Université Paris Cité **Programmation Orientée Objet**
Année 2023-2024

TP n°8 :

Exceptions et classes internes

Arborescence de fichiers

Le but de ce TP est de réaliser un programme capable de manipuler et 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` [1] et `FileNotFoundException` [2], qui se trouvent dans le package `java.io`.

Exercice 1 (Le modèle)

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

1. 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), avec les attributs suivants :
     - Un attribut `nom` de type `String` représentant le nom du fichier.
     - Un attribut `taille` de type `int` représentant la taille du fichier.
     - Un attribut `repertoire` de type `bool` permettant d'indiquer que le fichier représente un répertoire.
     - Un attribut `fils` de type `ArrayList<Noeud>` qui représente les fichiers contenus dans le répertoire. Notez que cet attribut doit donc être initialisé à `null` lorsque le noeud n'est pas un répertoire.
   - Un attribut `racine` de type `Noeud` contenant le répertoire représenté par cet arbre.

Nous voulons maintenant pouvoir créer des `Arbre`s directement depuis un chemin du système de fichiers.

2. Dans la classe `Noeud`, ajouter un constructeur prenant en 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`.
3. Dans la classe `Arbre`, ajouter un constructeur qui prend en paramètre un chemin (sous la forme d'une chaîne de caractères) désignant la racine de l'arbre à construire. Si le chemin n'existe pas, on lèvera une `FileNotFoundException`.

Aide : N'oubliez pas de consulter la javadoc des classes `File`[1] et `FileNotFoundException`[2] pour plus d'informations. On se servira notamment des méthodes `exists()`, `getName()`, `length()`, `isDirectory()` et `listFiles()`.

Exercice 2 (Affichage d'`Arbre`)

Maintenant que nous avons fini de construire le modèle du système de fichiers, concentrons-nous sur son affichage.

1. (Documentation) `https://docs.oracle.com/javase/8/docs/api/java/io/File.html`
2. (Documentation) `https://docs.oracle.com/javase/8/docs/api/java/io/FileNotFoundException.html`

1. Dans la classe interne `Noeud`, écrire une procédure `void afficher(int profondeur)` qui affiche le `nom` du `Noeud` suivit de sa `taille` entre crochets. L'ensemble doit être précédé de 2×`profondeur` espace(s). Par exemple, pour un `Noeud monNoeud` dont le `nom` est `"LaReponseUniverselle.txt"` et la `taille` est `42`, l'appel `monNoeud.afficher(3)` génère l'affichage suivant :

   ␣␣␣␣␣␣`LaReponseUniverselle.txt␣[42]`

2. Dans la classe `Arbre`, écrire une procédure `void afficher()` 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]
```

   Chaque `Noeud` doit être affiché à sa profondeur, avec son `nom` suivi de sa `taille` entre crochets.

Exercice 3 (Transformations d'`Arbre` et expressions lambda)
Nous souhaitons maintenant construire des fonctionnalités plus avancées. Par exemple, nous aimerions avoir la capacité de modifier les noms de tous les fichiers présents dans un répertoire, à condition qu'ils ne soient pas eux-mêmes des répertoires. De manière plus générale, nous aimerions être en mesure d'appliquer des transformations quelconques à tous les fichiers qui ne sont pas des répertoires.

1. Écrire une interface `StringTransformation` contenant une méthode abstraite `String transform(String str)`.
   Remarque : Par la suite, on pourra bien sûr remplacer `String` par un type `T` générique.
2. Dans la méthode `main` de votre programme, définir une transformation `StringTransformation addBlah` qui ajoute `".blah"` à une chaîne de caractères. Pour cela, utilisez une expression lambda. N'oubliez pas de tester votre transformation.

Maintenant que nous avons défini une transformation, nous cherchons à l'appliquer à l'arbre. Remarquez que la classe `ArrayList`[3] possède une méthode `forEach` qui prend en argument une expression lambda et l'appliquer un à un à tous les éléments de la liste.

3. Dans la classe `Noeud`, définir une procédure `mapOnFiles(StringTransformation transf)` qui, lorsque le noeud est un fichier, applique la transformation `transf` à son nom. Lorsque le noeud est un répertoire, on appellera `mapOnFiles` sur tous ses fils.
4. Dans la classe `Arbre`, définir une procédure `mapOnFiles(StringTransformation transf)` qui applique `transf` à tous les fichiers d'un arbre. N'oubliez pas de tester cette procédure avec la `StringTranformation addBlah`. En particulier, l'`Arbre` de l'exemple précédent est transformé en l'`Arbre` suivant :

```
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]
```

3. (Documentation) `https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html`

Exercice 4 (Parcours d'`Arbre`)
Nous cherchons maintenant à n'afficher que

1. Dans la classe `Arbre`, écrire une procédure `void traversal(String extension)` qui traverse l'arbre et affiche les fichiers qui terminent par `extension`. Par exemple, l'appel `traversal(".tex")` sur l'`Arbre` donné dans le premier exemple génère l'affichage suivant :

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

Nous voulons aussi avoir la possibilité de supprimer les fichiers ayant une certain extension.

2. Créer une classe `UnableToDeleteFileException` qui hérite de la classe `Exception` se trouvant dans le package `java.lang`.
3. Dans la classe `Arbre`, écrire une procédure `void delete(String extension)` qui parcours l'`Arbre` et supprime les fichiers se terminant par `extension`. Si on a pas pu supprimer un fichier, on lèvera une exception `UnableToDeleteFileException` accompagnée d'un message explicatif.

   ⚠ **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()`.