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 проєктування»

Виконав
студент групи ІА–34:
Бородай А.С.

Київ 2025

Зміст

Теоретичні відомості

Шаблон «Abstract Factory»

Призначення патерну «Abstract Factory»: використовується для створення сімейств об'єктів без вказівки їх конкретних класів. Для цього виноситься загальний інтерфейс фабрики (AbstractFactory) і створюються його реалізації для різних сімейств продуктів.

Цей шаблон передусім структурує знання про схожі об'єкти (що називаються сімействами) і створює можливість взаємозаміни різних сімейств. 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.

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.

Шаблон «Factory Method»

Призначення «Factory Method»: 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».

Рішення: 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.

Шаблон «Memento»

Призначення «Memento»: 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.

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.

Шаблон «Decorator»

Призначення «Decorator»: коли підписання відповідного документу проходить від його складання у одного із співробітників компанії через менеджера і начальника до головного начальника, який ставить свій підпис.

Проблема: 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.

Рішення: Основна ідея в тому, що нам не потрібно мати загальний метод формування контекстного меню. Ми можемо зробити загальний інтерфейс з методом UpdateContextMenu() і реалізувати цей метод в усіх візуальних компонентах. Додатково для візуальних компонентів добавляємо поле для переходу на "батьківський" елемент, якій в собі містить поточний візуальний компонент, а в реалізації UpdateContextMenu в кінці додаємо виклик цього метода у "батьківського" елемента. Якщо елемент не потребує додавання пунктів в контекстне меню, він просто викликає цей метод у наступного елемента в ланцюжку.

Переваги та недоліки:

+ Зменшує залежність між клієнтом та обробниками: клієнт не знає хто обробить запит, а обробники не знають структуру ланцюжка.

+ Реалізовує додаткову гнучкість в обробці запиту: легко добавити або вилучити з ланцюжка нові обробники.

- Запит може залишитися ніким не опрацьованим: запит не має вказаного обробника, тому може бути не опрацьованим.

Шаблон «Observer»

Призначення «Observer»: 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.

Припустимо, є деяка банківська система і декілька користувачів переглядають баланс на рахунку пана И. У цей момент пан И. кладе на свій рахунок деяку суму, яка міняє загальний баланс. Кожен з користувачів, що переглядали баланс, отримує про це звістку (для користувачів ця звістка може бути прозорою – просто зміна цифр, або попередження про те, що баланс змінився). Раніше неможливі дії для користувачів (переведення до іншої категорії клієнтів і тому подібне) стають доступними.

Переваги та недоліки:

+ Можливість паралельної та асинхронної обробки повідомлень про оновлення.

+ Спостерігачів можна добавляти та видаляти в будь-який момент часу.

+ Спостерігач і суб’єкт можуть працювати в різних потоках.

+ Реалізує принцип слабкого зв’язку між об’єктами.

- Послідовність розсилки повідомлень підписникам не підтримується.

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.

