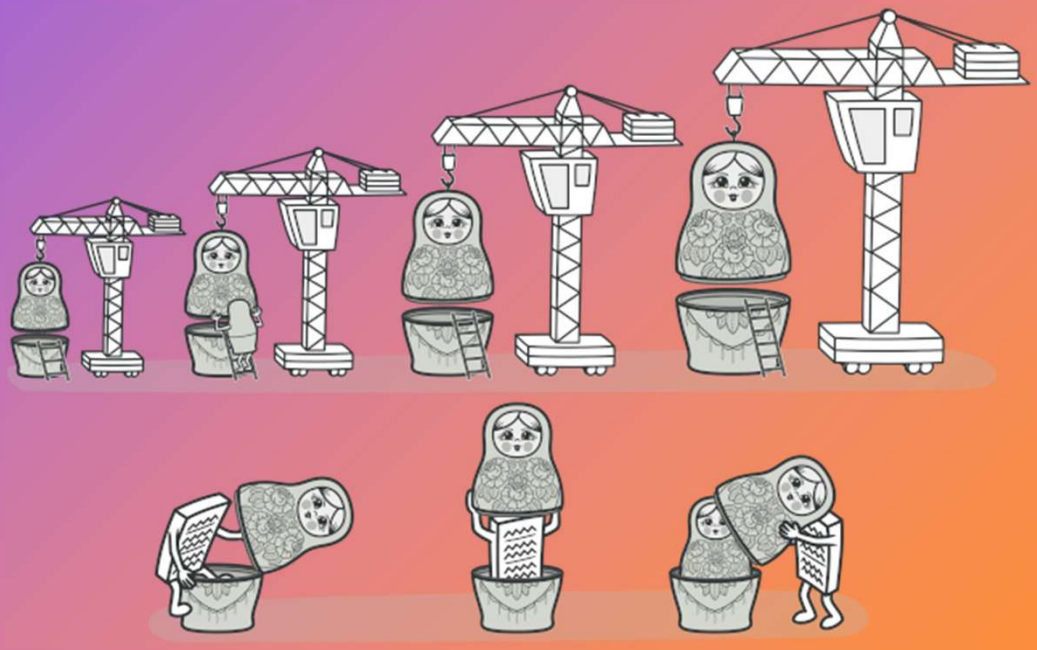
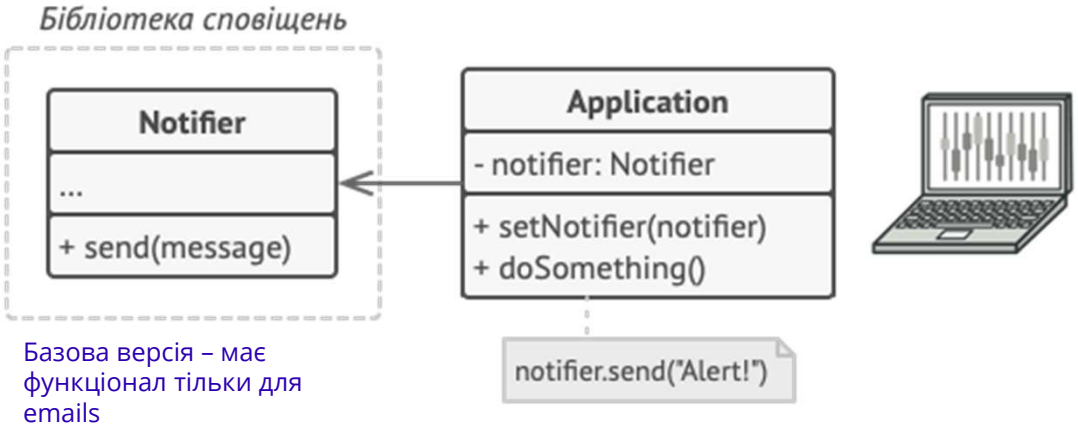
DECORATOR

