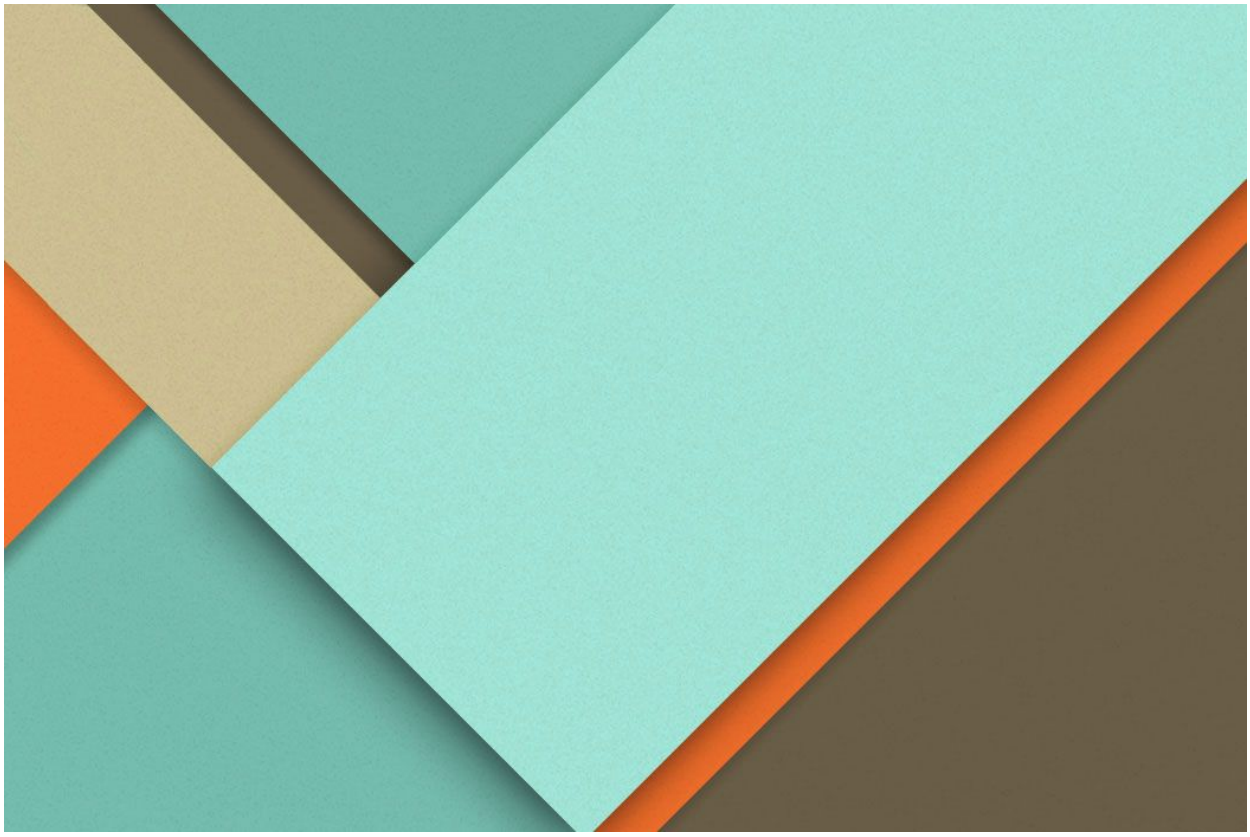
Rapport du projet
Conception et développement d'une application



Gestion d’un système de fichiers simplifié

2018-2019

Chargé de TP : Monsieur **KAHN GIACOMO**

Realisé par : **Fabrice BAZIRE & Elhadji SOW & Raghdah ELIAS**

Université d’Orléans

Sommaire

Mise en place

Lors de la première lecture du sujet, nous avons constaté qu'il s'agit d'un projet vaste auquel il faut faire attention, nous avons décidé de ne pas penser loin et de faire ce projet étape par étape comme il est indiqué sur le sujet.

Constructeurs utilisés

Nous avons utilisé un constructeur par défaut **ArbreFichiers** pour initialiser tous les objets ArbreFichiers à null. Ensuite, un constructeur avec paramètres pour affecter des valeurs à nos objets ArbreFichiers.

