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»

«Web crawler»

Виконала
студентка групи – ІА–31
Горлач Дар’я Дмитрівна

Перевірив:
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 проєкту: Web crawler (proxy, chain of responsibility, memento, template method, composite, p2p) Веб–сканер повинен вміти розпізнавати структуру сторінок сайту, переходити за посиланнями, збирати необхідну інформацію про зазначений термін, видаляти не семантичні одиниці (рекламу, об'єкти javascript і т.д.), зберігати знайдені дані у вигляді структурованого набору html файлів вести статистику відвіданих сайтів і метадані.

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.

Зміст

Теоретичні відомості

Шаблон «Memento»

Призначення: 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.
Таким чином вдається досягти наступних цілей:

- зберігання стану повністю відділяється від початкових об'єктів, що полегшує їх реалізацію;
- передача об'єктів «Memento» лягає на плечі Caretaker об'єктів, що дозволяє гнучкіше управляти станами об'єктів і спростити дизайн класів початкових об'єктів;
- збереження і відновлення стану реалізовані у вигляді двох простих методів і є закритими для кого–небудь ще окрім початкових об'єктів, таким чином не порушуючи інкапсуляцію.

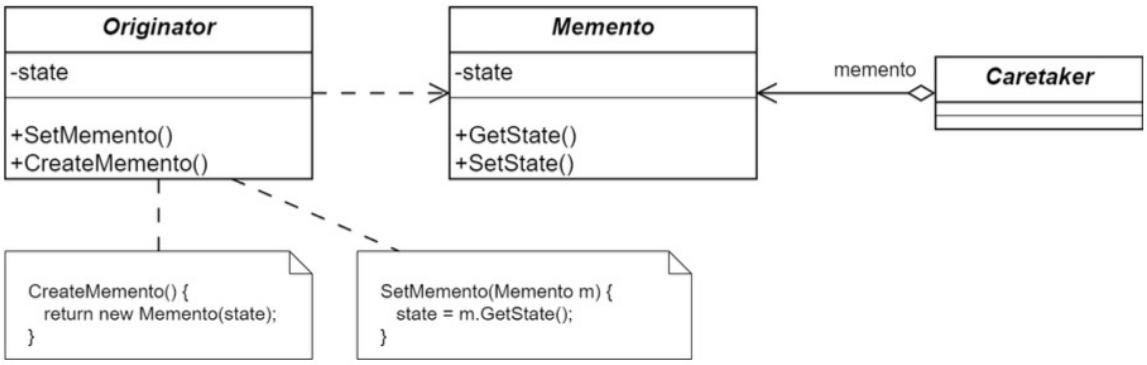


Рисунок 1. Структура патерну «Знімок»

Шаблон «Мементо» дуже зручно використати разом з шаблоном «Команда» для реалізації «скасовних» дій – дані про дію зберігаються в мементо, а команда має можливість вважати і відновити початкове положення відповідних об'єктів.
Переваги та недоліки:
+ Не порушує інкапсуляцію вихідного об’єкта.
+ Спрощує структуру вихідного об’єкта. Не потрібно зберігати історію версій свого стану.
– Вимагає багато пам’яті, якщо клієнти дуже часто створюють знімки.
– Може спричинити додаткові витрати пам’яті, якщо об’єкти, що зберігають історію, не звільняють ресурси, зайняті застарілими знімками.

Хід роботи

Реалізація патерну проєктування

Для реалізації Web Crawler використано патерн проєктування Memento, оскільки він забезпечує можливість зберігання та відновлення стану веб–сторінок під час обробки, що дозволяє реалізувати механізм скасування змін та відстеження історії трансформацій контенту. Це особливо важливо

для веб–краулера, де необхідно аналізувати та порівнювати різні версії оброблених сторінок.

