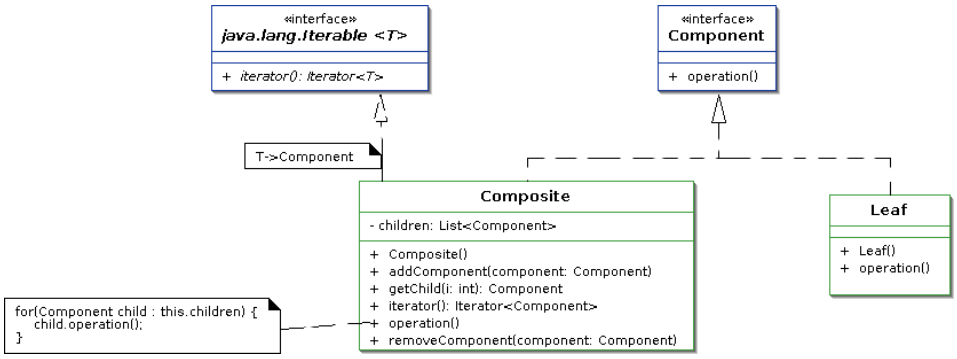
Design Pattern : composite

Intent

Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

Structure



Eléments caractéristiques

- il existe une ou plusieurs classes, sous-types de *Component*, qui ne sont pas des composites : *Leaf*.
- un *composite* est un sous-type de *Component* tel que :
  - il contient (a comme attribut) une collection d'objet du type *Componant* : ce sont les "composants fils" Ces éléments peuvent eux-mêmes être de type *Composite*. On peut ainsi construire des structures arborescentes de *Component*. Les feuilles de cette arbre sont de type *Leaf*.
  - la réalisation (traitement) de l'une des (ou plusieurs ou toutes) méthodes du type *composite* issue du type *Component* requiert l'invocation de cette même méthode sur les composants fils : c'est le cas de *operation()*.

On peut considérer qu'au niveau de *operation()* du *Composite* on a, du point de vue du type *Component*, un appel récursif (propagation de l'invocation de *operation()* dans la structure arborescente récursive construite via les *Composite*). Le cas d'arrêt de cette récursivité est réalisé par l'invocation sur un objet de type *Leaf*.

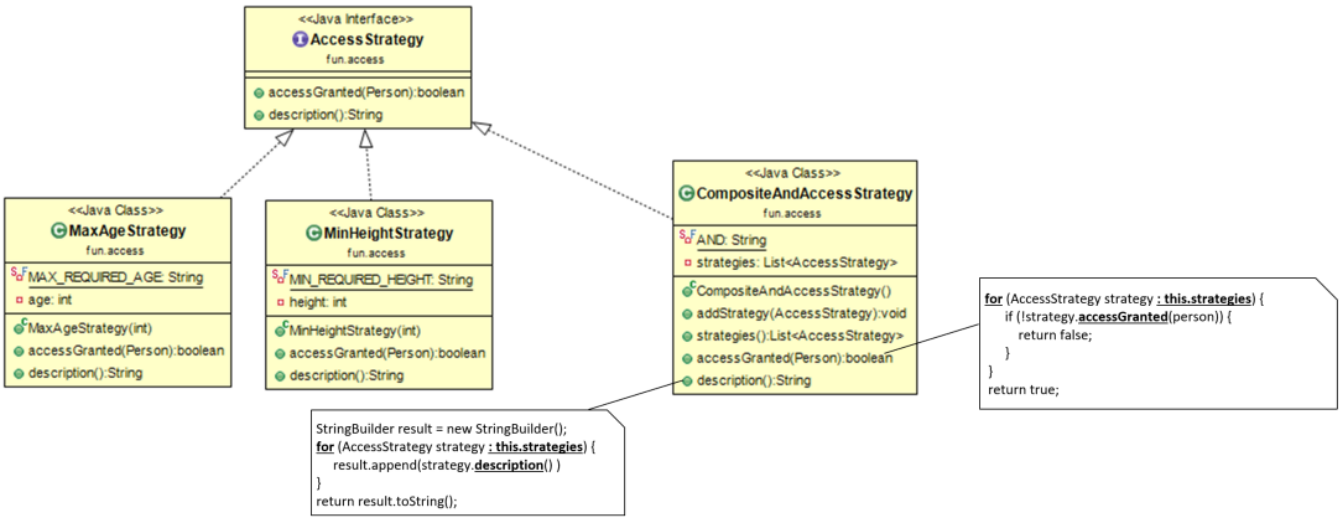
Exemples rencontrés

Dans l'api java Certains composants graphiques (les `Containers`) sont construits pas composition d'autres éléments graphiques plus "léger". Dessiner un tel élement graphique, par la méthode `paint`, nécessire de demander à chacun de ses composants "légers" de se dessiner :
Voici un extrait de la documentation de la méthode `paint(Graphics g)` de `java.awt.Container` :

> *Paints the container. This forwards the paint to any lightweight components that are children of this container. If this method is reimplemented, super.paint(g) should be called so that lightweight components are properly rendered. (...)*

Cette classe dispose également des méthodes :
public Component add(Component comp) *Appends the specified component to the end of this container. (...)*
public void remove(Component comp) *Removes the specified component from this container. (...)*

TD – Attractions. Les conditions d'accès multiples (`CompositeAndAccessStrategy`) sont obtenues par composition des stratégies de base. La méthode `accessGranted` de cette classe se résoud en itérant les invocations de cette même méthode sur chacun des *composants* stratégies. Il en est de même pour la méthode `description()`.



TD – Expressions : expressions binaires. Version un peu particulière du design pattern *Composite* dans la mesure où ici, l'expression binaire n'étant composée que de 2 expressions-composantes, il n'est pas naturel de proposer une liste de composants fils pour seulement 2 éléments. Cette liste est remplacée par 2 attributs explicites (les opérandes). Mais les opérations `eval()` et `format()` des expressions binaires s'appuient bien sur l'exécution de ces méthodes sur chacun des composants (`left/rightOperand`)