

Schémas de conception (*Design Patterns*)
Rappel “Composite” - Etude “Visiteur”

1 Schéma Composite appliqué aux fichiers et répertoires

Considérons l'exemple des éléments de stockage en mémoire secondaire fournis par un système d'exploitation : fichiers, liens symboliques et dossiers (répertoires). Pour ceux qui ne l'ont jamais fait, réalisez une représentation objet de ces éléments selon le schéma **COMPOSITE** en vue d'en programmer une **simulation**. Pour les autres, reprenez votre réalisation de l'année passée et remettez la éventuellement à jour ou imaginez une autre structure arborescente qui pourrait faire l'objet d'une utilisation a posteriori par le schéma visiteur. Par exemple vous pouvez prendre l'exemple d'un ordinateur en kit que l'on souhaiterait assembler, un tel ordinateur est, au niveau de l'acheteur, constitué de composants finaux (haut-parleur, ventilateur, disque dur) et de composants composites comme la carte mère qui est elle-même constitué de composants (processeur, mémoires), etc. Laissez libre votre imagination (Appliquer le schéma visiteur pour un document HTML représenté par une arborescence d'instances de classes représentant les différents sortes de composant d'une page).

Pour les éléments de stockage, on a les éléments de spécification et de conception suivants :

- un dossier (*Directory*) peut contenir des fichiers (*File*), des liens symboliques (*Link*) et des dossiers,
- un fichier possède un contenu que l'on représentera dans la simulation par une *String*.
- tout élément de stockage est contenu dans un dossier, sauf le dossier racine qui n'a pas de conteneur.
- tout élément de stockage possède un attribut `basicSize` indiquant l'espace de base qu'il occupe en mémoire quand il est vide (0 pour un fichier, 0 pour un lien, 4 pour un dossier).
- tous les éléments de stockage possèdent les méthodes :
 - `int size()` qui rend la taille occupée par l'élément au moment de l'invocation de la méthode.
 - `String absoluteAddress()` qui donne l'adresse absolue de l'élément. Dans notre simulation, cette méthode rend une string équivalente au résultat de la commande unix *pwd*.
 - `void ls()` avec les mêmes spécifications que la commande Unix (essayez).
- chaque élément de stockage possède des attributs et méthodes spécifiques :
 - pour un fichier ou un lien, `void cat()` qui affiche son contenu à l'écran, et `setContents(String)` qui permet de changer son contenu.
 - `int nbElem()` pour un répertoire ou pour un fichier rendant respectivement le nombre d'éléments du répertoire ou de mots du fichier,
 - `Collection find(String name)` pour un répertoire qui rend la collection des adresses absolues des éléments de nom `name` qu'il contient,
 - `Collection findR(String name)` pour un répertoire qui rend la collection des adresses absolues des éléments de nom `name` qu'il contient directement ou par transitivité.
 - pour un répertoire une méthode booléenne `include(Element)`

2 Etude du schéma Visiteur

Le schéma visiteur propose une solution pour le parcours non anticipé de structures de données complexes dans le but de les doter de fonctionnalités globales nouvelles, d'inspection ou de modification. Une fois le mécanisme de visite installé au sein de la structure de donnée et sans que son code source de cette dernière doivent être modifié, il est possible de réaliser un nombre quelconque de nouveaux visiteurs. Ce schéma est historique et largement utilisé, notamment dans le domaine du développement d'applications WEB à base de composants.

Lisez et comprenez le schéma visiteur tel que décrit dans le livre "Design Patterns, Elements of Reusable Software". Voici en premier lieu quelques questions relatives à ce texte.

1. En quoi les noeuds d'un arbre de syntaxe abstraite (c'est l'exemple utilisé dans le texte) relèvent-ils du schéma de conception "Composite". Donnez le schéma UML de la hiérarchie des classes de noeuds.

2. Quelles sont les exemples de visiteurs d'arbre de syntaxe proposés dans le texte ?
3. Par une instance de quelle classe un visiteur est-il globalement représenté ?
4. Combien de méthodes trouve-t-on sur un visiteur ?
5. Les visiteurs peuvent-ils manipuler la structure interne privée des objets qu'ils visitent ?
6. Etudiez particulièrement le code solution de la page 338.

3 Application à la structure arborescente des éléments de stockage.

1. Dotez les éléments de stockage d'un mécanisme de visite selon le schéma étudié précédemment.
2. Réaliser un `RazVisitor` qui remet à zéro la taille de tous les fichiers du répertoire visité et récursivement de tous les fichiers de tous ses sous-répertoires.
3. `JavaCleanVisitor` : Réaliser une classe `JavaCleanVisitor` qui permet de visiter un répertoire et ses sous-répertoires pour y détruire tous les fichiers dont le nom est suffixé par ".class".
Cet exercice pose le problème intéressant du parcours modificateur d'une collection.
4. `CountVisitor` : Une des questions que pose le schéma "visiteur" est de savoir comment rendre des résultats. Pour travailler cet exercice, on se propose d'écrire un visiteur capable de donner le nombre de fichiers dans un répertoire dont la taille dépasse une taille donnée. Pour cela, réaliser un visiteur à qui l'on puisse demander en premier lieu de visiter récursivement un répertoire pour faire le comptage et auquel on puisse ensuite envoyer le message `getCount()` qui rende le nombre calculé.
5. Voyez vous en Java une solution pour paramétrer ce visiteur par une fonction ? Ceci éviterait d'écrire une nouvelle classe pour chaque requête spécifique. (compter le nombre de fichiers vides ou le nombre de fichier préfixés par "." ou autre). Il est probable qu'une solution en Java passe par l'utilisation du package `Reflect`
6. Qu'est ce que le "double dispatch" évoqué dans l'article ? A quoi sert-il ?
7. Comparez, du point de vue des fonctionnalités, un structure arborescente dotée d'un visiteur et une collection Java dotée d'itérateurs prédéfinis.