Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №6

Технології розроблення програмного забезпечення

«*Патерни проектування. Паттерн «Factory Method»*»

«10. VCS all-in-one»

Виконав
студент групи ІА–32:
Варивода К. С.

Перевірив:

Мягкий Михайло Юрійович

Київ 2025

Зміст

Варіант

10. VCS all-in-one (iterator, adapter, factory method, facade, visitor, p2p)

Клієнт для всіх систем контролю версій повинен підтримувати основні команди і дії (commit, update, push, pull, fetch, list, log, patch, branch, merge, tag) для 3-х основних систем управління версіями (svn, git, mercurial), а також мати можливість вести реєстр репозиторіїв (і їх типів) і відображати дерева фіксації графічно.

Теоретичні відомості

Будь-який патерн проєктування, використовуваний при розробці інформаційних систем, являє собою формалізований опис, який часто зустрічається в завданнях проєктування, вдале рішення даної задачі, а також рекомендації по застосуванню цього рішення в різних ситуаціях. Крім того, патерн проєктування обов'язково має загальновживане найменування. Правильно сформульований патерн проєктування дозволяє, відшукавши одного разу вдале рішення, користуватися ним знову і знову.

Призначення: 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».

Розглянемо простий приклад. Нехай наш застосунок працює з мережевими драйвер-мі і використовує клас Packet для зберігання даних, що передаються в мережу. Залежно від використовуваного протоколу, існує два перевантаження – TcpPacket, UdpPacket. І відповідно два створюючі об'єкти (TcpCreator, UdpCreator) з фабричним методом (який створює відповідні реалізації).
Проте базова функціональність (передача пакету, прийом пакету, заповнення пакету даними) нічим не відрізняється один від одного, відповідно поміщається у базовий клас PacketCreator. Таким чином поведінка системи залишається тим же, проте з'являється можливість підстановки власних об'єктів в процес створення і роботи з пакетами.
Переваги та недоліки:
+ Позбавляє клас від прив'язки до конкретних класів продуктів.
+ Виділяє код виробництва продуктів в одне місце, спрощуючи підтримку коду.
+ Спрощує додавання нових продуктів до програми.
- Може призвести до створення великих паралельних ієрархій класів.

Релевантність до проекту:

В моєму проекті цей патерн є корисним, так як він дає можливість приховати логіку створення класів за фабрік класом, який займається цим «під капотом». В цьому проекті я використовую «Factory Method» для отримання різноманітних класів DAO для роботи з базою даних, тобто я створюю лише 1 клас фабрики з якого викликаю необхідні класи, замість явного створення цих класів, яких може бути дуже багато.