Патерн Memento реалізовано у класах PageContext.Memento, PageHistory та інтегровано з ланцюжком обробників, що забезпечує збереження стану HTML–контенту на кожному етапі обробки. Такий підхід дозволяє відстежувати та відновлювати проміжні стани сторінок під час складних трансформацій.

```
public class PageContext {
    5 usages
    private String htmlContent;
    2 usages
    private final String keyword;

    1 usage  dariahorlach
    public PageContext(String htmlContent, String keyword) {
        this.htmlContent = htmlContent;
        this.keyword = keyword;
    }

    4 usages  dariahorlach
    public String getHtmlContent() { return htmlContent; }

    2 usages  dariahorlach
    public void setHtmlContent(String htmlContent) { this.htmlContent = htmlContent; }

    2 usages  dariahorlach
    public String getKeyword() { return keyword; }

    1 usage  dariahorlach
    public Memento save() { return new Memento(htmlContent); }

    no usages  dariahorlach
    public void restore(Memento memento) { this.htmlContent = memento.htmlContent; }

    6 usages  dariahorlach
    public static class Memento {
        2 usages
        private final String htmlContent;

        1 usage  dariahorlach
        private Memento(String htmlContent) { this.htmlContent = htmlContent; }
    }
}
```

Рис. 2 – Код класу PageContext

```
public class PageHistory {
    3 usages
    private final Stack<PageContext.Memento> history = new Stack<>();

    1 usage   dariahorlach
    public void save(PageContext.Memento m) {
        history.push(m);
    }

    no usages   dariahorlach
    public PageContext.Memento undo() {
        if (!history.isEmpty()) {
            return history.pop();
        }
        return null;
    }
}
```

Рис. 3 – Код класу PageHistory

Використання цього патерну надає:

1. Механізм скасування змін: Клас PageHistory зберігає стек моментумів, що дозволяє відновлювати попередні стани HTML–контенту через метод undo(). Це критично важливо для відладки та аналізу ефективності різних обробників.
2. Інкапсуляцію стану: Внутрішній клас Memento інкапсулює стан HTML–контенту, забезпечуючи контрольований доступ до даних збереження та відновлення стану.
3. Історію трансформацій: Кожен обробник у ланцюжку зберігає стан перед виконанням своїх змін через метод saveState(context), створюючи детальний журнал усіх модифікацій сторінки.
4. Інтеграцію з Chain of Responsibility: Патерн Memento ідеально інтегрується з ланцюжком обробників, де кожен етап може зберігати свій проміжний стан для подальшого аналізу або відновлення.

У нашому випадку патерн Memento забезпечує надійний контроль над процесом обробки веб–сторінок, що дозволяє аналізувати ефект кожного обробника окремо, відновлювати сторінку до будь–якого попереднього стану та в подальшому реалізувати безпечне тестування нових обробників з можливістю відкату змін.

