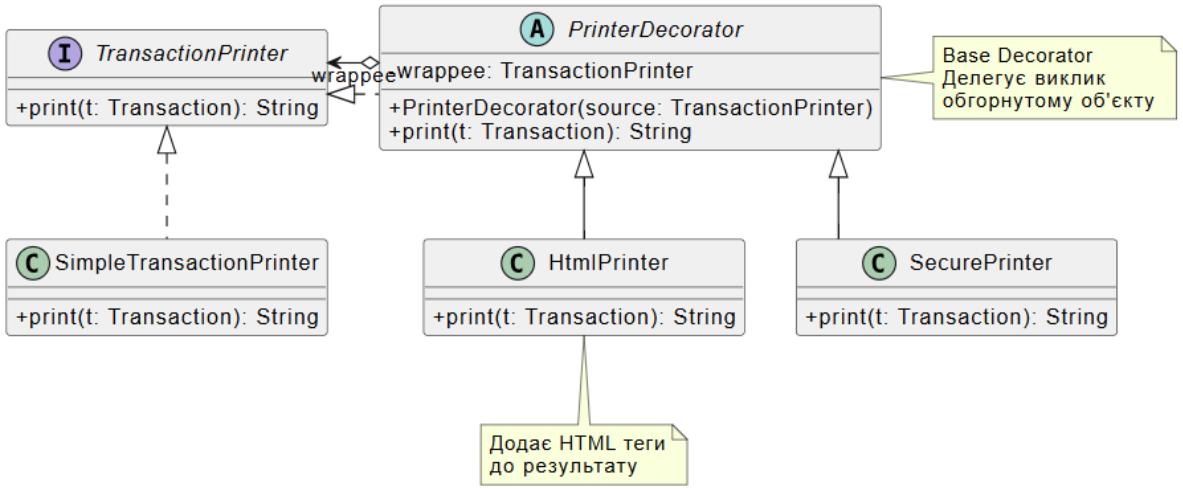
+ Більша гнучкість, ніж у спадкування.

- Велика кількість крихітних класів.

- Важко конфігурувати об’єкти, які загорнуто в декілька обгорток одночасно.

Хід роботи

Діаграма класів:



Опис реалізації патерну «Decorator»

Реалізовано структурний патерн Decorator для розширення функціональності виводу інформації про фінансові транзакції.
Метою впровадження патерну була необхідність динамічно додавати нові обов'язки об'єктам (наприклад, форматування або захист даних) без зміни їхнього коду та без використання успадкування для створення великої кількості підкласів.

Система дозволяє комбінувати різні властивості відображення:

1. Базовий вивід: Отримання "чистих" даних у текстовому форматі.
2. HTML-форматування: Обгортання даних у теги для веб-інтерфейсу.
3. Захист даних: Маскування конфіденційної інформації (суми транзакції).

Механізм роботи: Абстрактний клас PrinterDecorator реалізує той самий інтерфейс, що і базовий компонент, і містить посилання на об'єкт цього інтерфейсу (wrappee). При виклику методу print() декоратор спочатку делегує виконання вкладеному об'єкту, а потім модифікує отриманий результат (додає теги або шифрує текст). Це дозволяє створювати ланцюжки декораторів, комбінуючи їхню поведінку.

Учасники патерну:

- TransactionPrinter (Component / Компонент) Спільний інтерфейс для базових об'єктів та декораторів. Він визначає контракт (метод print), який повертає рядкове представлення транзакції. Завдяки цьому клієнтський код може працювати як з простим принтером, так і зі складною комбінацією декораторів однаково.
- SimpleTransactionPrinter (Concrete Component / Конкретний Компонент) Клас, що визначає базову поведінку. Він повертає простий рядок у форматі «Опис: Сума», який згодом може бути розширений декораторами. Це початкова ланка в ланцюжку декорування.
- PrinterDecorator (Base Decorator / Базовий Декоратор) Абстрактний клас, який зберігає посилання на вкладений об'єкт-компонент у полі wrappee. Реалізує інтерфейс компонента і за замовчуванням просто перенаправляє запит до вкладеного об'єкта. Слугує базою для всіх конкретних декораторів.
- HtmlPrinter / SecurePrinter (Concrete Decorators / Конкретні Декоратори) Класи, що додають нову функціональність.
    - HtmlPrinter: Додає HTML-теги таблиці до рядка, отриманого від вкладеного компонента.
    - SecurePrinter: Здійснює обробку рядка, замінюючи цифри символами зірочок (*) для захисту інформації на екрані.