Хід роботи

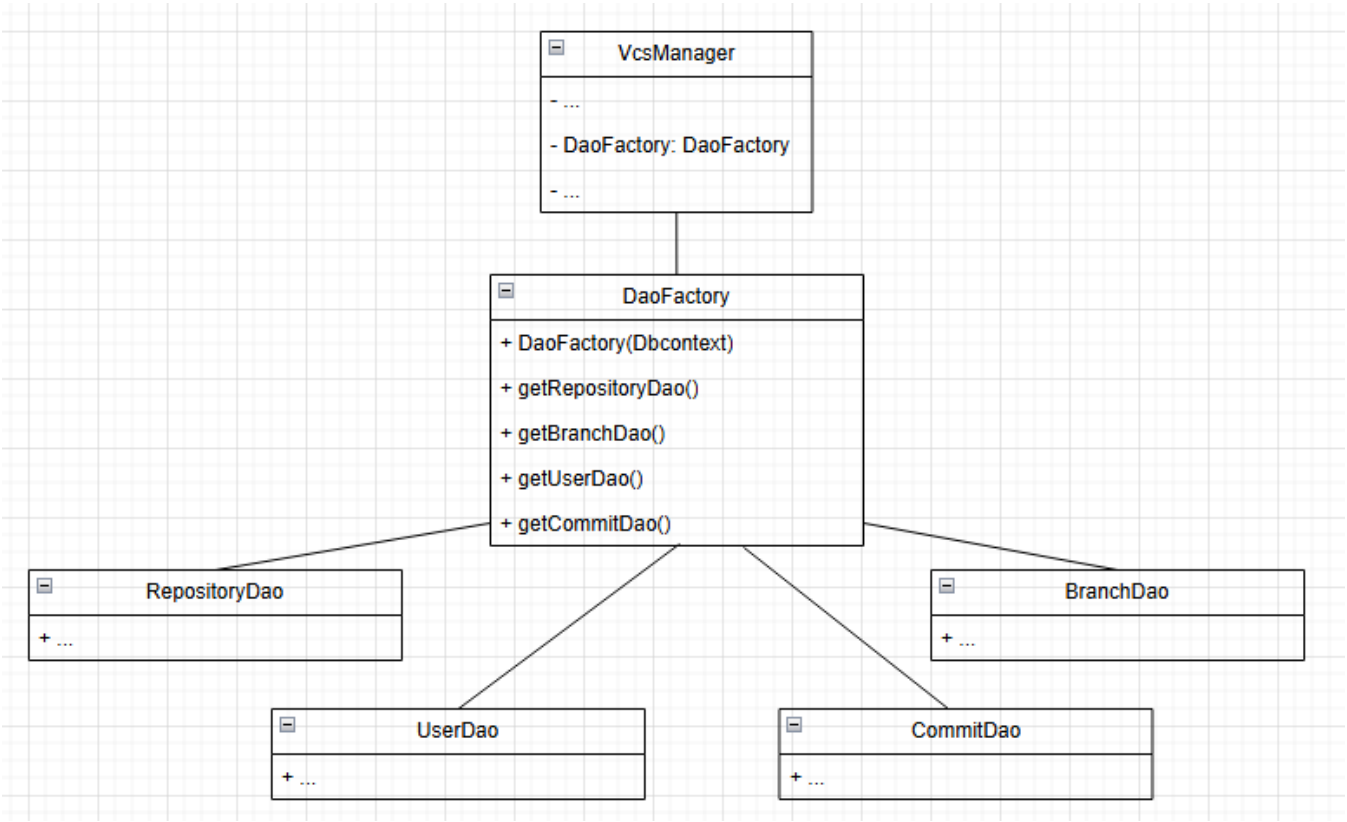


Рис 1 – Діаграма класів модулю з використанням паттерну «Фабричний метод»

```java
package vcs.repository.factories;

import vcs.repository.dao.db.DatabaseContext;
import vcs.repository.dao.impl.BranchDao;
import vcs.repository.dao.impl.CommitDao;
import vcs.repository.dao.impl.RepositoryDao;
import vcs.repository.dao.impl.UserDao;

public class DaoFactory {  3 usages  mavr1iq
    private final DatabaseContext dbContext;  5 usages

    public DaoFactory(DatabaseContext dbContext) { this.dbContext = dbContext; }

    public RepositoryDao getRepositoryDao() { return new RepositoryDao(dbContext); }
    public BranchDao getBranchDao() { return new BranchDao(dbContext); }
    public CommitDao getCommitDao() { return new CommitDao(dbContext); }
    public UserDao getUserDao() { return new UserDao(dbContext); }
}
```

Рис 2 – клас DaoFactory

```java
package vcs.repository.factories;

import vcs.repository.VcsType;
import vcs.repository.adapters.GitAdapter;
import vcs.repository.adapters.MercurialAdapter;
import vcs.repository.adapters.SvnAdapter;
import vcs.repository.adapters.VcsAdapter;

public class AdapterFactory {  3 usages  mavr1iq

    public VcsAdapter getAdapter(VcsType type) {  2 usages  mavr1iq
        if (type == null) {
            throw new IllegalArgumentException("VCS type cannot be null");
        }

        return switch (type) {
            case GIT -> new GitAdapter();
            case SVN -> new SvnAdapter();
            case MERCURIAL -> new MercurialAdapter();
            default -> throw new IllegalArgumentException("Unknown VCS type: " + type);
        };
    }
}
```

Рис 3 - клас AdapterFactory

Висновок

Отже, після виконання даної лабораторної роботи було вивчено паттерн програмування «Фабричний метод», його сильні та слабкі сторони, тонкощі імплементації та use кейси. Для закріплення отриманих знань було розроблено паттерн у існуючому проекті. Було проведено аналіз потреб проекту та визначено потребу в імплементації вказаного патерну. Було виконано 2 можливі реалізації – через case з передаванням enum значення типу адаптера та через виклик методу. Під час виконання було досліджено разючу різницю у чистоті коду з використанням фабричного методу, так як він дозволяє приховати деталі створення класів за «обгорткою».