Також відомий як: Wrapper, Обгортка

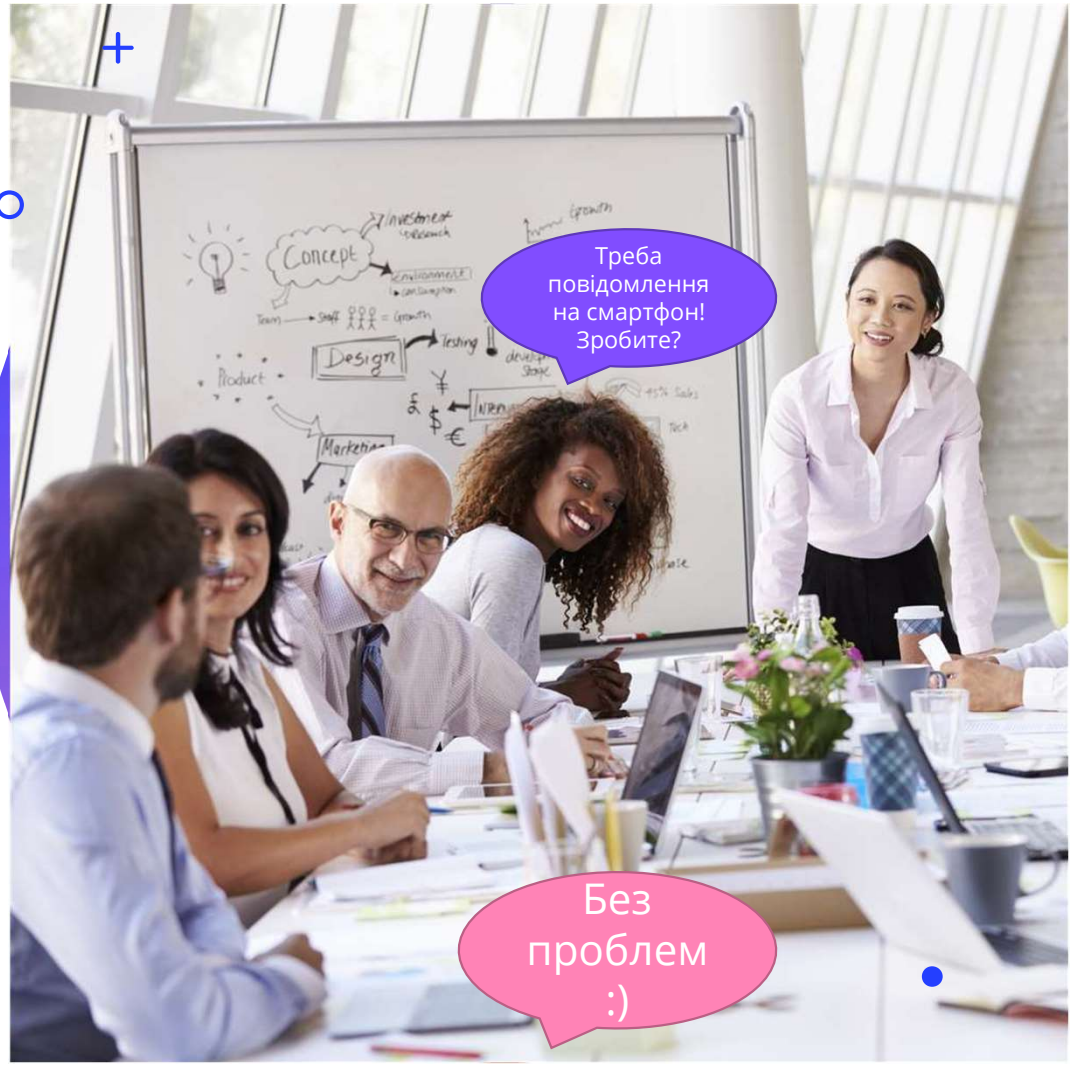
ДЕКОРАТОР — це структурний патерн проектування, що дає змогу динамічно додавати об’єктам нову функціональність, загортаючи їх у корисні «обгортки».



ПОСТАНОВКА ЗАДАЧІ



- Маємо бібліотеку сповіщень, яку можна підключати до різноманітних програм, щоб отримувати сповіщення про важливі події.
- Клас ***Notifier*** - основа бібліотеки.
- Метод ***send*** приймає на вхід повідомлення і надсилає його всім адміністраторам *електронною поштою.*
- Програма ззовні повинна створити й налаштувати об'єкт ***Notifier***, вказавши, кому надсилати сповіщення, та використовувати його щоразу, коли щось відбувається.

