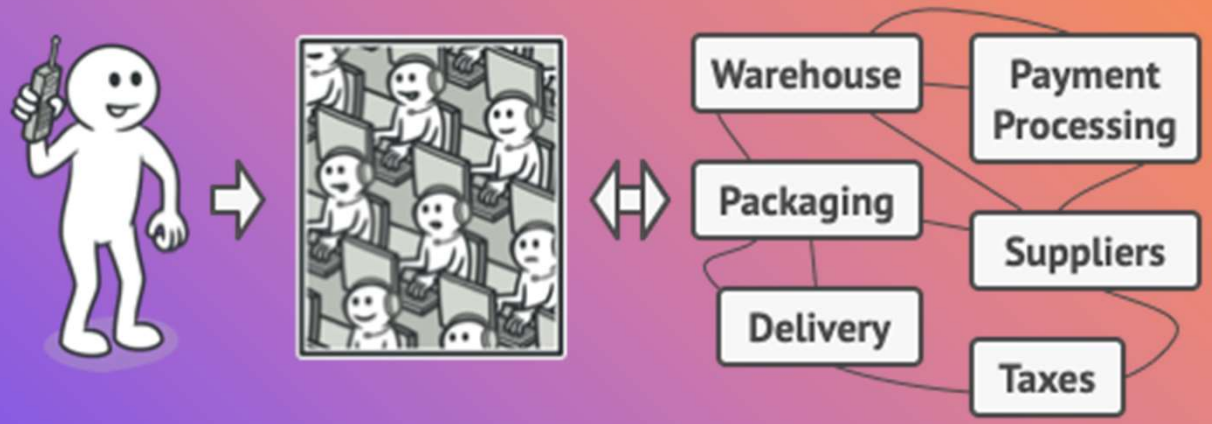
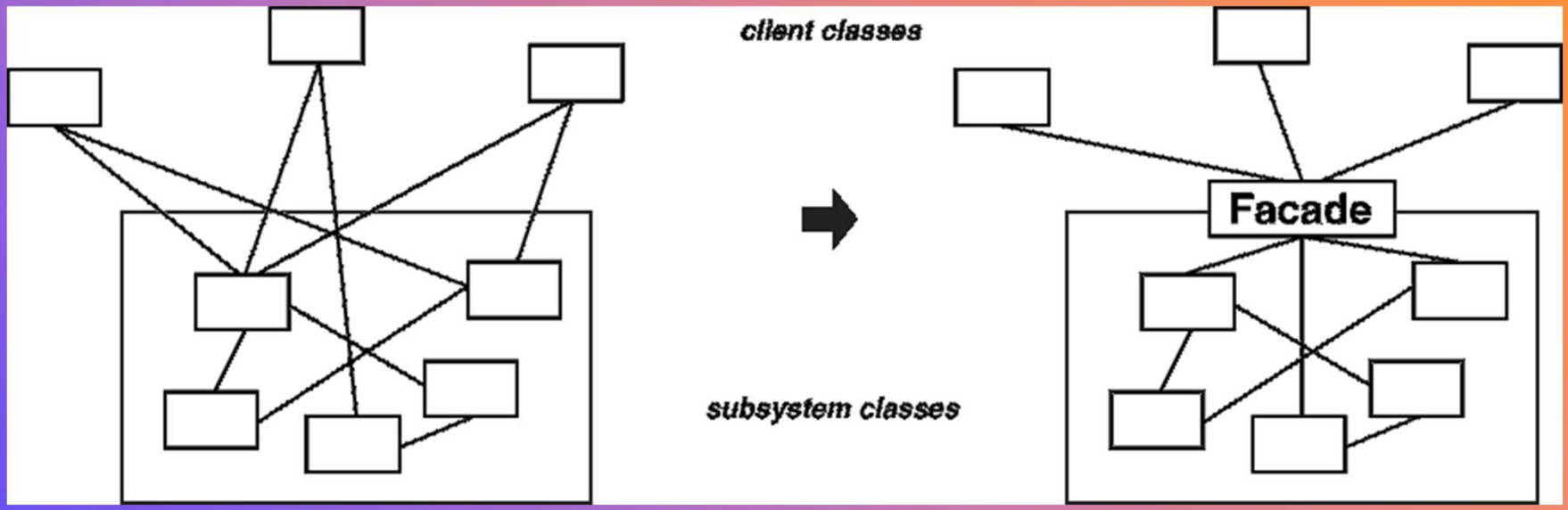


