Java Design Patterns

Composite

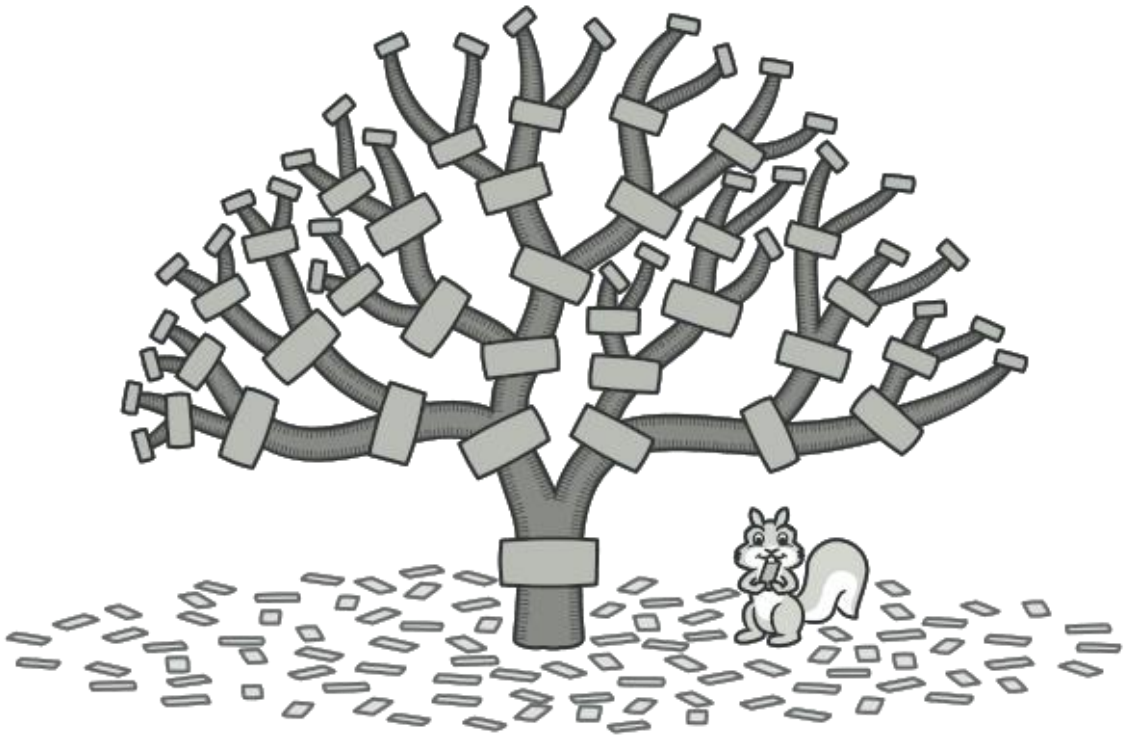
Тема

Composite

Суть паттерна

Компоновщик

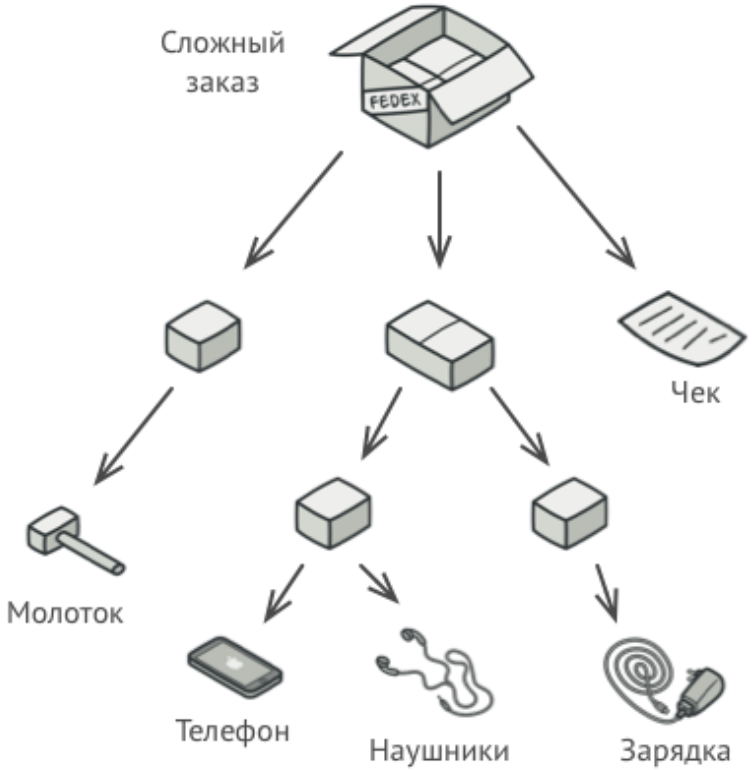
Компоновщик — это структурный паттерн проектирования, который позволяет сгруппировать объекты в древовидную структуру, а затем работать с ними так, если бы это был единичный объект.