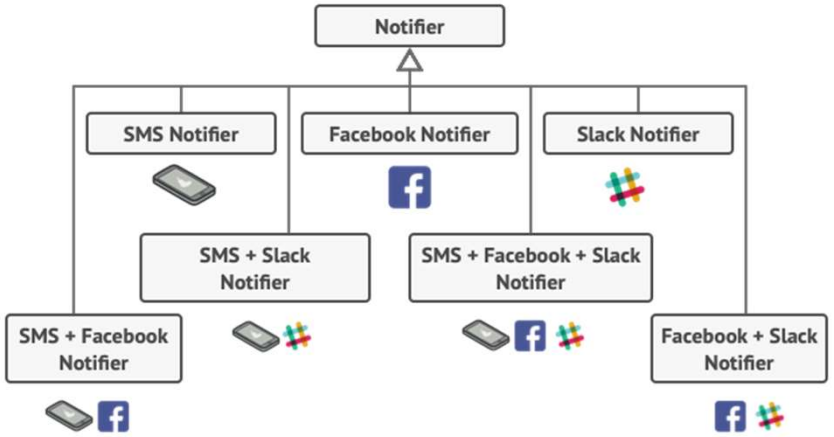


ОТРИМАЛИ ЗАПИТ ВІД КОРИСТУВАЧІВ...

- Деякі звернулись з запитом отримувати сповіщення про критичні проблеми через **SMS**.
- Користувачі корпоративної версії ззовнішньої програми просять зробити функціонал для **Slack**.
- Клієнти з програми неформального напрямку попросили додати можливість сповіщень через **Facebook**.

ПЕРША ДУМКА? СПАДКУВАННЯ !

Ми додаємо кожен з типів сповіщень до програми, успадкувавши їх від базового класу ***Notifier***. Тепер користувачі могли вибрати один з типів сповіщень, який і використовувався надалі.



А МОЖЕ, СПАДКУВАННЯ – НЕ НАЙКРАЩЕ РІШЕННЯ?

•Користувач додатку домашньої сигналізації тепер попросив запустити **всі види повідомлень одночасно**.

•Після того, як ми додали перший десяток класів, стало зрозуміло, що такий підхід неймовірно роздуває код програми...

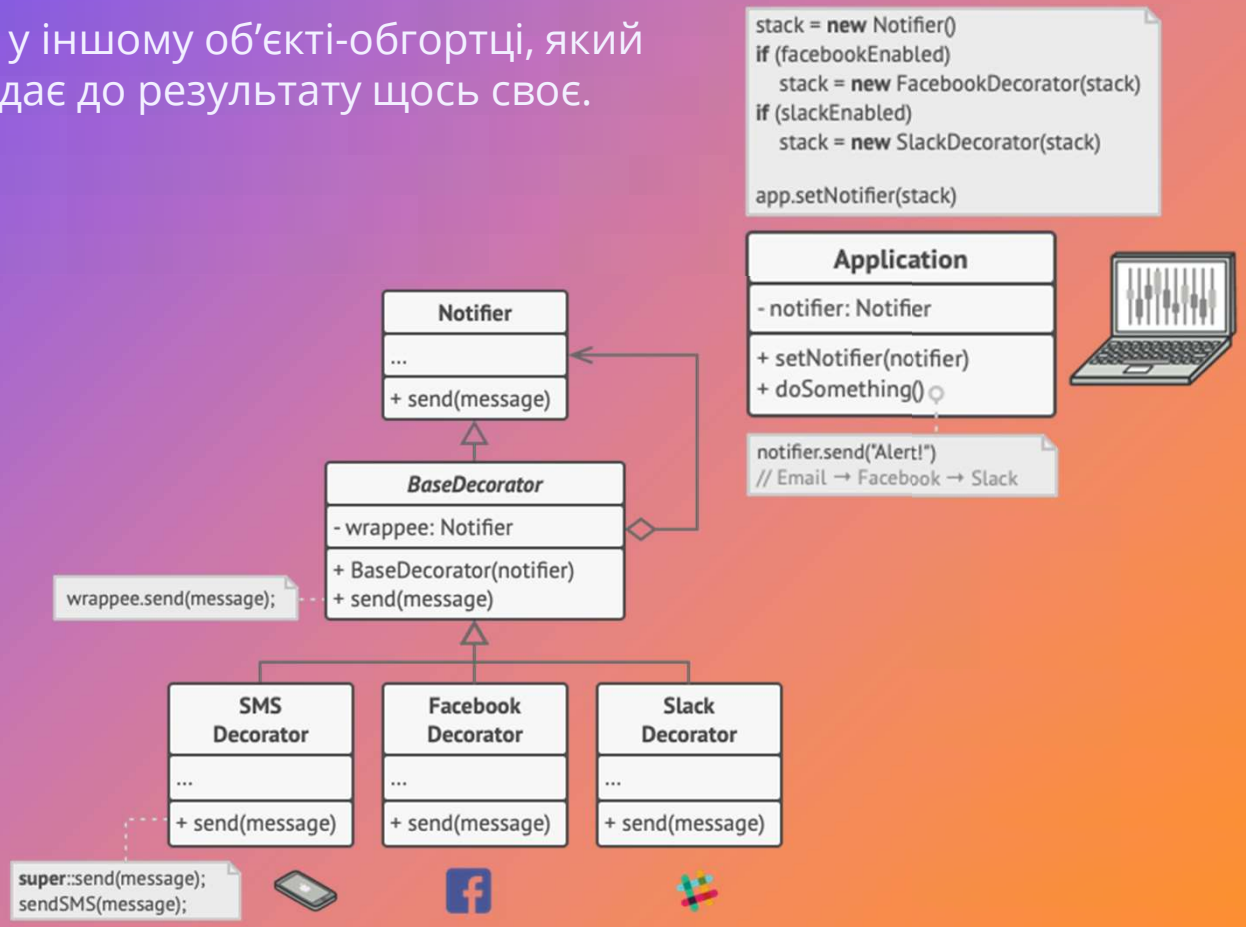
•Успадкування статичне (не можемо змінювати поведінку об’єкта, що вже існує) та не дозволяє наслідувати поведінку декількох класів одночасно. Тому доведеться створювати безліч підкласів-комбінацій, щоб досягти поєднання поведінки.

