Système de fichiers

A31

Automne 2020

1 Chemins virtuels

On se propose de modéliser un système de fichiers, ou plutôt un système de chemins et d'en faire une implantation virtuelle.

Un système de chemins est une structure **composite**, un arbre avec quelques traits particuliers.

Un chemin a un nom (une chaîne de caractères), un parent (un répertoire, voir ci-dessous).

Il y a deux sortes de chemins, les fichiers et les répertoires. Un répertoire est une *agrégation* de chemins alors qu'un fichier encapsule un objet (de type `String` pour simplifier). De plus, les répertoires ont une taille commune (par défaut, 4096 octets) et la taille d'un fichier est celle de la chaîne encapsulée.

Du côté des opérations, outre les accesseurs et les modificateurs, les chemins doivent pouvoir être supprimés et déplacés (changer de parent). En plus, les répertoires doivent pouvoir ajouter, ou enlever un arbre de leur liste d'enfants.

Il y a un chemin particulier qui n'a pas de parent. On le nomme « racine », ou simplement "/".

1. Faire le modèle du système de chemins qui vient d'être décrit.
2. Implanter ce modèle en Java. Ajouter une classe Main créant un petit système de fichiers et tester en mode débogage pour observer la structure arborescente.

2 Services

On se propose maintenant d'écrire une classe de services — une classe utilitaire — pour ces chemins. Cette classe doit proposer les services suivants :

3. nom pleinement qualifié du chemin ;
4. liste de tous les chemins descendants d'un ancêtre donné (la liste comprend l'ancêtre lui-même) ;
5. liste de tous les descendants portant un nom donné ;
6. taille totale d'un répertoire.
7. sérialisation du chemin.

Compléter votre classe Main pour tester tous ces services.

3 Fabrique

10. Plutôt que d'utiliser les constructeurs des fichiers et répertoires, écrire une classe fabrique avec deux méthodes dédiées à la création des fichiers et répertoires. Cette classe fabrique doit avoir une structure de **singleton**. Les méthodes fabrique doivent vérifier la validité de leurs arguments avant d'appeler les constructeurs associés.
11. Modifier votre fichier de test pour appeler maintenant les méthodes de la classe de fabrication.

4 Terminal contrôleur

12. Écrire une classe `Terminal` proposant les méthodes publiques suivantes :
    `touch(<nom_fichier>)`,
    `mkdir(<nom_répertoire>)`,
    `ls()`,

`pwd()`,
`cd(<nom_répertoire>)` ou `cd("..")`,
`rm(<nom_fichier>)`,
`rmdir(<nom_répertoire>)`,
`mv(<ancien_nom>, <nouveau_nom>)`
où `<nom_fichier>` et `<nom_répertoire>` sont des noms simples (sans "/" ou "\").

13. tester votre classe à l'aide de la classe `InteractiveShell` fournie sur Moodle. Attention, il faut lancer l'application dans un terminal, pas dans l'IDE.

Facultatif : Création d'une vue (modèle MVC historique : le contrôleur pilote la vue).

14. Faire de votre classe *Component* une réalisation de l'interface `TreeModel` sans mettre de code dans les trois méthodes dont le type de retour est `void` qui ne seront pas utilisées.
15. Créer une vue dont le panneau principal contient uniquement un `JTree` construit en lui passant en argument la racine de votre système (`this.add( new JTree( maRacine ) );`
16. A chaque fois qu'une commande est frappée dans le shell interactif, on supprime le `JTree` dans la vue (`maFenetre.getContentPane().remove(0);`) et on en créé un nouveau `JTree` (c'est un peu bourrin mais très simple).