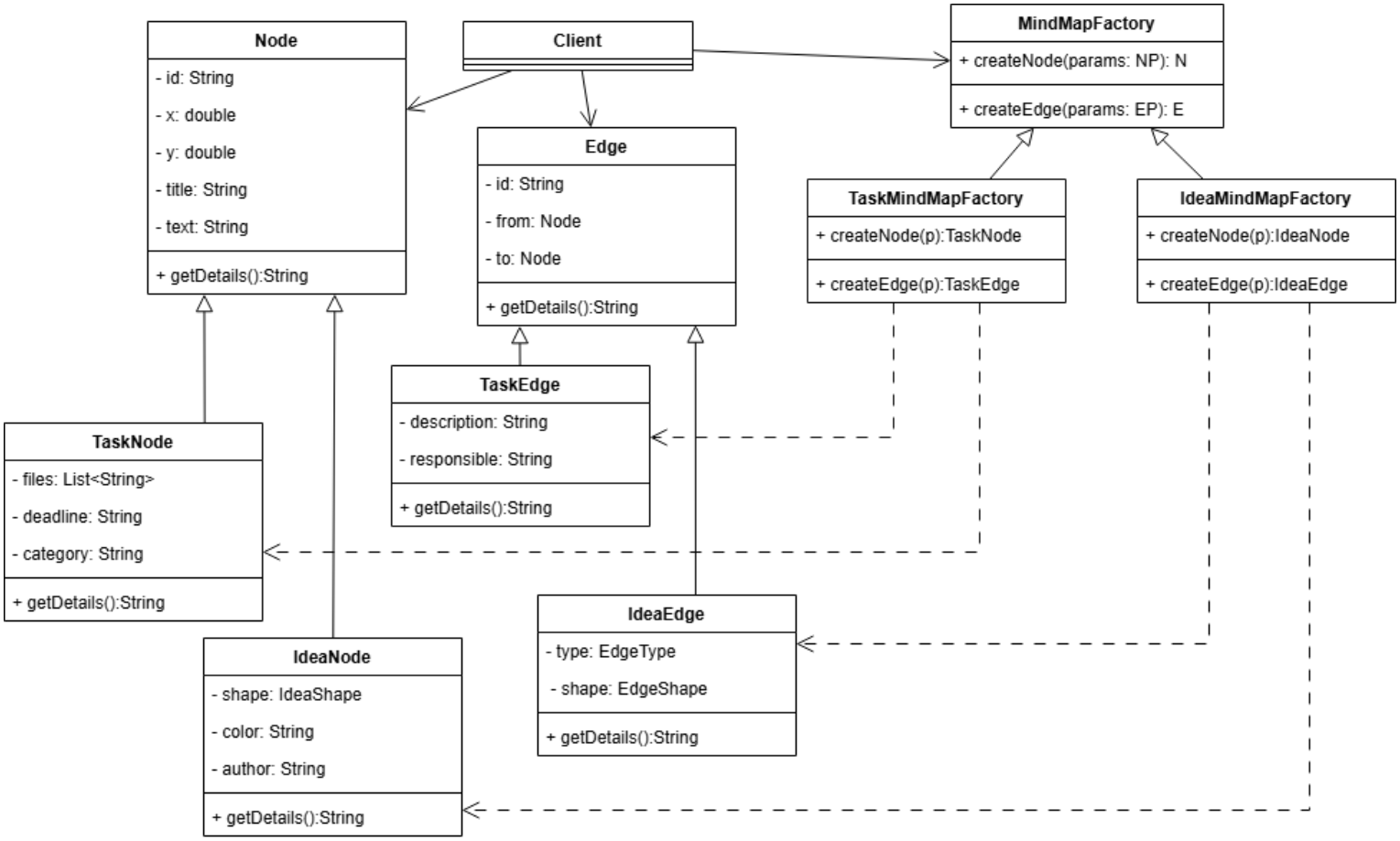
Хід роботи

20. **Mind-mapping software**

(strategy, prototype, abstract factory, bridge, composite, SOA)

Візуальний додаток для складання "карт пам'яті" з можливістю роботи з декількома картами (у вкладках), автоматичного промальовування ліній, додавання вкладених файлів, картинок, відеофайлів (попередній перегляд); можливість додавання значків категорій / терміновості, обведення областей карти (поділ пунктирною лінією).

ClassDiagram (Abstract factory)



MindMapFactory

IdeaMindMapFactory

TaskMindMapFactory

Edge

Node

IdeaEdge

IdeaNode

IdeaEdgeDTO

IdeaNodeDTO

TaskEdge

TaskNode

TaskEdgeDTO

TaskNodeDTO

Висновок

Під час виконання лабораторної роботи я ознайомився із шаблонами «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator». Мною були розроблені основні класи, які реалізують шаблон Abstract Factory та його поведінку.

Цей шаблон спростив створення об’єктів, які належать різним тирам карт. Він відокремив складне створення об’єктів із сторони клієнта та зменшив кількість можливих помилок, наприклад із неправильною взаємодією об’єктів різних сімейств.