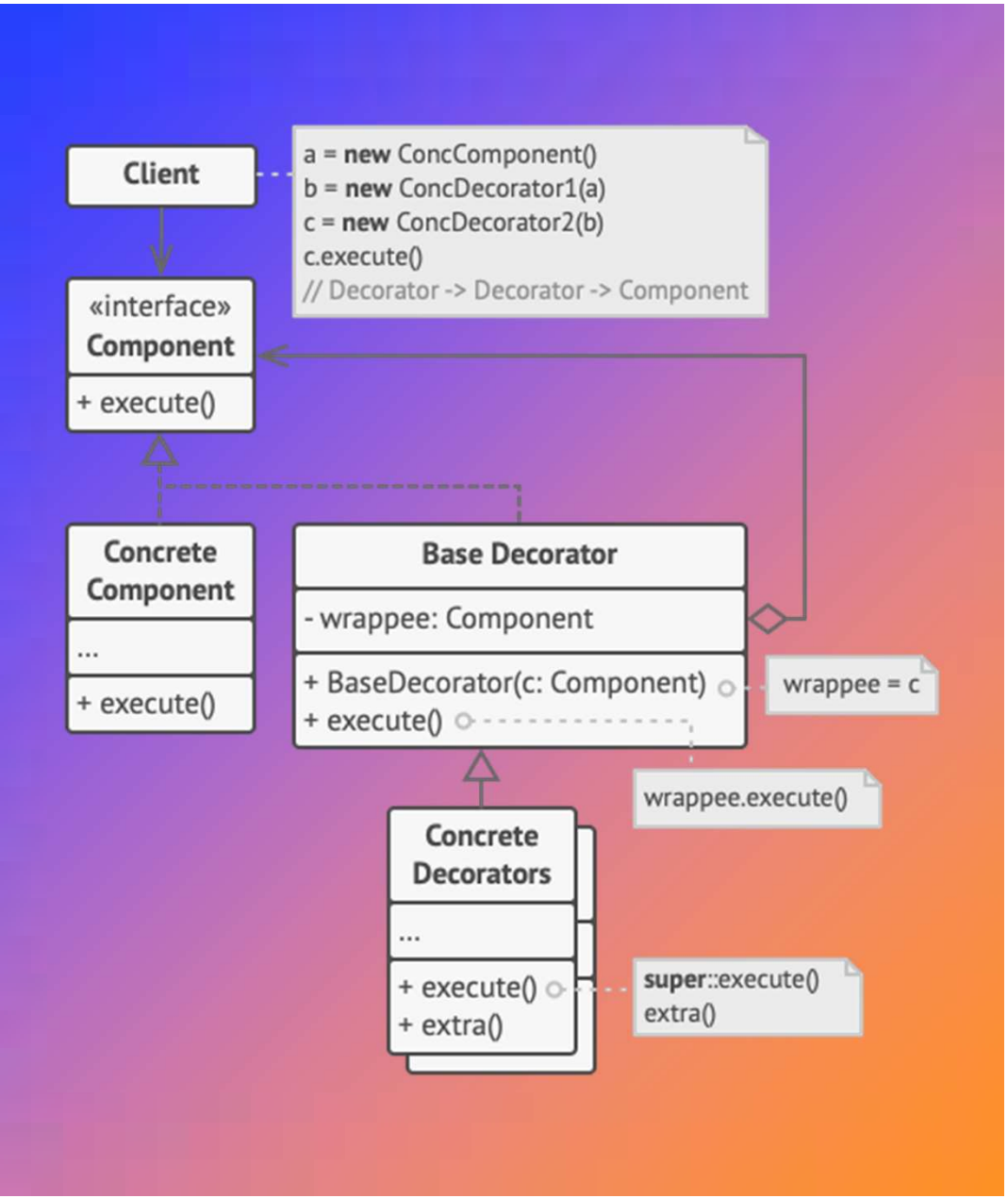
•**Агрегація** : один об’єкт утримує інший і делегує йому роботу, замість того, щоб самому успадкувати його поведінку. Саме на цьому принципі побудовано патерн Декоратор.

