Java Design Patterns

Decorator

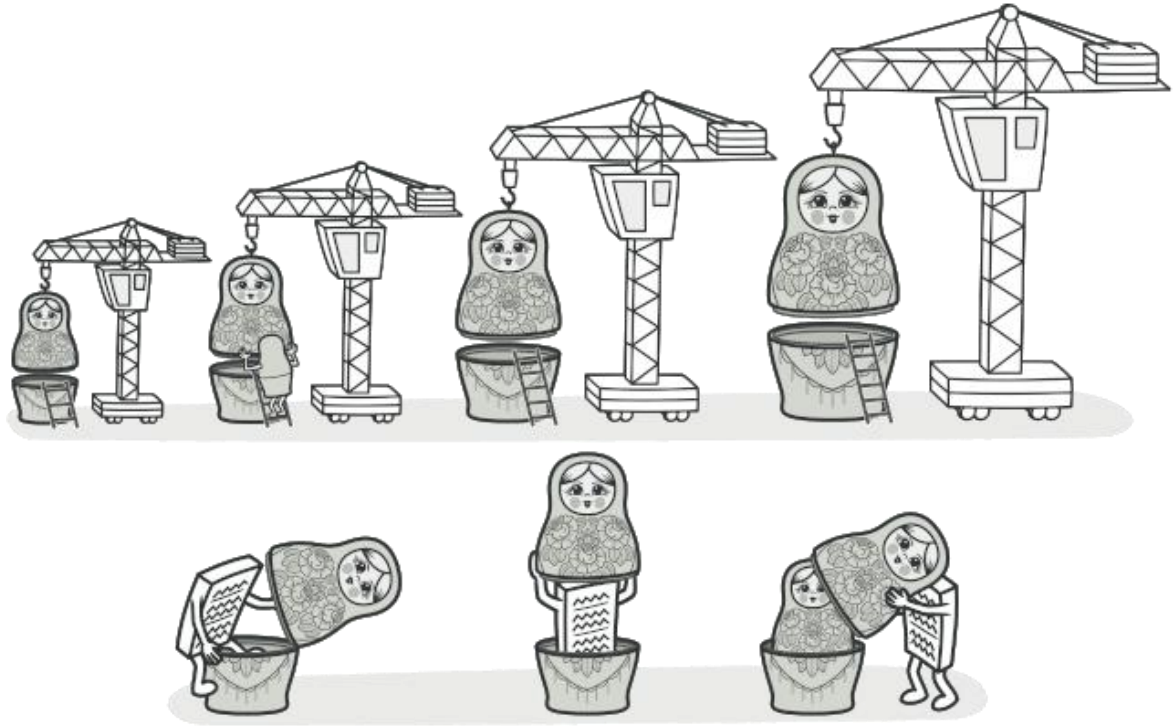
Тема

Decorator

Суть паттерна

Декоратор

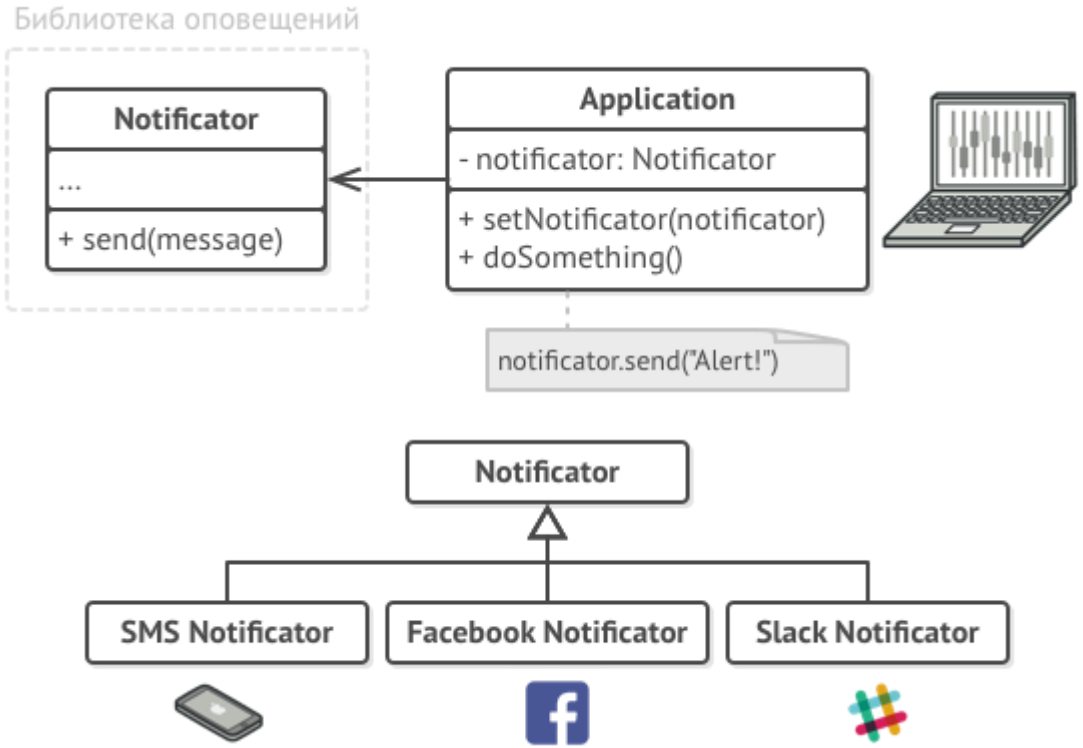
Декоратор — это структурный паттерн проектирования, который позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки».