Проблема

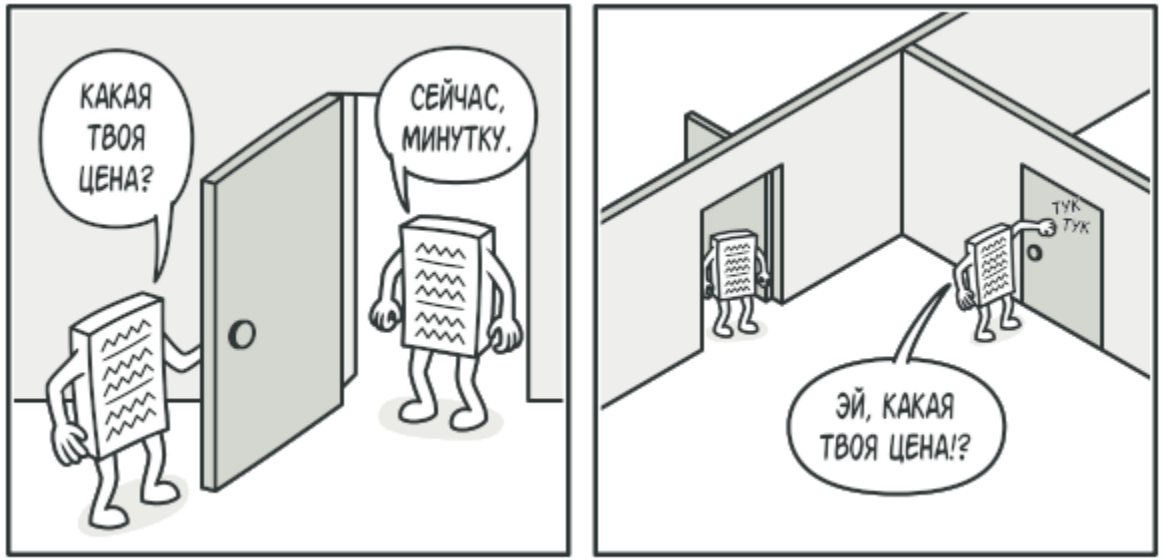
Постановка задачи

Паттерн Компоновщик имеет смысл только тогда, когда основная модель вашей программы может быть структурирована в виде дерева.