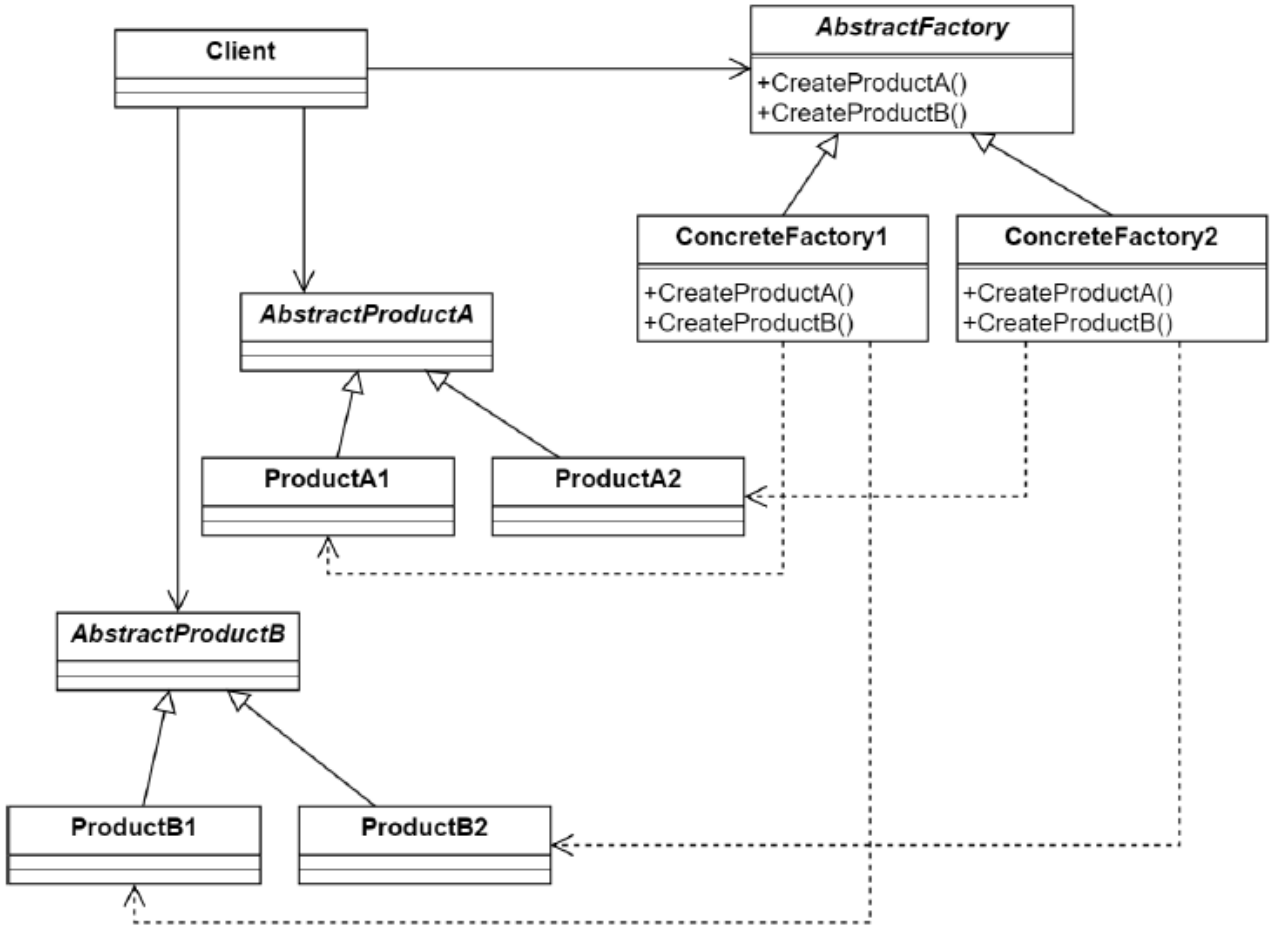
Відповіді на питання

1. Яке призначення шаблону «Абстрактна фабрика»?

Призначення шаблону «Абстрактна фабрика» (Abstract Factory) — надати інтерфейс для створення сімейств взаємопов'язаних об'єктів, не вказуючи їхні конкретні класи.