Explication du code étape par étape

Première étape

Première méthode :

Permet d'ajouter un fils (un dossier ou un fichier), en paramètres, nous avons le dossier/fichier à ajouter, on regarde si le premier fils du répertoire courant est null (Le premier fils du répertoire courant correspond au fils gauche).

S'il est null alors ça signifie que le répertoire courant est vide, donc nous avons plus qu'à ajouter le fils dans la variable fils gauche.

Sinon, si le répertoire courant n'est pas vide, on fait un parcours des fils du répertoire courant. Cela signifie que l'on doit regarder chaque élément (dossier/fichier) se trouvant dans le répertoire courant.

Afin de comparer les noms et trouver le bon emplacement entre 2 éléments pour ajouter le fils (car c'est rangé par ordre alphabétique donc il faut que l'ajout du fils soit au bon

endroit selon son nom). Et pour cela, on a besoin de 2 variables qui vont enregistrer les fils que l'on parcours.

Par exemple, on parcourt les éléments du répertoire courant (Tant que le nom du fils à ajouter est > au nom de l'élément que l'on regarde dans le répertoire courant), une fois que le parcours trouve l'élément qui a un nom supérieur au nom du fils à ajouter, le parcours s'arrête.

Et les 2 variables auront enregistré le dernier fils où le parcours s'est arrêté (donc le fils dont le nom est supérieur au nom du fils à ajouter) et l'avant-dernier fils qui a été parcouru

Donc, cela veut dire que nous avons trouvé l'endroit où ajouter le fils.

C'est entre ces 2 fils que tu as enregistré dans les 2 variables (finalement on ne les a pas utilisées).

Et pour ajouter le fils, on doit changer le frère droite du dernier fils qui a été enregistré à la fin du parcours (donc on met frere droite = fils en paramètre) et on change le frère gauche de l'avant-dernier fils qui a été enregistré à la fin du parcours (donc on met frere gauche = fils en paramètre)

Mais, Il y a aussi une condition à prendre en compte, c'est que si par exemple, tous les éléments du répertoire courant ont un nom < au nom du fils à ajouter, alors il faut ajouter le fils tout à la fin.

Donc cela veut dire que lorsque le parcours se finit, c'est qu'il n'y a plus de fils à parcourir, on aura parcouru tous les fils.

Ce qui signifie que la variable qui enregistrer le dernier fils à la fin du parcours, il sera null

Ce qui nous permettra de savoir que le fils à ajouter devra être ajouter tout à la fin.

Donc si cette variable est null, on prend la 2ème variable qui, elle, a enregistré le dernier fils du répertoire courant, et on change son frère droite avec le fils à ajouter.

Et il faut changer aussi le frère gauche et le frère droite du fils à ajouter une fois le parcours fini, changer le père du fils à ajouter qui sera juste this.père et mettre à jour la taille du répertoire courant.

C'était donc les idées du départ qui a été ensuite codé en JAVA mais pas tout à fait de la même manière.

Deuxième méthode :

Permet de supprimer un noeud.

Donc, pour supprimer le noeud, il n'y a aucun paramètre, donc le noeud à supprimer sera "this".

On vérifie d'abord si "this" est le premier fils du répertoire courant donc le fils gauche.

Si c'est le cas, on a juste à changer le fils gauche par son frère droite

Sinon, on refait un parcours des fils du répertoire courant, on prend 2 variables qui vont enregistrer le fils qu'on regarde et l'ancien fils qu'on a regardé, on le parcours jusqu'à trouver le fils qui est égal à "this".

Ensuite, une fois qu'on l'a trouvé, on a la 2ème variable qui a enregistré l'ancien fils regardé, et la 1ère variable est le fils égal à "this"

Donc si on veut supprimer "this", on change le frère droite de la 2ème variable (ce frère droite est "this" pour rappel), et on le change pour le frère droite de la 1ère variable (qui est le frère droite de "this").

Et si le frère droite de la 1ère variable est null (donc le frère droite de "this" est null) alors cela signifie que "this" est le dernier fils du répertoire.

