Travaux Dirigés n°7

Java Avancé

—M1—

Classes internes, anonymes etc et NIO2

Interfaces, classes internes et anonymes... Classes NIO

- Pour ce TD, créez un nouveau projet Maven dans votre repertoire `ja` (voir td00) avec les `groupid` et `artifactId` suivants.
  - groupId `fr.dauphine.ja.`*`nomprenom`*`.nio`
  - artifactId `nio`
- N'oubliez pas de commiter régulièrement et de charger les dernières versions de votre dépot avec git push !

Instructions pour utiliser les tests fournis dans ce TD :

- Vous aurez besoin de la version 4.0 de JUnit. Éditez le fichier `pom.xml` de votre projet et mettez à jour la version minimale requise de JUnit depuis 3.8.1 vers 4.0.
- Pour ajouter un des fichiers test fournis à votre projet, il faut charger le fichier test dans le répertoire `src/test/java/fr/dauphine/javaavance/nomprenom/iterable/` et de modifier la ligne `package` du fichier test pour qu'elle corresponde à la ligne suivante.

```
1 package fr.dauphine.javaavance.nomprenom.iterable;
```

Dans ce TD, on utilisera uniquement les nouvelles entrées sorties (NIO2) pour manipuler dossiers et fichiers (depuis Java 7). Les interfaces utiles sont :

- `Path` (qui représente un chemin (un fichier, un répertoire etc)).
- `Paths` pour créer un Path.
- `Files` avec toutes les méthodes static utiles pour manipuler des `Path`.

Son but est de remplacer les classes liées à `File` de la très ancienne API IO. Avec les NIO2, la manipulation des répertoires, selon les systèmes est facilité, et des fonctionnalités sont ajoutées (gestion des liens physique et symboliques, des attributs de fichiers, notification de changements, parcours d'un répertoire, copie/déplacement...).

Le tutoriel Oracle sur les NIO2 : `https://docs.oracle.com/javase/tutorial/essential/io/fileio.html`.

— Pour ce TD, créez un nouveau projet Maven dans votre repertoire `javaavance` (voir td00) avec les `groupid` et `artifactId` suivants.
  — groupId `fr.dauphine.javaavance.`*`nomprenom`*`.nio2`
  — artifactId `nio2`
— N'oubliez pas de commiter régulièrement et de charger les dernières version de votre dépot avec git push !

▸ Exercice 1. Fichiers

1. Créer une classe `DirMonitor` et son constructeur prenant un Path (un répertoire). Celui-ci lève une exception si le Path n'est pas un répertoire ou n'est pas readable.
2. Écrire une méthode affichant tous les fichiers et répertoires du répertoire. Il y a dans `Files` une méthode `newDirectoryStream` qui renvoie un itérable sur des `Path`. Tester avec le répertoire courant (".").
3. Écrire une méthode `sizeOfFiles` renvoyant la somme des octets des fichiers (pas les répertoires) du répertoire.
4. Écrire une méthode `mostRecent` renvoyant le fichier ou répertoire le plus récent (au sens dernière modification).
5. Tester dans un main (si votre répertoire pour tester est ".", il s'agit du répertoire où se trouve votre projet eclipse) et avec ces tests JUnit DirMonitorTest.

▸ Exercice 2. Filtres et factorisation

1. On veut modifier le méthode affichant le contenu d'un répertoire afin d'y ajouter un filtre n'affichant que les fichiers ayant une taille d'au moins n octets, où n est spécifié à la création du filtre. La méthode `newDirectoryStream` est surchargée, rappeler ce que cela veut dire.
   (a) Écrire une classe `PrefixFilter` implémentant `DirectoryStream.Filter<Path>`, et modifier la méthode d'affichage afin qu'elle utilise ce filtre.
   (b) Transformer votre classe `PrefixFilter` en classe interne.
   (c) Transformer votre classe en classe anonyme dans votre méthode d'affichage.
2. On souhaite maintenant que le filtre puisse être utilisé pour les méthodes concernant la somme des tailles et du fichier le plus récent. Pour cela, écrire une méthode `public void applyAction(String prefix, MyAction action) throws IOException;` qui sera utilisée par chacune de vos méthodes précédentes. L'interface `MyAction` est définie de la manière suivante :

```java
interface MyAction {
  void perform(Path p) throws IOException;
}
```

(a) Ré-écrire la méthode d'affichage pour qu'elle utilise `applyAction`, et où l'action d'affichage est implémentée sous forme de classe anonyme implémentant `MyAction`.

(b) Est-ce intéressant de stocker la classe anonyme dans un champ ?

(c) Ré-écrire la méthode de somme des tailles en utilisant une classe interne pour l'action. Pourquoi il n'est pas possible d'utiliser une classe anonyme ici ?

(d) Ré-écrire la méthode du fichier le plus récent en utilisant une classe interne de méthode.

(e) Tester avec ces tests JUnit DirMonitorFilterTest.