Це дозволяє створювати набори об'єктів, які гарантовано сумісні один з одним і легко замінювати все сімейство одразу.

2. Нарисуйте структуру шаблону «Абстрактна фабрика».



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

AbstractFactory (Абстрактна фабрика): Інтерфейс, що оголошує методи для створення кожного продукту в сімействі.

ConcreteFactory (Конкретна фабрика): Клас, що реалізує інтерфейс AbstractFactory. Він знає, як створити конкретну версію кожного продукту

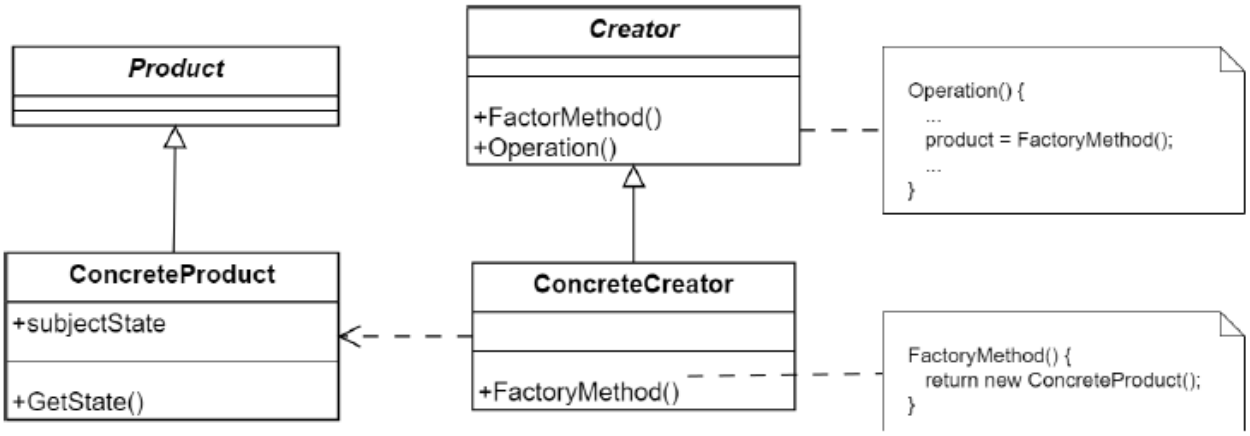
AbstractProduct (Абстрактний продукт): Загальний інтерфейс для типу продукту

ConcreteProduct (Конкретний продукт): Клас, що реалізує AbstractProduct.

Взаємодія: Клієнтський код отримує екземпляр ConcreteFactory і використовує його для створення продуктів. Клієнт працює з продуктами через їхні абстрактні інтерфейси, не знаючи, що отримав саме WinButton.

4. Яке призначення шаблону «Фабричний метод»?
Призначення шаблону «Фабричний метод» (Factory Method) — визначити інтерфейс для створення одного об'єкта, але дозволити підкласам вирішувати, екземпляр якого саме класу створювати. Він делегує логіку створення об'єкта підкласам.

5. Нарисуйте структуру шаблону «Фабричний метод».



6. Які класи входять в шаблон «Фабричний метод», та яка між ними взаємодія?
Product (Продукт): Інтерфейс для об'єкта, що створюється.
ConcreteProduct (Конкретний продукт): Клас, що реалізує інтерфейс Product.
Creator (Творець): Абстрактний клас або інтерфейс, що містить "фабричний метод"
ConcreteCreator (Конкретний творець): Клас, що успадковує Creator та реалізує createProduct(), повертаючи екземпляр ConcreteProduct.

Взаємодія: Клієнт викликає фабричний метод у об'єкта ConcreteCreator. ConcreteCreator створює ConcreteProduct і повертає його клієнту під виглядом Product.

7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?
Масштаб: «Фабричний метод» створює один продукт. «Абстрактна фабрика» створює сімейство (набір) пов'язаних продуктів.