Зв'язки: Декоратор (PrinterDecorator) містить посилання на Компонент (TransactionPrinter). Водночас, усі декоратори та конкретний компонент реалізують (implements) цей спільний інтерфейс.

Обґрунтування вибору шаблону «Decorator»

Для вирішення задачі форматування звітів було обрано саме шаблон Decorator, а не інші (наприклад, Strategy або Adapter), з наступних причин:

- Якби ми використовували звичайне успадкування, нам довелося б створити окремий клас для кожної можливої комбінації вимог: HtmlPrinter, SecurePrinter, SecureHtmlPrinter (захищений HTML), SimpleSecurePrinter (захищений текст) тощо. Декоратор дозволяє отримати SecureHtmlPrinter, просто обгорнувши один об'єкт в інший під час виконання програми: new SecurePrinter(new HtmlPrinter(...)).
- На відміну від патерну *Strategy*, який повністю замінює алгоритм (один на інший), Декоратор доповнює результат роботи об'єкта. Нам потрібно було зберегти базовий текст транзакції, але "нашаровувати" на нього нові властивості (теги, маскування), що є прямою відповідальністю Декоратора.

Фрагменти програмного коду:

TransactionPrinter.java

```
package ia32.koliada.finance.decorator;

import ia32.koliada.finance.entity.Transaction;

public interface TransactionPrinter {
   String print(Transaction transaction);
}
```

SimpleTransactionPrinter.java

```
package ia32.koliada.finance.decorator;

import ia32.koliada.finance.entity.Transaction;

public class SimpleTransactionPrinter implements TransactionPrinter {
   @Override
   public String print(Transaction transaction) {
      // Базовий вивід: "Опис: сума"
      return transaction.getDescription() + ": " + transaction.getAmount() + "
грн";
```

```
    }
}
```

PrinterDecorator.java

```
package ia32.koliada.finance.decorator;

import ia32.koliada.finance.entity.Transaction;

public abstract class PrinterDecorator implements TransactionPrinter {
    protected TransactionPrinter wrappee;
    public PrinterDecorator(TransactionPrinter wrappee) {
        this.wrappee = wrappee;
    }

    @Override
    public String print(Transaction transaction) {
        return wrappee.print(transaction);
    }
}
```

HtmlPrinter.java

```
package ia32.koliada.finance.decorator;

import ia32.koliada.finance.entity.Transaction;

public class HtmlPrinter extends PrinterDecorator {

    public HtmlPrinter(TransactionPrinter wrappee) {
        super(wrappee);
    }

    @Override
    public String print(Transaction transaction) {
            return "<tr><td>" + super.print(transaction) + "</td></tr>";
    }
}
```

SecurePrinter.java

```
package ia32.koliada.finance.decorator;

import ia32.koliada.finance.entity.Transaction;

public class SecurePrinter extends PrinterDecorator {

    public SecurePrinter(TransactionPrinter wrappee) {
        super(wrappee);
    }

    @Override
    public String print(Transaction transaction) {
        String original = super.print(transaction);
            return "[SECURE] " + original.replaceAll("\\d", "*");
    }
}
```

Вихідний код:

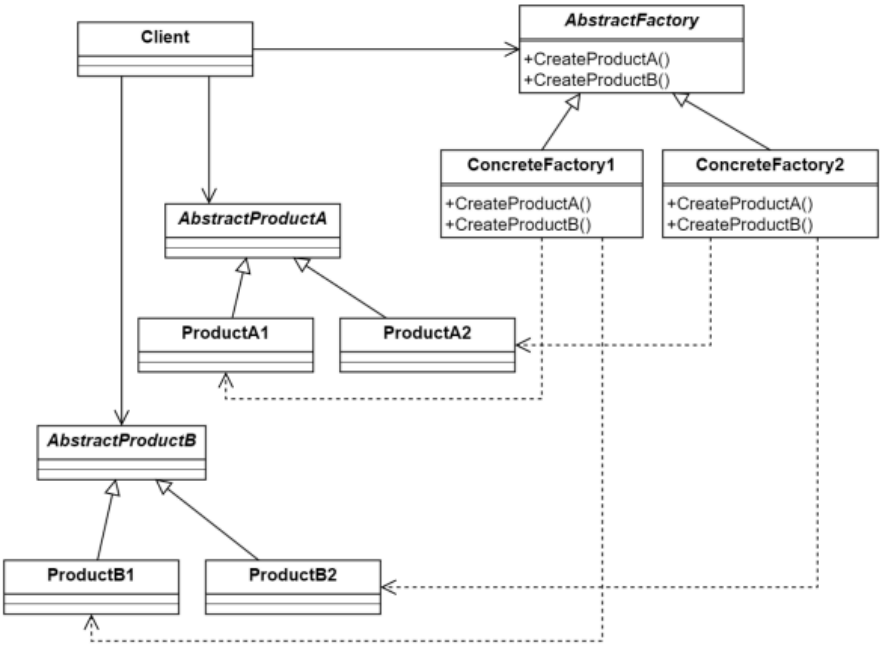
https://github.com/mariiakoliada/TRPZ/tree/master/src/ia32/koliada/finance/decorator

Питання до лабораторної роботи

1. Яке призначення шаблону «Абстрактна фабрика»?

Шаблон надає інтерфейс для створення сімейств взаємопов'язаних або залежних об'єктів, не специфікуючи їхніх конкретних класів.

2. Нарисуйте структуру шаблону «Абстрактна фабрика».



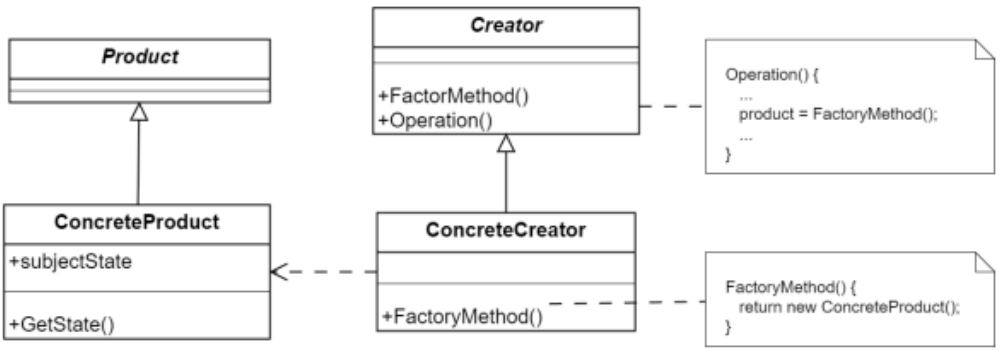
3. Які класи входять в шаблон «Абстрактна фабрика», та яка між ними взаємодія?

До складу входять: AbstractFactory (оголошує методи створення), ConcreteFactory (реалізує створення продуктів), AbstractProduct та ConcreteProduct (сімейства об'єктів), Client. Взаємодія полягає в тому, що Клієнт звертається до інтерфейсу фабрики, а Конкретна фабрика створює і повертає екземпляри Конкретних продуктів.

4. Яке призначення шаблону «Фабричний метод»?

Шаблон визначає загальний інтерфейс для створення об'єкта, але дозволяє підкласам змінювати тип створюваного об'єкта, делегуючи їм процес інстанціювання.

5. Нарисуйте структуру шаблону «Фабричний метод».



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

До складу входять: Product (інтерфейс), ConcreteProduct (реалізація), Creator (оголошує метод створення), ConcreteCreator (перевизначає метод). Взаємодія: ConcreteCreator реалізує фабричний метод так, щоб він повертав новий екземпляр ConcreteProduct.