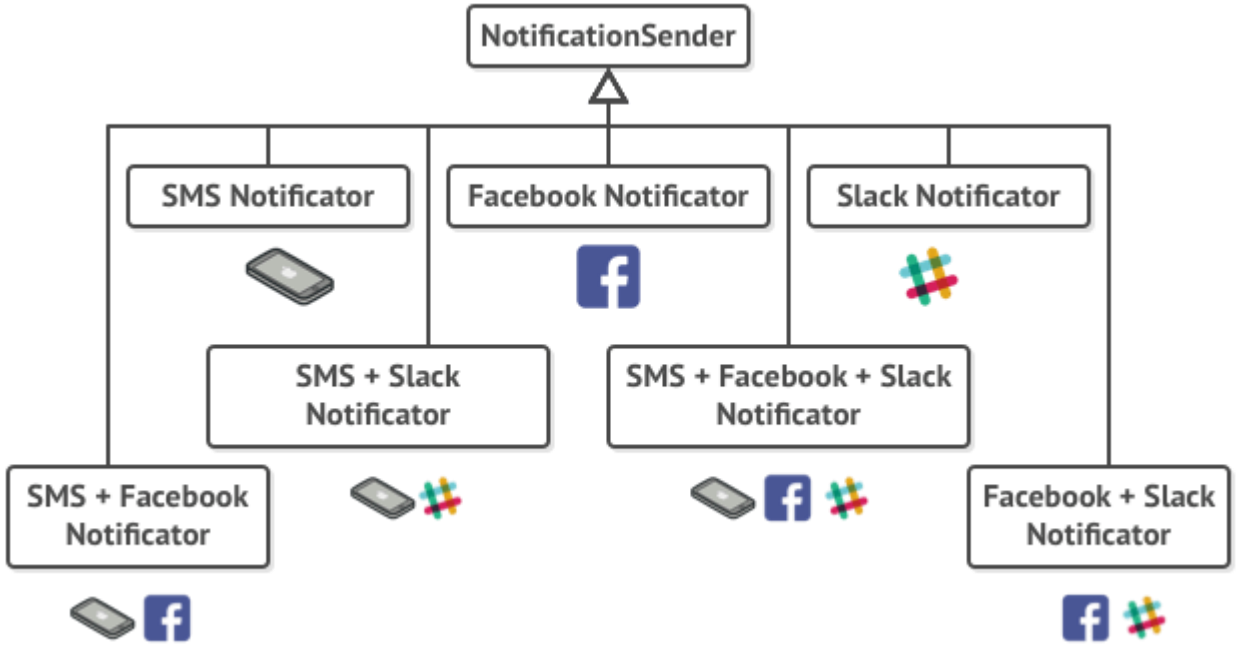


Проблема

Постановка задачи

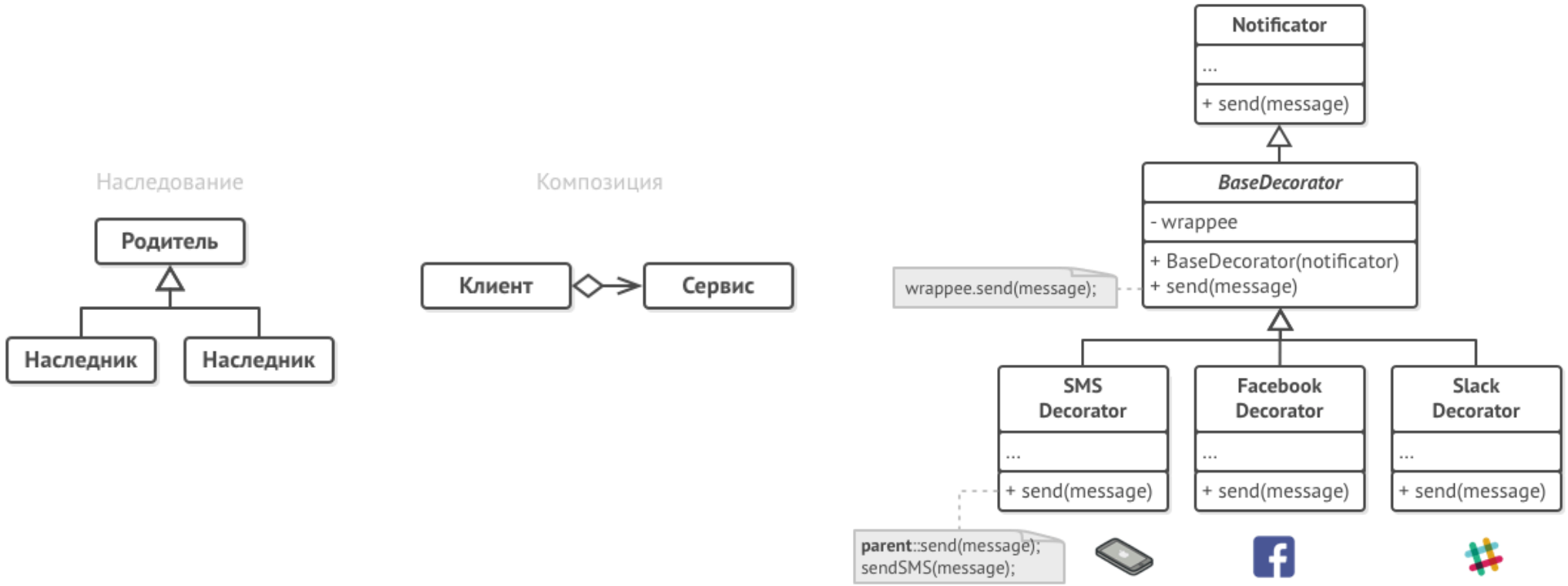
Вы работаете над библиотекой оповещений, которую можно подключать к разнообразным программам, чтобы получать уведомления о важных событиях.