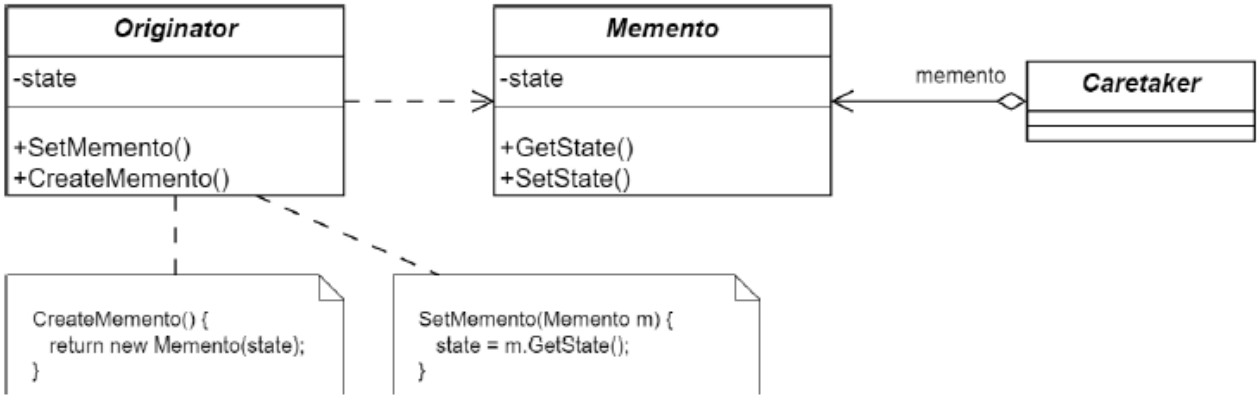
Спосіб реалізації: «Фабричний метод» — це метод, який реалізується в підкласах. «Абстрактна фабрика» — це об'єкт з багатьма фабричними методами.

Призначення: «Фабричний метод» дозволяє підкласам змінювати тип об'єкта, що створюється. «Абстрактна фабрика» дозволяє створювати групи сумісних об'єктів.

8. Яке призначення шаблону «Знімок»?
Призначення шаблону «Знімок» (Memento) — дозволити зберігати та відновлювати внутрішній стан об'єкта, не порушуючи при цьому його інкапсуляцію (тобто не

розкриваючи його внутрішню структуру для зовнішнього світу). Це основний патерн для реалізації операцій "скасувати" (Undo).

9. Нарисуйте структуру шаблону «Знімок».