Donc pour le supprimer, on change le frère droite de la 2ème variable (donc l'avant-dernier fils du répertoire) pour "null" et on met à jour la taille du répertoire courant.

C'était donc les idées du départ qui a été ensuite codé en JAVA mais pas tout à fait de la même manière.

Troisième méthode :

Dans cette méthode, il suffisait d'examiner de gauche à droite l'ensemble des fils du noeud sur laquelle la méthode est appelé comme le sujet l'indique. Effectivement, on a fait la méthode qui s'appelle **info** qui affiche les fils de l'objet appelant. On a utilisé la boucle **while (this.fils1 != null)** pour se mettre tout à gauche pour parcourir de gauche à droite.

Quatrième méthode :

Elle n'a aucun paramètre, donc cela signifie que pour avoir le chemin du répertoire courant, on utilisera "this" qui est le répertoire courant, on regarde si le père du répertoire courant est la racine et on sait que le nom de la racine est vide. Donc on regarde si le nom de la racine est vide.

Si oui, alors on retourne juste le nom du répertoire courant.

Sinon, on fait de la récursivité et on rappelle la fonction mais cette fois-ci avec le père du répertoire courant et on met cela dans une variable String.

Cela nous permettra de construire le chemin à l'envers à chaque boucle de récursivité.

Par exemple, si nous avons la racine à l'intérieur on a un dossier "d1" qui a un fichier "f1", la récursivité va se faire de cette manière :

1. /f1
2. d1/f1
3. /d1/f1

Fabrice BAZIRE a essayé de coder cette idée en JAVA mais, cela n'a pas fonctionné correctement. Ensuite, il a fait à sa manière mais cette fonction provoque un problème.

On pense que le problème ne vient pas de cette méthode mais vient de la **Première méthode** et le problème a une relation avec le pere qui n'enregistre pas correctement les étapes.

Cinquième méthode

C'est la méthode qui représente le "cd" en shell.

Elle prend en paramètre un nom de dossier pour rentrer à l'intérieur du dossier, mais si ce nom est ".." alors cela signifie qu'on sort du répertoire courant pour retourner dans le répertoire précédent.

Donc au début on vérifie si ce nom est ".." et on retourne le pere.

Si ce n'est pas le cas, on fait un parcours des fils du répertoire courant.

Si le fils est un dossier alors on retourne le fils.

sinon, on retourne null

C'était donc les idées du départ qui a été ensuite codé en JAVA mais pas tout à fait de la même manière.

Deuxième étape

Dans cette étape, on nous a demandé dans le sujet de faire la construction d'une arborescence de fichiers initiale à partir d'un fichier texte, et en effet, vous trouverez dans le code que l'on a pu effectuer en faisant une méthode qui depuis un fichier respectant le format indiqué dans le sujet, retourne l'arbreFichier correspondant. En suite, nous avons utilisé un compteur pour connaître le numéro de ligne en cours de lecture **ligneactuelle**.

Et aussi la chaine de caractere s, qui a pour but de récupérer la ligne en cours de lecture du fichier, et la boucle while fait en sorte que tant que le fichier n'est pas à la fin, s prend la ligne suivante.

Dès qu'on a la chaine de caractère, on utilise la commande **x=i.split(" ")** pour transformer cette chaîne qui contient toute la ligne en un tableau de mot pour analyser mot par mot.

Après, on regarde si le premier caractère du premier mot de la ligne est une **étoile**, alors on fait **arbre = arbre.getPere()**.

Ensuite, on regarde si le deuxième caractère du mot est un **d** dans ce cas on doit créer un dossier. Et finalement, on regarde si le deuxième mot est un **f**, dans ce cas on doit créer un fichier.

Troisième étape

C'est une méthode qui selon la commande passée en paramètre va l'exécuter sur l'arbreFichiers en paramètre.

Comme nous l'avons expliqué dans la méthode ci-dessus, on transforme la chaîne s (qui cette fois-ci la commande saisi par l'utilisateur) en tableau de mots **string[] x = s.split(" ").**

Ensuite, on traite la commande saisi par ce dernier. On a fait en sorte que l'utilisateur puisse entrer le contenue du fichier qu'il veut créer, ensuite on crée le fichier avec le nom et le contenu que l'utilisateur a choisi.

