Université de Paris **Programmation Orientée Objet**
L2 **Année 2022-2023**

TP n°8

Exceptions et classes internes

Arborescence de fichiers

Le but de cette partie est de réaliser un programme capable d'afficher sous forme d'arborescence le contenu d'un répertoire de fichiers (dans le vrai système de fichiers). Pour ce faire, nous aurons besoin des classes `File` et `FileNotFoundException`, qui se trouvent dans le package `java.io`.

Nous allons commencer par créer une classe `Arbre` qui représente le contenu d'un répertoire.

Exercice 1 Le modèle
Écrire la classe `Arbre`, avec les attributs suivants :

- Une classe interne `Noeud`, qui représente un noeud de l'arborescence (c'est à dire un fichier), et qui contiendra un champ `nom` de type string, un champ `taille` de type entier, et un champ `repertoire` de type bool. On ajoutera également un champ `fils` de type `ArrayList<Noeud>` qui représente les fichiers contenus dans ce noeud, si ce noeud est un répertoire (il sera à `null` dans le cas contraire).
- Un attribut `Noeud racine`, le répertoire représenté par cet arbre.

Exercice 2 Création de l'arbre
Ajouter à la classe `Noeud` un constructeur qui prend un paramètre un objet `File` et qui effectue les opérations suivantes :

- Si le fichier correspondant n'existe pas, lever une `FileNotFoundException`
- Sinon, initialiser les champs `nom`, `taille` et `repertoire`. Si le fichier concerné est un répertoire, alors on initialise le membre `fils` et on le remplit récursivement. Sinon, `fils` est laissé à `null`.

On ajoutera enfin un constructeur à la classe `Arbre` qui prend en paramètre une chaîne de caractères représentant le chemin de la racine de l'arbre. Si le chemin n'existe pas, on lèvera une `FileNotFoundException`.

Aide : Consulter la javadoc de la classe `File` pour plus d'informations. On se servira notamment des méthodes `exists()`, `getName()`, `length()`, `isDirectory()` et `listFiles()`.

Exercice 3 Affichage
Écrire une méthode `void afficher()` dans la classe `Arbre`, qui affiche l'arbre de la manière suivante :

```
racine [15]
  fichier1.txt [100]
  fichier2 [200]
  rep1 [200]
    fichier3 [0]
    fichier4.txt [5]
  rep2 [100]
    rep3 [100]
     fichier5.txt [100]
```

Où chaque noeud est affiché à sa profondeur, avec son nom suivi de sa taille entre crochets.

Exercice 4 Transformations de l'arbre et expressions lambda

Étant donné un `Arbre a`, et une fonction quelconque `String transf (String s)`, on veut écrire une procédure qui applique `transf` aux noms de tous les noeuds de `a` qui ne sont pas des répertoires. (**Remarque :** cette procédure ne devrait évidemment pas dépendre de ce que fait `transf`.)

- Écrire une interface `StringTransformation` qui contient une seule méthode abstraite `String transf(String s)`. (**Remarque :** On peut bien sûr remplacer `String` par un `T` générique.)
- **Échauffement :** Définir (dans la méthode `main` de votre programme) en utilisant une expression lambda, la transformation `StringTransformation addBlah`, qui ajoute `".blah"` à une chaîne de caractères. Le tester.
- La méthode `forEach` de `ArrayList` peut prendre en argument une expression lambda. Déduire (en consultant la javadoc) ce qu'elle fait dans ce cas.
- Dans `Noeud`, définir une procédure `map (StringTransformation t)` qui, lorsque le noeud est un fichier, applique la transformation `t` à son nom. Lorsque le noeud est un répertoire, on appellera `map` sur tous ses fils. **Indice :** Utiliser `forEach` avec une expression lambda.
- Définir une procédure `map (StringTransformation t)` dans `Arbre` qui applique `t` à tous les fichiers d'un arbre. Le tester avec `addBlah`. L'exemple précédent donnera :

  ```
  racine [15]
    fichier1.txt.blah [100]
    fichier2.blah [200]
    rep1 [200]
      fichier3.blah [0]
      fichier4.txt.blah [5]
    rep2 [100]
      rep3 [100]
       fichier5.txt.blah [100]
  ```

Exercice 5 Traverser de l'arbre

— Écrire une méthode `void traverser(String extension)` dans la classe `Arbre`, qui traverse l'arbre et affiche les fichiers qui terminent par `extension` de la manière suivante :

```
fichier1.txt [100]
fichier4.txt [5]
fichier5.txt [100]
```

Où chaque noeud est affiché avec son nom suivi de sa taille entre brackets.

— Écrire une méthode `void supprimer(String extension)` dans la classe `Arbre`, qui traverse l'arbre et supprime les fichiers qui terminent par `extension`. Si on a pas pu supprimer un fichier, on lèvera une exception avec le message `UnableToDeleteFileException` (on doit créer une classe qui hérite de la classe `Exception` qui se trouvent dans le package `java.lang` pour afficher cette message).

Attention : Pour ne pas risquer de supprimer des fichiers importants par accident sur votre système de fichiers personnel, *on ne va pas* utiliser la fonction `delete` de `File`, mais seulement supprimer le noeud dans la représentation interne de Java. Pour tester si on *pouvait* supprimer le fichier on va faire comme suit : On suppose qu'on peut supprimer le fichier si on a la permission d'écriture pour son dossier parent : `f.getParentFile().canWrite()`.