Структура патерну

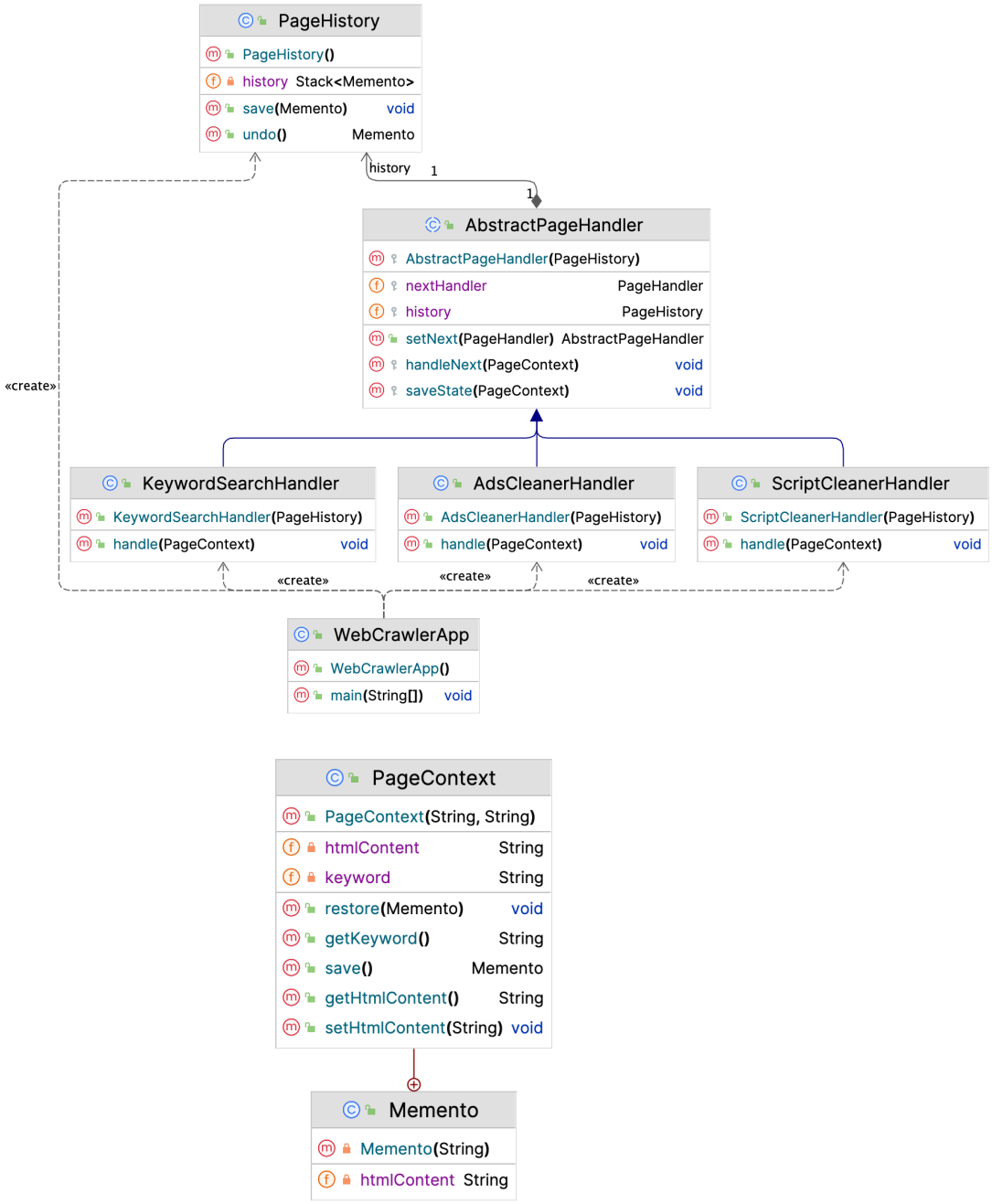


Рис. 4 – Структура патерну «Знімок»

Опис реалізації:

- PageContext.Memento – внутрішній клас, що зберігає стан HTML–контенту

- PageHistory – управляє історією змін через стек моментумів

- AbstractPageHandler – базовий клас обробників, що інтегрує механізм збереження стану

- save() та restore() методи – забезпечують серіалізацію та десеріалізацію стану

Такий підхід особливо ефективний для веб–краулерів, де необхідно відстежувати еволюцію контенту через послідовність обробок, аналізувати ефективність різних алгоритмів очищення та трансформації, надавати можливість користувачу відкатити невдалі зміни або зберігати проміжні результати для подальшого порівняльного аналізу.