ПОЗБУВАЄМОСЬ МИЛИЦЬ - ЗАСТОСОВУЄМО DECORATOR

Суть патерна: цільовий об’єкт розміщується у іншому об’єкті-обгортці, який запускає базову поведінку об’єкта, а потім додає до результату щось своє.

- В нашому прикладі залишимо в базовому класі просте надсилання сповіщень електронною поштою, а розширені способи зробимо ***декораторами***.
- Стороння програма, під час початкового налаштовування буде загортати об’єкт сповіщення в ті обгортки, які відповідають бажаному способу сповіщення.
- Остання обгортка у списку буде саме тим об’єктом, з яким клієнт працюватиме увесь інший час.





СТРУКТУРА

1. **Компонент** задає загальний інтерфейс обгорток та об'єктів, що загортаються.
2. **Конкретний компонент** визначає клас об'єктів, що загортаються. Він містить якусь базову поведінку, яку потім змінюють декоратори.
3. **Базовий декоратор** зберігає посилання на вкладений об'єкт-компонент. Це може бути як конкретний компонент, так і один з конкретних декораторів. Базовий декоратор *делегує* всі свої операції вкладеному об'єкту. Додаткова поведінка житиме в конкретних декораторах.
4. **Конкретні декоратори** — це різні варіації декораторів, що містять додаткову поведінку. Вона виконується до або після виклику аналогічної поведінки загорнутого об'єкта.
5. **Клієнт** може обертати прості компоненти й декоратори в інші декоратори, працюючи з усіма об'єктами через загальний інтерфейс компонентів.

Застосування

- **Якщо існує потреба додавати об’єктам функціонал, непомітно для клієнтського коду**

Обгортки і самі об’єкти мають однаковий інтерфейс, тому клієнтам не важливо, з чим працювати

- **Якщо не можна розширити обов’язки об’єкта за допомогою спадкування.**

У багатьох мовах програмування є ключове слово final, яке може заблокувати спадкування класу. Розширити такі класи можна тільки за допомогою Декоратора

Кроки реалізації

1. Переконайтеся, що у вашому завданні присутні основний компонент і декілька опціональних доповнень-надбудов над ним.
2. Створіть інтерфейс компонента, який описував би загальні методи як для основного компонента, так і для його доповнень.
3. Створіть клас конкретного компонента й помістіть в нього основну бізнес-логіку.
4. Створіть базовий клас декораторів. Він повинен мати поле для зберігання посилань на вкладений об'єкт-компонент. Усі методи базового декоратора повинні делегувати роботу вкладеному об'єкту.
5. Конкретний компонент, як і базовий декоратор, повинні дотримуватися одного і того самого інтерфейсу компонента.
6. Створіть класи конкретних декораторів, успадковуючи їх від базового декоратора. Конкретний декоратор повинен виконувати свою додаткову функціональність, а потім (або перед цим) викликати цю ж операцію загорнутого об'єкта.
7. Клієнт бере на себе відповідальність за конфігурацію і порядок загортання об'єктів.

Переваги та недоліки

- ✔ Більша гнучкість, ніж у спадкування.
- ✔ Дозволяє додавати обов’язки «на льоту».
- ✔ Можна додавати кілька нових обов’язків одразу.
- ✔ Дозволяє мати кілька дрібних об’єктів, замість одного об’єкта «на всі випадки життя».

- Важко конфігурувати об’єкти, які загорнуто в декілька обгорток одночасно.
- Легко захопитись і понастворювати велику кількість крихітних класів