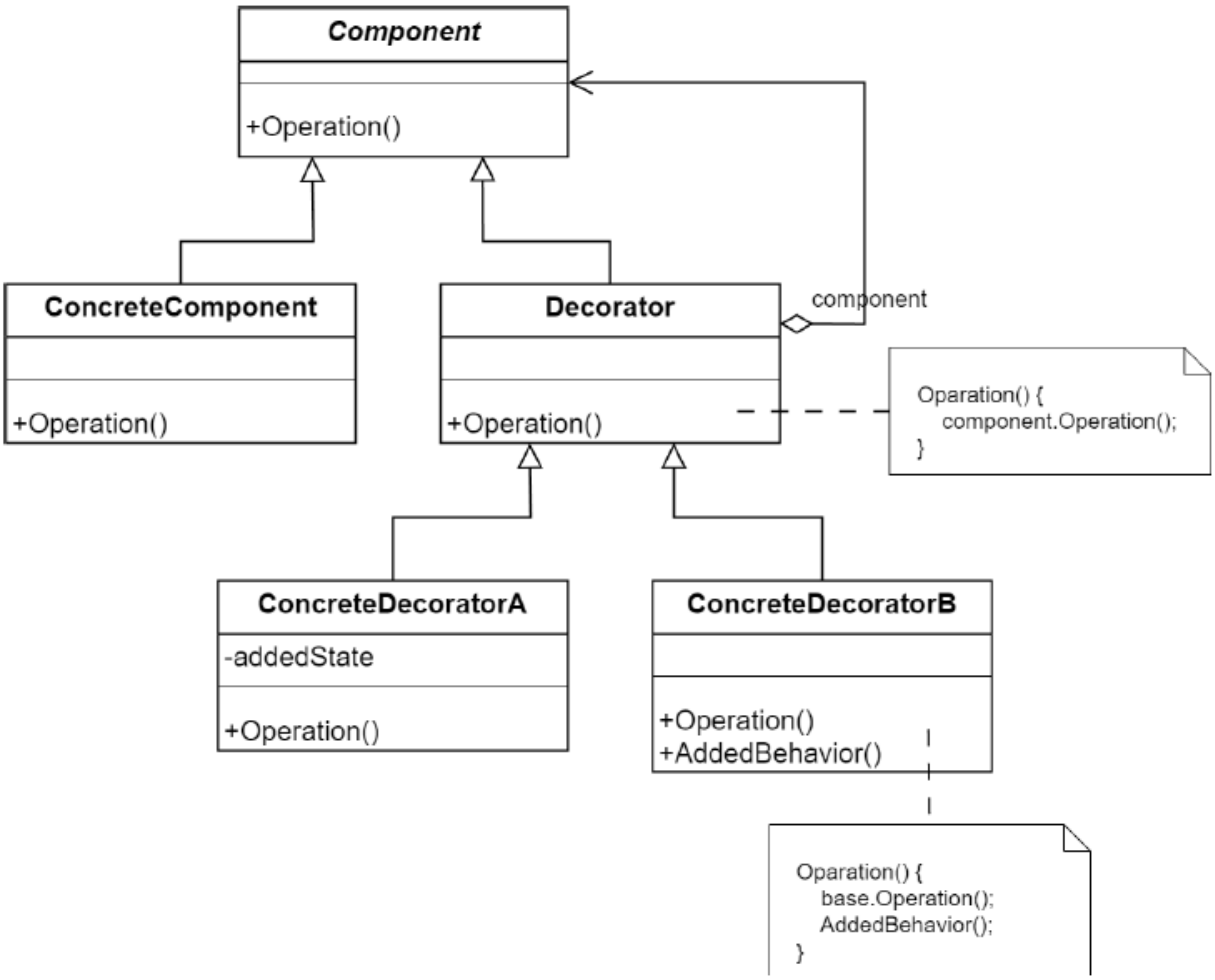


10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія?
Originator (Творець): Об'єкт, стан якого потрібно зберегти (напр., текстовий редактор). Має методи saveStateToMemento() та restoreStateFromMemento().
Memento (Знімок): Простий об'єкт, який зберігає дані стану Originator. Він має бути "непрозорим" для всіх, окрім Originator.
Caretaker (Опікун): Клас, який "дбає" про знімки. Він зберігає список знімків, але не має доступу до їхнього внутрішнього вмісту.

Взаємодія: Caretaker просить Originator "зберегтися". Originator створює Memento зі своїм поточним станом і віддає Caretaker. Пізніше Caretaker повертає Memento назад Originator, щоб той відновив свій стан.

11. Яке призначення шаблону «Декоратор»?
Призначення шаблону «Декоратор» (Decorator) — динамічно додавати об'єктам нову функціональність під час виконання програми, "обгортаючи" їх в об'єкти-декоратори. Це гнучка альтернатива спадкуванню для розширення можливостей класу.

12. Нарисуйте структуру шаблону «Декоратор».



13. Які класи входять в шаблон «Декоратор», та яка між ними взаємодія?
Component (Компонент): Загальний інтерфейс для "справжнього" об'єкта і для декораторів
ConcreteComponent (Конкретний компонент): "Справжній" об'єкт, якому ми додаємо функціональність
Decorator (Декоратор): Абстрактний клас, що реалізує Component і містить посилання на інший об'єкт Component.
ConcreteDecorator (Конкретний декоратор): Клас, що успадковує Decorator
Взаємодія: Клієнт "обгортає" ConcreteComponent в один або декілька ConcreteDecorator. Коли клієнт викликає метод, виклик проходить через ланцюжок декораторів. Кожен декоратор додає свою логіку (напр., логування) до або після виклику методу "обгорнутого" об'єкта.

14. Які є обмеження використання шаблону «декоратор»?
Велика кількість класів: Патерн створює багато дрібних класів, які відрізняються лише реалізацією (один декоратор на кожну додаткову функцію), що може "засмітити" проєкт.

Складність конфігурації: Створення об'єкта стає складним (наприклад, new LoggingDecorator(new EncryptionDecorator(new FileReader()))).

Непрозорість: Ззовні важко сказати, які саме декоратори "навішані" на об'єкт, і яка буде його точна поведінка.

Порядок має значення: Якщо декоратори не спроєктовані обережно, результат їхньої роботи може залежати від порядку, в якому вони були застосовані (напр., шифрувати, а потім стискати — не те саме, що стискати, а потім шифрувати).