RAPPORT

FileParser :

Pour pouvoir interpréter le fichier d'arborescence nous avons choisit de dédier une classe à cette fonction. FileParser permet donc de parser un fichier afin d'obtenir une arborescence d'ArbreFichier utilisable.

Tout d'abord, tous les messages d'exceptions sont stockés dans des constantes tout en haut de la classe afin de pouvoir les retrouver facilement.

Ensuite, pour pouvoir garder une trace du fichier courant, nous avons une variable currentFile que nous mettons à une instance de Fichier lors de la rencontre d'une ligne nous indiquant la création d'un fichier, ou alors à null lorsque nous avons mis du contenu et que nous exécutons une nouvelle instruction de notre fichier arborescence.

Concernant la trace des dossier, nous avons choisi d'implémenter une Pile<Dossier>. Ce choix nous semblait être le plus judicieux car on peut empiler et dépiler des dossiers à notre guise, nous assurant ainsi la trace parfaite du chemin actuel.

Pour pouvoir parser une ligne nous avons décidé d'utiliser un pattern REGEX nous permettant d'éviter toutes les lignes non correctes et de lever une exception le cas échéant.
Nous avons également utiliser une Pile<String> ou l'on peut empiler le mot "fin" nous permettant ainsi de la dépiler à chaque fois que nous rencontrons le mot "fin" dans le fichier d'arborescence et donc de dépiler la Pile<Dossier>.

Enfin, nous avons également une List<String> de mots réservés. Cette liste nous permet de stocker des mots interdits, c'est à dire ne pouvant pas être en début de contenu d'un fichier.

SystemeFichier :

Pour pouvoir gérer les inputs utilisateurs, nous utilisons une classe SystemeFichier.
Cette dernière contient en attributs un Dossier arborescence contenant toute l'arborescence actuelle; une HashMap<String, Commande> commandes nous permettant de stocker les différentes commandes par leur nom (c'est à dire la commande à taper dans le terminal) ainsi que leur fonctionnalité, stockée dans une classe différente pour chaque commandes, toutes implémentant l'interface Commande. En effet, cette interface associé aux différentes classes est en réalité une application du design pattern Stratégie.

L'interface contient donc une méthode execute(Dossier currDir, String[] args) et retourne le type Optional<T> nous permettant de vouloir ou non retourner une valeur. Nous avons estimé que l'utilisation du type Optional<T> était le plus approprié au vu du fait que toutes les commandes ne retournent pas forcément des valeurs.

L’utilisation du design pattern Stratégie nous permet donc dans la classe SystemeFichier de récupérer l’input utilisateur et de regarder pour cette clé dans la Map<String, Commande> quelle commande est disponible. En récupérant la commande on peut ensuite directement utiliser la méthode de l’interface execute() sans avoir à se soucier de ce qu’elle réalise.

ArbreFichiers:

On retrouvera dans cette classe Abstraite les méthodes communes aux classes Dossier et Fichier, la distinction entre Dossier et Fichier est importante car certaine méthodes de Dossier (par exemple ajouterNoeud) ne doit pas pouvoir être appelé sur un Fichier.

Pour ce qui est de la manipulation des objets ArbresFichiers, les méthodes ont étés pensées pour être flexible si des changements venaient à venir comme par exemple la définition d'un "premier fils".
La méthode parcoursLargeurFrere permet selon son utilisation de récupérer par exemple le frère le plus à gauche de façon constante même si l'on définit que le premier fils est, par exemple, tout à droite.

IArbreFichiers:

Nous avons choisi d’implémenter une interface IArbreFichier nous permettant de proposer seulement certains services pour les ArbreFichiers. Grâce à cela, nous pouvons travailler directement sur cette interface au lieu de travailler spécifiquement sur un fichier ou sur un dossier.