ФАСАД — ЦЕ СТРУКТУРНИЙ ПАТЕРН ПРОЕКТУВАННЯ, ЯКИЙ НАДАЄ ПРОСТИЙ ІНТЕРФЕЙС ДО СКЛАДНОЇ СИСТЕМИ КЛАСІВ, БІБЛІОТЕКИ АБО ФРЕЙМВОРКУ.

Приклад простого інтерфейсу із життя: колл-центр



ПОСТАНОВКА ЗАДАЧІ

Вашому коду доводиться працювати з великою кількістю об’єктів певної складної бібліотеки чи фреймворка. Ви повинні самостійно ініціалізувати ці об’єкти, стежити за правильним порядком залежностей тощо.

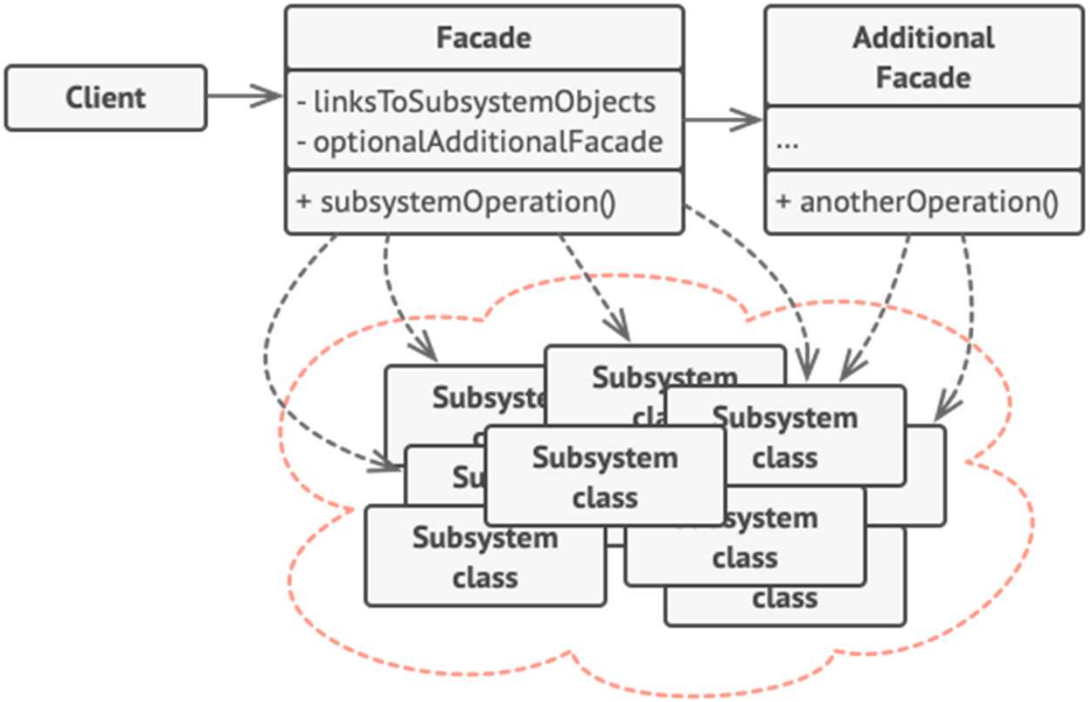
В результаті бізнес-логіка ваших класів тісно переплітається з деталями реалізації сторонніх класів. Такий код досить складно розуміти та підтримувати.

