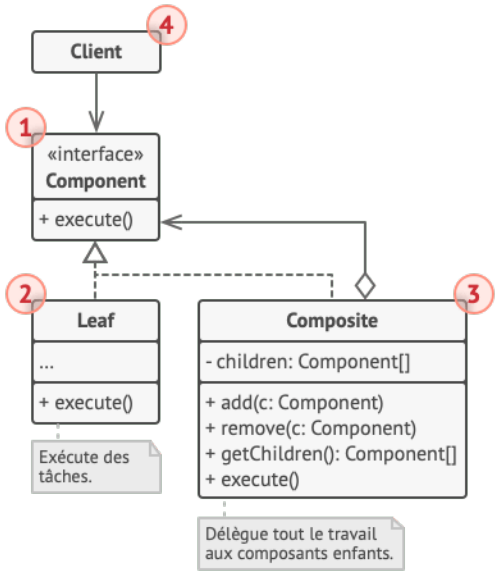
Design Pattern : Composite

Les patterns structurels s'occupent de la composition des classes et des objets. Ils permettent de former des structures flexibles et efficaces pour l'organisation des classes ou des objets.

Le design pattern Composite est un pattern structurel qui permet de composer des objets en structures arborescentes. Ce pattern permet de traiter de manière uniforme des objets simples (appelés feuilles ou leaf) et des groupes d'objets complexes (appelés composites).



1. L'interface **Composant** (Component) est l'interface abstraite commune pour tous les objets de la hiérarchie (feuilles et composites), elle décrit les opérations qui doivent être implémentées de manière uniforme pour tous les objets.

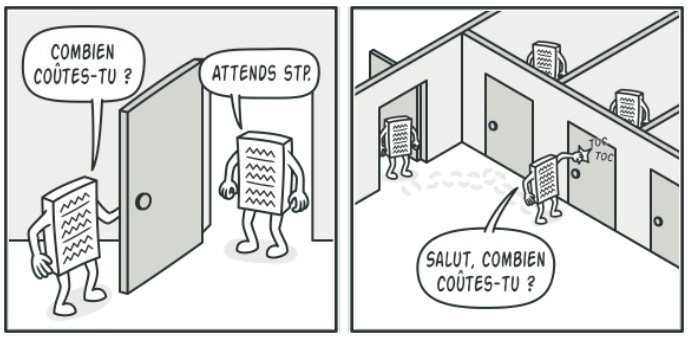
2. La **feuille** (Leaf) représente les objets simples qui n'ont pas de sous-éléments. En général les feuilles font le plus gros du travail, car elles n'ont personnes à qui le déléguer.

3. Le **Conteneur** (Composite) est un élément composé de sous-éléments : des feuilles ou d'autres conteneurs. Un conteneur ne connait pas les classes de ses enfants. Il passe par l'interface composant pour interagir avec ses sous-éléments (ajout, suppression, etc.) .

4. Le **Client** utilise les objets via l'interface composant sans se soucier de savoir s'il manipule une feuille ou un composite. Cela permet de traiter tous les objets de manière uniforme.

Traitement des opérations

Lorsqu'une opération est effectuée sur un objet composite, celle-ci peut être propagée à tous ses enfants de manière récursive. Par exemple, un appel à la méthode operation() sur un composite va entraîner l'appel de la même méthode sur chacun de ses enfants (qui peuvent également être des composites ou des feuilles).



Avantages et inconvénients

- ✅ Manipulation des hiérarchies complexes
- ✅ Ajout de nouveaux composant
- ✅ Flexibilité et extensibilité

- ❌ Gestion de la complexité
- ❌ Difficulté de mise en œuvre