Посилання на репозиторій: https://github.com/dariahorlach/Lab–TRPZ

Висновки

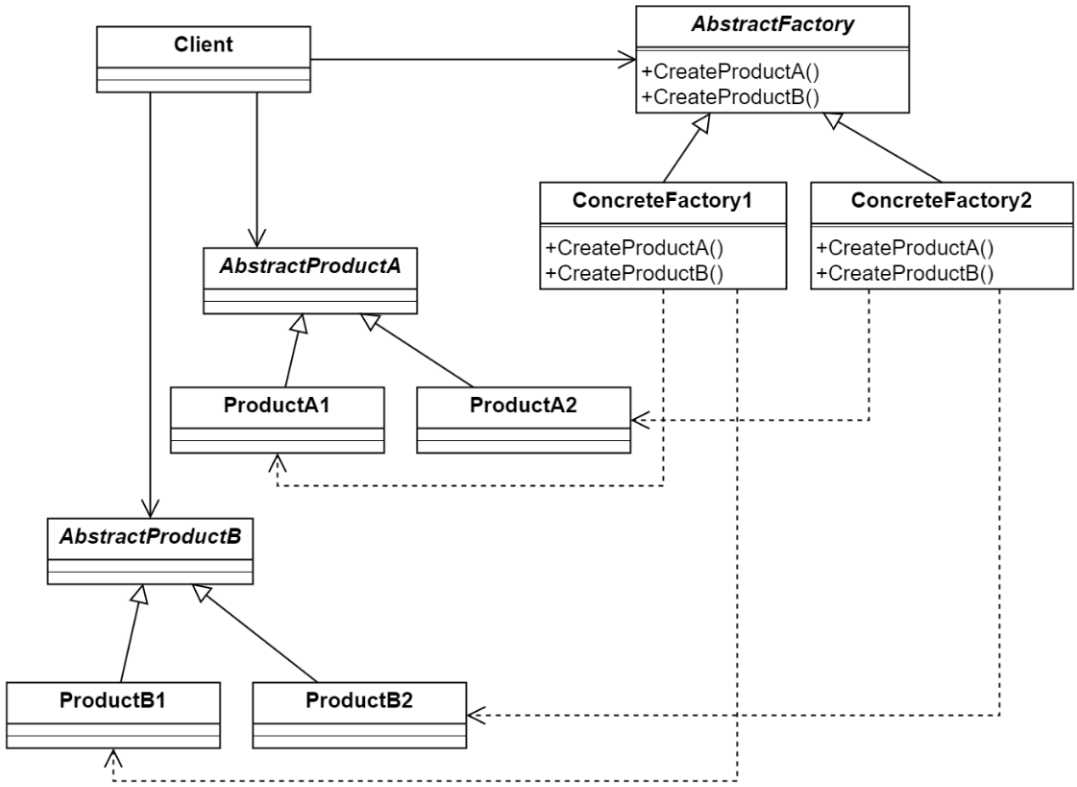
Висновки: під час виконання лабораторної роботи, було реалізовано патерн Memento для веб–краулера, що дозволило створити надійний механізм збереження та відновлення стану веб–сторінок під час обробки. Клас PageHistory зі стеком моментумів забезпечив можливість відстежувати повну історію трансформацій HTML–контенту та реалізувати функціонал скасування змін. Реалізація продемонструвала, що патерн Memento ефективно вирішує завдання контролю над складним процесом обробки веб–сторінок шляхом інкапсуляції стану HTML–контенту у внутрішньому класі. Архітектура з Memento дозволила створити веб–краулер з розвиненими аналітичними можливостями, де кожна трансформація контенту може бути проаналізована, порівняна з попередніми станами та при необхідності скасована. Інтеграція з ланцюжком обробників забезпечила автоматичне збереження стану перед кожною модифікацією, створюючи повний журнал змін.

Питання до лабораторної роботи

1. Яке призначення шаблону «Абстрактна фабрика»?

Шаблон «Абстрактна фабрика» (Abstract Factory) забезпечує створення сімейств взаємопов’язаних об’єктів без зазначення їх конкретних класів.

2. Нарисуйте структуру шаблону «Абстрактна фабрика».



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

AbstractFactory – інтерфейс із методами створення об’єктів.

ConcreteFactory – створює конкретні варіанти продуктів.

AbstractProduct – інтерфейс продукту.

ConcreteProduct – конкретна реалізація продукту.