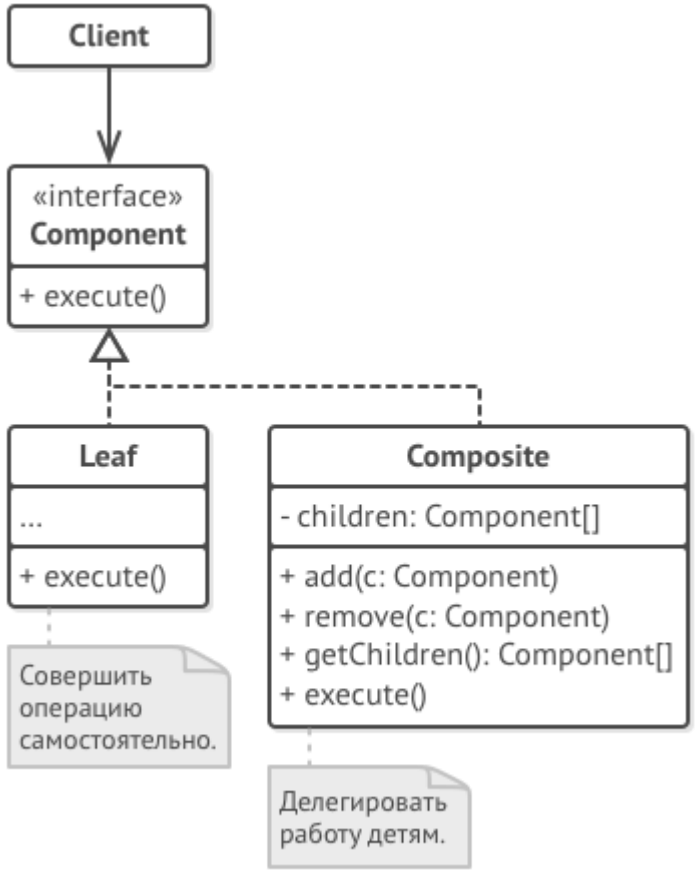
Решение задачи

Компоновщик предлагает рассматривать Продукт и Коробку через единый интерфейс с общим методом получения цены.



Структура паттерна

1. Компонент определяет общий интерфейс для простых и составных компонентов дерева.
2. Лист – это простой элемент дерева, не имеющий ответвлений.
3. Из-за того, что им некому больше передавать выполнение, классы Листьев будут содержать большую часть полезного кода.
4. Контейнер (или «*композит*») — это составной элемент дерева. Он содержит набор дочерних компонентов, но ничего не знает об их типах. Это могут быть как простые компоненты-листья, так и другие компоненты-контейнеры. Но это не является проблемой, так как все дочерние элементы следуют общему интерфейсу.
5. Методы контейнера переадресуют основную работу своим дочерним компонентам, хотя и могут добавлять что-то своё к результату.
6. Клиент работает с деревом через общий интерфейс компонентов.
7. Благодаря этому, клиенту без разницы что перед ним находится — простой или составной компонент дерева



Применимость

Применение паттерна

1. Когда вам нужно представить древовидную структуру объектов.
2. Когда клиенты должны единообразно трактовать простые и составные объекты.

Шаги реализации

Алгоритм реализации паттерна

1. Убедитесь, что вашу бизнес-логику можно представить как древовидную структуру. Попытайтесь разбить её на простые элементы и контейнеры. Помните, что контейнеры могут содержать как простые элементы, так и другие контейнеры.
2. Создайте общий интерфейс компонентов, который объединит операции контейнеров и простых элементов дерева. Интерфейс будет удачным, если вы сможете взаимозаменять простые и составные компоненты без потери смысла.
3. Создайте класс компонентов-листьев, не имеющих дальнейших ответвлений. Имейте в виду, что программа может содержать несколько видов таких классов.
4. Создайте класс компонентов-контейнеров, и добавьте в него массив для хранения ссылок на вложенные компоненты. Этот массив должен быть способен содержать как простые, так и составные компоненты, поэтому убедитесь, что он объявлен с типом интерфейса компонентов.
5. Добавьте операции добавления и удаления дочерних элементов в класс контейнеров.

Плюсы и недостатки

Плюсы:

- Упрощает архитектуру клиента при работе со сложным деревом компонентов.
- Облегчает добавление новых видов компонентов.

Минусы:

- Создаёт слишком общий дизайн классов.

Отношение с другими паттернами

- Строитель позволяет пошагово сооружать дерево Компоновщика.
- Цепочку обязанностей часто используют вместе с Компоновщиком. В этом случае, запрос передаётся от дочерних компонентов к их родителям.
- Вы можете обходить дерево Компоновщика, используя Итератор.
- Вы можете выполнить какое-то действие над всем деревом Компоновщика при помощи Посетителя.
- Компоновщик часто совмещают с Легковесом, чтобы реализовать общие ветки дерева и сэкономить при этом память.
- 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.

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