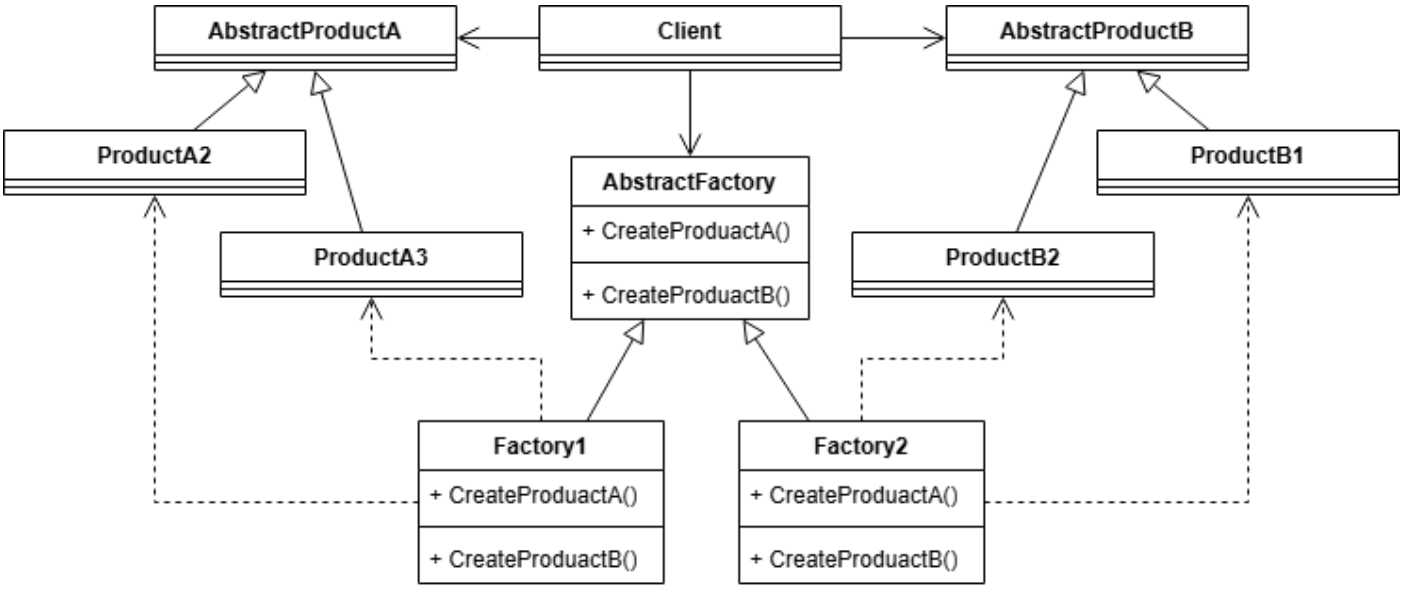
Нові шаблони будуть мною використані для подальшої реалізації системи та подальшого її розуміння.

Відповіді на питання для самоперевірки

1. Яке призначення шаблону «Абстрактна фабрика»?

Спростити та відокремити створення, на стороні клієнта, сімейства об’єктів, які будуть добре узгоджені між собою.

2. Нарисуйте структуру шаблону «Абстрактна фабрика».



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

AbstractFactory – оголошує інтерфейси для створення об’єктів різних типів.

ConcreteFactory - реалізує методи створення конкретних продуктів.

AbstractProduct – спільний інтерфейс для сімейства продуктів.

ConcreteProduct – конкретна реалізація продукту.

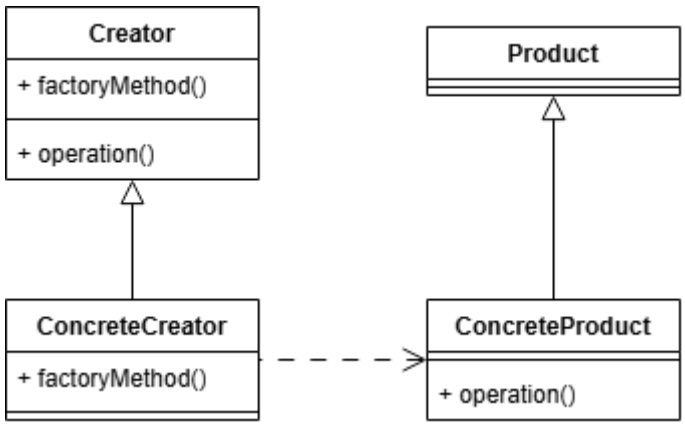
Client – використовує лише інтерфейс фабрики, не знаючи конкретних класів продуктів.

Клієнт створює екземпляр фабрики та використовує для створення продуктів.

4. Яке призначення шаблону «Фабричний метод»?

Делегувати створення об’єктів підкласам, щоб дозволити змінювати тип створюваних об’єктів без зміни клієнтського коду

5. Нарисуйте структуру шаблону «Фабричний метод».



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

Product – спільний інтерфейс створюваних об’єктів.

ConcreteProduct – конкретна реалізація продукту.

Creator – оголошує фабричний метод, який повертає об’єкт типу Product.

ConcreteCreator – перевизначає фабричний метод для створення певного типу продукту.