Решение задачи

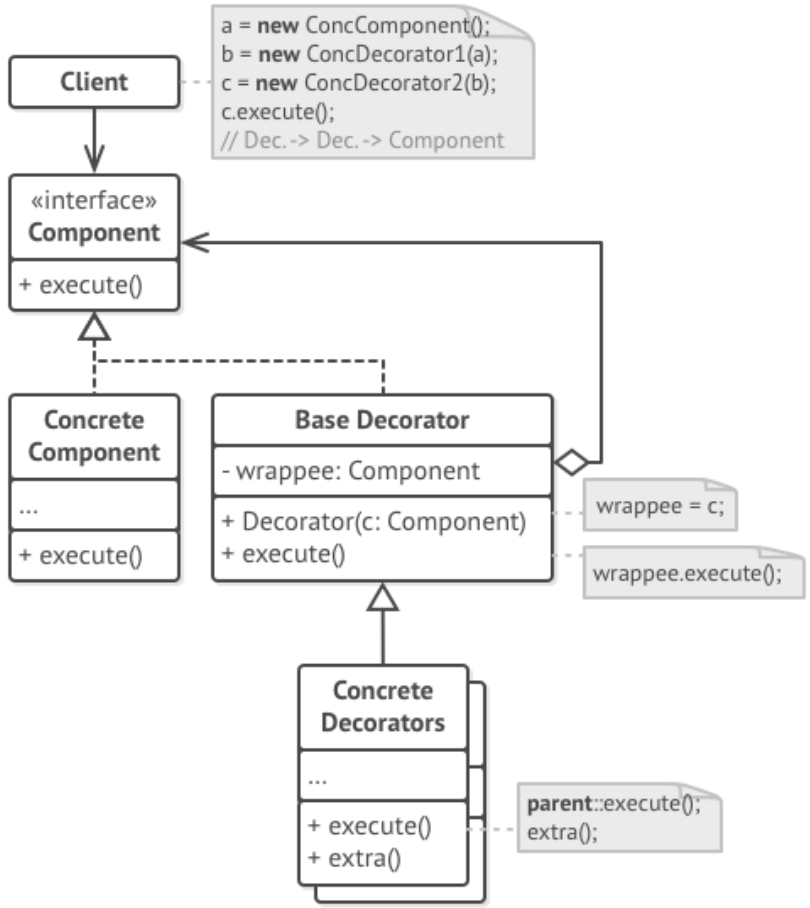
Наследование — это первое, что приходит в голову многим программистам, когда нужно расширить какое-то существующее поведение. Но механизм наследования имеет несколько досадных проблем.