РІШЕННЯ

Фасад корисний у тому випадку, якщо ви використовуєте якусь складну бібліотеку з безліччю рухомих частин, з яких вам потрібна тільки частина.

Наприклад, програма, що заливає в соціальні мережі відео з кошенятками, може використовувати професійну бібліотеку для стискання відео, але все, що потрібно клієнтському коду цієї програми, — це простий метод encode(filename, format). Створивши клас з таким методом, ви реалізуєте свій перший фасад.

Структура



- **Фасад** надає швидкий доступ до певної функціональності підсистеми. Він «знає», яким класам потрібно переадресувати запит, і які дані для цього потрібні.
- **Додатковий фасад** можна ввести, щоб не захаращувати єдиний фасад різнорідною функціональністю. Він може використовуватися як клієнтом, так й іншими фасадами.
- **Складна підсистема** має безліч різноманітних класів. Для того, щоб примусити усіх їх щось робити, потрібно знати подробиці влаштування підсистеми, порядок ініціалізації об’єктів та інші деталі. Класи підсистеми не знають про існування фасаду і працюють один з одним безпосередньо.
- **Клієнт** використовує фасад замість безпосередньої роботи з об’єктами складної підсистеми.

Застосування

•Якщо вам потрібно надати простий або урізаний інтерфейс до складної підсистеми.

•Часто підсистеми ускладнюються в міру розвитку програми. Застосування більшості патернів призводить до появи менших класів, але у великій кількості. Таку підсистему простіше використовувати повторно, налаштовуючи її кожен раз під конкретні потреби, але, разом з тим, використовувати таку підсистему без налаштовування важче. Фасад пропонує певний вид системи за замовчуванням, який влаштовує більшість клієнтів.

•Якщо ви хочете розкласти підсистему на окремі рівні.

•Використовуйте фасади для визначення точок входу на кожен рівень підсистеми. Якщо підсистеми залежать одна від одної, тоді залежність можна спростити, дозволивши підсистемам обмінюватися інформацією тільки через фасади.

•Наприклад, візьмемо ту ж саму складну систему конвертації відео. Ви хочете розбити її на окремі шари для роботи з аудіо й відео. Можна спробувати створити фасад для кожної з цих частин і примусити класи аудіо та відео обробки спілкуватися один з одним через ці фасади, а не безпосередньо.

Кроки реалізації

1. Визначте, чи можна створити більш простий інтерфейс, ніж той, який надає складна підсистема. Ви на правильному шляху, якщо цей інтерфейс позбавить клієнта від необхідності знати подробиці підсистеми.
2. Створіть клас фасаду, що реалізує цей інтерфейс. Він повинен переадресовувати виклики клієнта потрібним об’єктам підсистеми. Фасад повинен буде подбати про те, щоб правильно ініціалізувати об’єкти підсистеми.
3. Ви отримаєте максимум користі, якщо клієнт працюватиме тільки з фасадом. В такому випадку зміни в підсистемі стосуватимуться тільки коду фасаду, а клієнтський код залишиться робочим.
4. Якщо відповідальність фасаду стає розмитою, подумайте про введення додаткових фасадів.

Переваги та недоліки

+

- Ізолює клієнтів від компонентів складної підсистеми.
- Підтримує принцип loose coupling
- Програмне забезпечення стає більш гнучким і легко розширюваним

-

- Фасад ризикує стати божественим об’єктом, прив’язаним до всіх класів програми (Високий ступінь залежності)
- Складна реалізація (особливо з існуючим кодом)
- Підхід поєднується з додатковим рівнем опосередкованості

Робота з іншими патернами

Singleton

- Фасад можна зробити Одинаком, оскільки зазвичай потрібен тільки один об’єкт-фасад.

Abstract Factory

- Абстрактна фабрика може бути використана замість Фасаду для того, щоб приховати платформо-залежні класи.

Адаптер

- Фасад задає новий інтерфейс, тоді як Адаптер повторно використовує старий.
- Адаптер обгортає тільки один клас, а Фасад обгортає цілу підсистему.