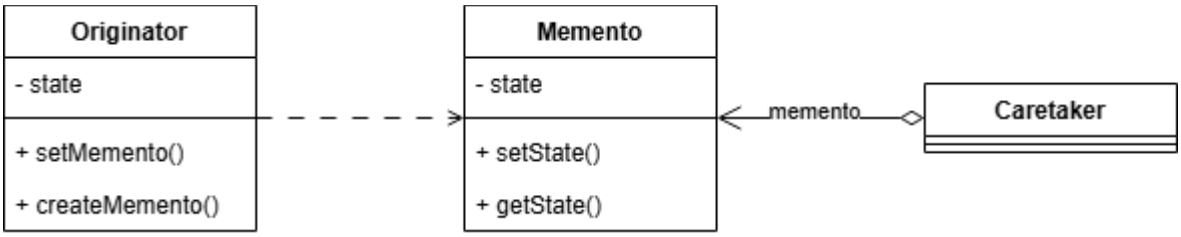
7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?

Абстрактна фабрика створюється для сімейства вже взаємопов’язаних об’єктів, а не один тип об’єкта. Тому, абстрактна фабрика використовує кілька фабричних методів, а не один, і тому є більш складнішим.

8. Яке призначення шаблону «Знімок»?

Призначення шаблону полягає в збережені та відновлені попереднього стану об’єкта без порушення його інкапсуляції.

9. Нарисуйте структуру шаблону «Знімок».



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

Originator – об’єкт, стан якого потрібно зберегти.

Memento – зберігає знімок стану об’єкта.

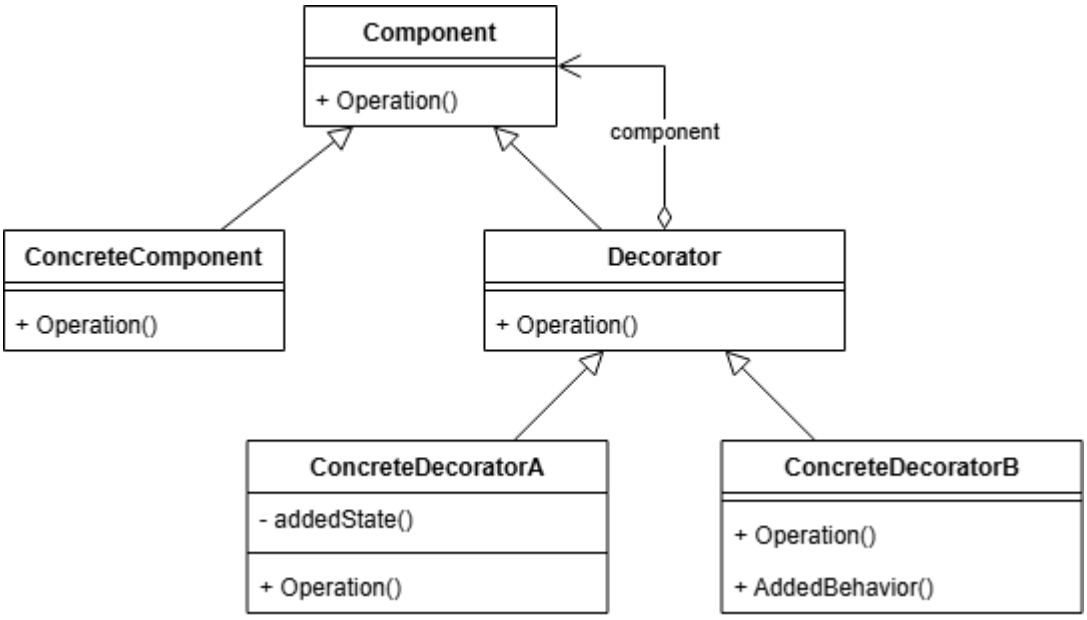
Caretaker – управляє збереженням і відновленням знімків (історія станів).

Взаємодія: Originator створює Memento. Caretaker зберігає його. При потребі Caretaker віддає Memento назад Originator для відновлення стану.

11. Яке призначення шаблону «Декоратор»?

Динамічно додавати нову поведінку об’єктам, не змінюючи їхнього класу.

12. Нарисуйте структуру шаблону «Декоратор».



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

Component – спільний інтерфейс для базового об’єкта і декораторів.

ConcreteComponent – базовий об’єкт, який можна декорувати.

Decorator – містить посилання на компонент і делегує йому виклики.

ConcreteDecorator – додає нову поведінку перед або після делегування виклику.

14. Які є обмеження використання шаблону «декоратор»?

a) Велика кількість дрібних класів ускладнює структуру.

b) Складно відлагоджувати, коли застосовано багато рівнів обгортання.

c) Неможливо змінити поведінку вже створеного ланцюга декораторів без його повного відновлення.

d) Не підходить, коли потрібно глобально змінити поведінку для всіх об’єктів певного класу.