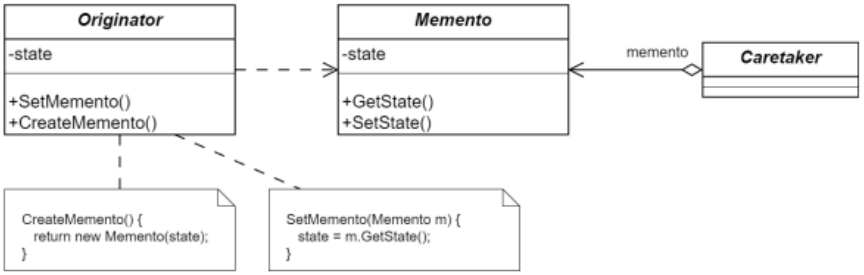
7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?

Фабричний метод базується на наслідуванні (створення делегується підкласам) і створює один продукт. Абстрактна фабрика базується на композиції (створення делегується об'єкту-фабриці) і призначена для створення цілих сімейств продуктів.

8. Яке призначення шаблону «Знімок»?

Шаблон дозволяє зафіксувати і зберегти внутрішній стан об'єкта так, щоб не порушувати інкапсуляцію, і пізніше відновити об'єкт у цьому стані.

9. Нарисуйте структуру шаблону «Знімок».



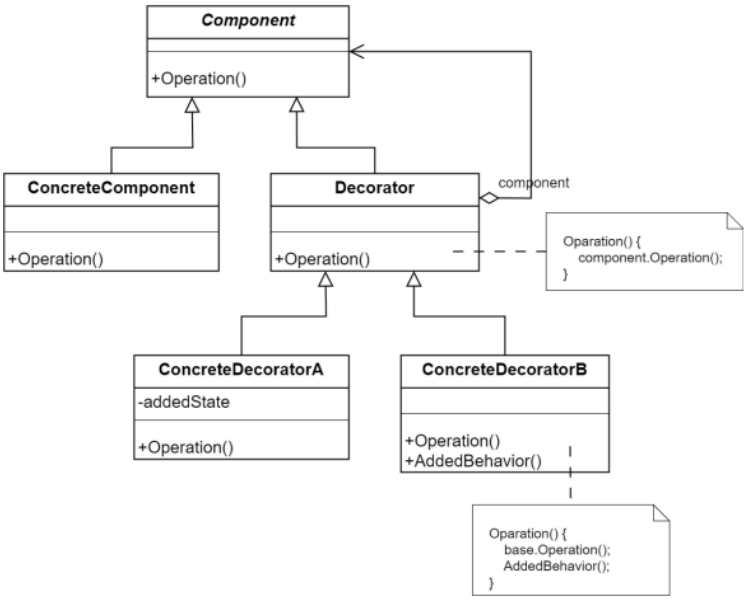
10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія?

До складу входять: Originator (власник стану), Memento (контейнер стану), Caretaker (зберігач). Взаємодія: Originator створює Memento, копіюючи туди свій стан, і віддає його Caretaker. Для відновлення Caretaker повертає Memento, і Originator витягує з нього стан.

11. Яке призначення шаблону «Декоратор»?

Шаблон дозволяє динамічно додавати об'єктам нову функціональність, загортаючи їх у спеціальні об'єкти-обгортки (декоратори), що є гнучкою альтернативою розширенню класів через наслідування.

12. Нарисуйте структуру шаблону «Декоратор».



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

До складу входять: Component (спільний інтерфейс), ConcreteComponent (основний об'єкт), Decorator (базова обгортка), ConcreteDecorator (додаткова поведінка). Взаємодія: Декоратор містить посилання на об'єкт Компонента; він виконує свою додаткову дію і викликає метод вкладеного об'єкта.

14. Які є обмеження використання шаблону «декоратор»?

Важко вилучити конкретну оболонку з середини стеку декораторів; порядок накладання декораторів може впливати на результат; код ускладнюється великою кількістю маленьких класів; процес конфігурації об'єкта стає складнішим.

Висновок: під час виконання лабораторної роботи, було вивчено призначення та структуру структурних шаблонів проєктування, зокрема

патернів «Adapter», «Bridge», «Composite», «Decorator», «Facade», «Flyweight» та «Proxy». Набуто практичних навичок їх застосування для покращення архітектури програмного забезпечення. На прикладі системи було успішно реалізовано структурний шаблон «Decorator» (Декоратор). Це дозволило створити гнучку підсистему відображення фінансових звітів, забезпечивши можливість динамічного додавання нових обов'язків об'єктам без необхідності створення великої кількості підкласів.