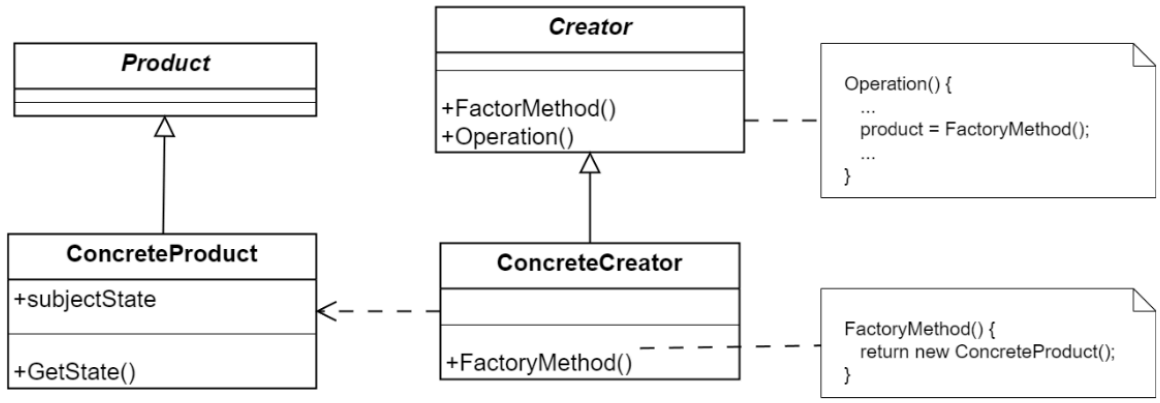
Client – використовує об’єкти, створені фабрикою.

Client звертається до AbstractFactory для створення об’єктів, не знаючи, які конкретні класи створюються.

4. Яке призначення шаблону «Фабричний метод»?

Шаблон «Фабричний метод» (Factory Method) визначає інтерфейс для створення об’єктів, але дозволяє підкласам вирішувати, який клас створювати.

5. Нарисуйте структуру шаблону «Фабричний метод».



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

Creator – оголошує метод factoryMethod(), який повертає об’єкт типу Product.

ConcreteCreator – перевизначає метод, створюючи конкретний продукт.

Product – інтерфейс або базовий клас для продуктів.

ConcreteProduct – конкретна реалізація продукту.

Creator викликає factoryMethod() для створення продукту, а конкретний підклас визначає, який саме продукт створити.

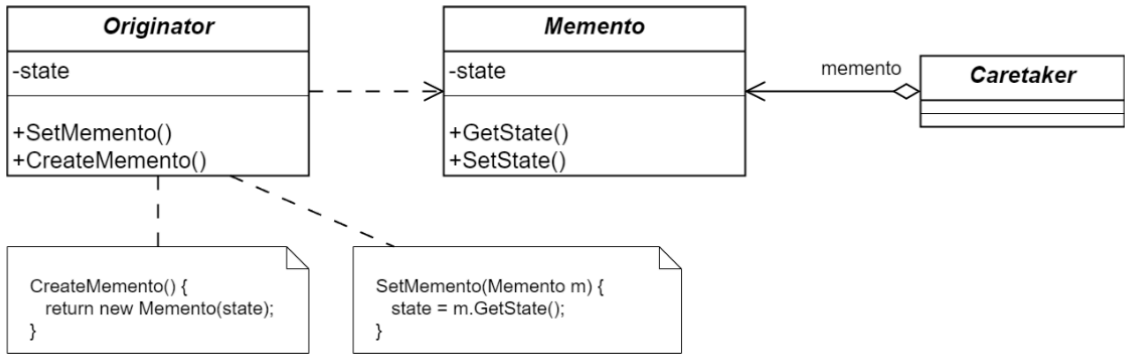
7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?

Абстрактна фабрика створює сімейства пов’язаних об’єктів, Фабричний метод у свою чергу, створює один тип об’єкта. Абстрактна фабрика містить кілька фабричних методів, а Фабричний метод є лише складовою абстрактної фабрики, тому Абстрактна фабрика має вищий рівень абстракції. Абстрактну фабрику краще використовувати, коли потрібна узгоджена група продуктів, а Фабричний метод, коли створюється один об’єкт через підкласи.