Структура

Структура паттерна

1. Компонент задаёт общий интерфейс обёрток и оборачиваемых объектов.
2. Конкретный Компонент определяет класс оборачиваемых объектов. Он содержит какое-то базовое поведение, которое потом изменяют декораторы.
3. Базовый Декоратор хранит ссылку на вложенный объект-компонент. Им может быть как конкретный компонент, так и один из конкретных декораторов. Базовый декоратор делегирует все свои операции вложенному объекту. Дополнительное поведение будет жить в конкретных декораторах.
4. Конкретные Декораторы — это различные вариации декораторов, которые содержат добавочное поведение. Оно выполняется до или после вызова аналогичного поведения обёрнутого объекта.



Применимость

Применение паттерна

1. Когда вам нужно добавлять обязанности объектам на лету, незаметно для кода, который их использует.
2. Когда нельзя расширить обязанности объекта с помощью наследования.

Шаги реализации

Алгоритм реализации паттерна

1. Убедитесь, что в вашей задаче есть один основной компонент и несколько опциональных дополнений или надстроек над ним.
2. Создайте интерфейс компонента, который описывал бы все общие методы как для основного компонента, так и для его дополнений.
3. Создайте класс конкретного компонента и поместите в него основную бизнес-логику.
4. Создайте базовый класс декораторов. Он должен иметь поле для хранения ссылки на вложенный объект-компонент. Все методы базового декоратора должны делегировать действие вложенному объекту.
5. И конкретный компонент, и базовый декоратор должны следовать одному и тому же интерфейсу компонента.
6. Теперь создайте классы конкретных декораторов, наследуя их от базового декоратора. Конкретный декоратор должен выполнять свою добавочную функциональность, а затем (или перед этим) вызывать эту же операцию обёрнутого объекта.
7. Клиент берёт на себя ответственность за конфигурацию и порядок обёртывания объектов.

Преимущества и недостатки

Плюсы и недостатки

Плюсы:

- Большая гибкость, чем у наследования.
- Позволяет добавлять обязанности на лету.
- Можно добавлять несколько новых обязанностей сразу.
- Позволяет иметь несколько мелких объектов вместо одного объекта на все случаи жизни.

Минусы:

- Трудно конфигурировать многократно обёрнутые объекты.
- Обилие крошечных классов.

Отношение с другими паттернами

- Адаптер меняет интерфейс существующего объекта. Декоратор улучшает другой объект без изменения его интерфейса. Причём *Декоратор* поддерживает рекурсивную вложенность, чего не скажешь об *Адаптере*.
- Адаптер предоставляет классу альтернативный интерфейс. Декоратор предоставляет расширенный интерфейс. Заместитель предоставляет тот же интерфейс.
- Цепочка обязанностей и Декоратор имеют очень похожие структуры. Оба паттерна базируются на принципе рекурсивного выполнения операции через серию связанных объектов. Но есть и несколько важных отличий.
- Компоновщик и Декоратор имеют похожие структуры классов из-за того, что оба построены на рекурсивной вложенности. Она позволяет связать в одну структуру бесконечное количество объектов.
- Архитектура, построенная на Компоновщиках и Декораторах, часто может быть улучшена за счёт внедрения Прототипа. Он позволяет клонировать сложные структуры объектов, а не собирать их заново.
- Стратегия меняет поведение объекта «изнутри», а Декоратор изменяет его «снаружи».

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association