Maintenant, si les 3 premiers caractères sont des _, alors on saute une ligne puis on affiche le contenu de la ligne.

Et enfin, si les 3 derniers caractere sont des _, alors on affiche la ligne puis on saute une ligne, sinon on affiche simplement la ligne.

Quelques tests

Le code

```
public static void main (String[]args){
    //Test des méthodes add et info
    ArbreFichiers test = new ArbreFichiers(null,null,null,null, "root", false, null, 0);
    test.add(new ArbreFichiers(null,null,null,null, "francais", true, "bonjour", "bonjour".length()));
    test.add(new ArbreFichiers(null,null,null,null, "allemand", false, null,0));
    test.add(new ArbreFichiers(null,null,null,null, "bool", false, null,0));
    test.add(new ArbreFichiers(null,null,null,null, "chaise", false, null,0));
    test.add(new ArbreFichiers(null,null,null,null, "anglais", false, null, 0));
    test.fils1.add(new ArbreFichiers(null,null,null,null, "italie", true, "heyé", "heye".length()));
    test.add(new ArbreFichiers(null,null,null,null, "suedois", true, "hej", "hej".length()));
    test.add(new ArbreFichiers(null,null,null,null, "suedois", false, null, 0));
    test.fils1.add(new ArbreFichiers(null,null,null,null, "suisse", false, null, 0));
    test.fils1.fils1.add(new ArbreFichiers(null,null,null,null, "danois", false, null, 0));
    test.fils1.fils1.fils1.add(new ArbreFichiers(null,null,null,null, "amical", true, "hey", "hey".length()));
    System.out.println("danois : " + test.fils1.fils1.fils1.size);
    System.out.println("suisse : " + test.fils1.fils1.size);
    System.out.println("suedois : " + test.fils1.size);
    System.out.println(test.info_branche());
    System.out.println(test.fils1.name);
    System.out.println(test.info());
    System.out.println(test.fils1.info());
    System.out.println(test.fils1.name);
    System.out.println(test.fils1.pere.name);
    System.out.println(test.fils1.fils1.name);
    System.out.println(test.fils1.fils1.pere.name);
}
```

L’affichage

```
danois : 3
suisse : 0
suedois : 0
root/
suedois
dossier :  allemand (taille : 0 kb)
dossier :  anglais (taille : 4 kb)
dossier :  bool (taille : 0 kb)
dossier :  chaise (taille : 0 kb)
fichier :  francais (taille : 7 kb)
fichier :  suedois (taille : 3 kb)
dossier :  suedois (taille : 0 kb)dossier :  danois (taille : 3 kb)


dossier :  suisse (taille : 0 kb)fichier :  amical (taille : 3 kb)


suedois
root
suisse
```

Répartition du travail

Nous avons commencé par la recherche des idées pour la première étape avant d'écrire la base du projet. Celles-ci a été effectuée par **Raghdah ELIAS.**

Ces idées ont été expliquées au fur et à mesure dans la **Première étape.**

Elles n'ont pas été codé de la même manière comme expliqué ci-dessus.

Fabrice BAZIRE a commencé à coder directement en JAVA à sa manière. Et on revenait aux idées de **Raghdah ELIAS** régulièrement pour essayer d'améliorer le code.

Lors de notre rencontre en travaillant ensemble, **Elhadji SOW** a participé en donnant son avis pour améliorer le code et effectivement nous avons pu avancer en travaillant ensemble.

Ensuite, **Fabrice BAZIRE** avait commencé à coder la **Deuxième étape** à sa manière et lors de la rencontre en groupe, **Fabrice BAZIRE** et **Elhadji SOW** ont pu profiter pour s'échanger les idées afin d'avancer sur la **Deuxième étape** et commencer la **Troisième étape**.

Raghdah ELIAS s'est chargé du code de la première étape pour corriger ce qui n'allait pas.

Finalement, nous avons trouvé que ce sujet était intéressant de ce qu'il contenait comme informations à chercher. Nous avons senti qu'on a progressé d'une part sur notre pensé et réflexion et d'autre part sur la programmation en JAVA.