8. Яке призначення шаблону «Знімок»?

Шаблон «Знімок» (Memento) зберігає внутрішній стан об’єкта, щоб потім можна було відновити його без порушення інкапсуляції.

9. Нарисуйте структуру шаблону «Знімок».



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

Originator – створює знімок власного стану й може його відновити.

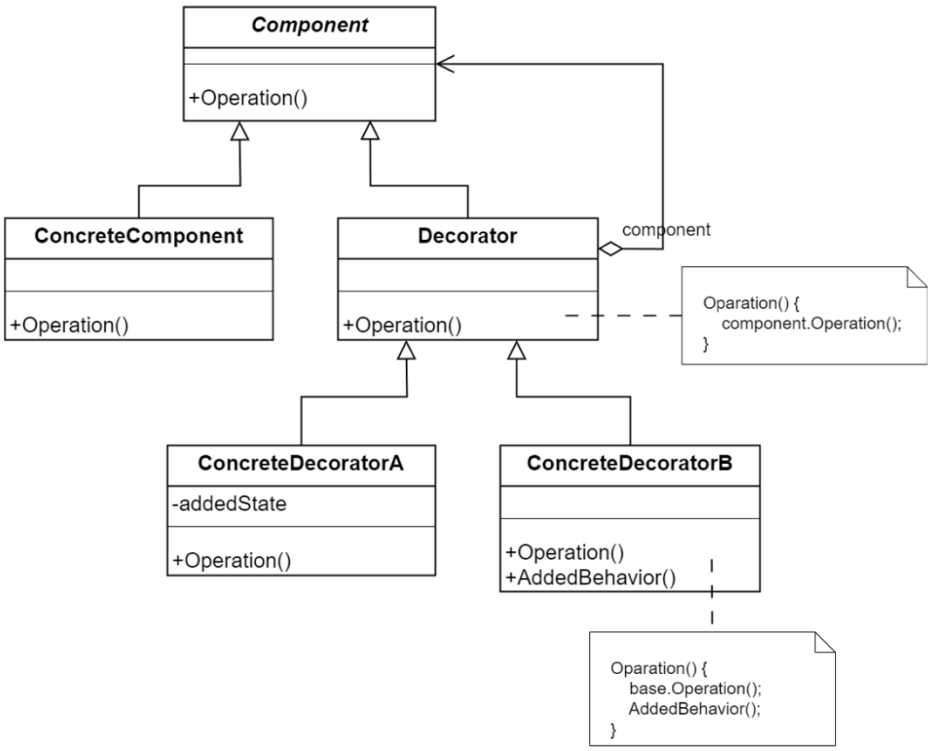
Memento – об’єкт, що зберігає стан Originator.

Caretaker – зберігає Memento, але не змінює його.

Originator створює Memento, Caretaker його зберігає, а потім може передати назад для відновлення стану.

11. Яке призначення шаблону «Декоратор»?

Шаблон «Декоратор» (Decorator) дозволяє динамічно додавати об’єктам нову функціональність без зміни їхнього коду або створення підкласів.
12. Нарисуйте структуру шаблону «Декоратор».



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

Component – базовий інтерфейс об’єкта.

ConcreteComponent – об’єкт, до якого додаються нові функції.

Decorator – зберігає посилання на компонент і реалізує той самий інтерфейс.

ConcreteDecorator – додає нову поведінку до компонента.

Decorator викликає метод базового компонента й додає додаткову поведінку до або після нього.

14. Які є обмеження використання шаблону «декоратор»?

Основні обмеження використання шаблону «декоратор» включають:

- Велика кількість дрібних класів може ускладнити структуру програми.

- Складно відлагоджувати через вкладеність декораторів.

- Порядок застосування декораторів може впливати на результат.
- Декоратори не завжди легко поєднуються між собою.