Le langage Java

Traitement de fichiers

Programme détaillé ou sommaire

Le Traitement des fichiers avec l'API NIO 2
- La classe Paths et l'interface Path
- La classe Files
- Lecture d'un fichier
- Ecriture d'un fichier

TP

Chapitre 1
Présentation

L’API java.io

Une **API** est un acronyme avec une signification assez large :

- Signifie **A**pplication **P**rogramming **I**nterface
- Désigne une librairie (ensemble de classes), ou,
- Désigne un ensemble d’URL fournissant des données, ou,
- Désigne un ensemble d’interfaces qu’il faut implémenter

L’API **java.io** est l’API historique du langage JAVA, assez bas niveau et verbeuse.

Aujourd’hui vous pouvez utiliser l’API **NIO 2** pour des besoins simples

L’API NIO 2

L’API java NIO 2 est un ensemble de classes du package **java.nio.file** et qui permet de réaliser des opérations sur les fichiers et les répertoires.

Apparu en **Java 7**

Parmi ces opérations:

- Vérifier si un fichier existe, ou s’il est accessible en lecture/écriture
- Copier, modifier, supprimer ou déplacer un fichier
- Créer, modifier ou supprimer un répertoire
- Lire le contenu d’un fichier pour l’exploiter

Chapitre 2
Les principales classes

La classe Paths

La manipulation des ressources, fichiers et répertoires, passent par la notion de chemin (Path en anglais).

La première opération consiste donc à récupérer le chemin d'accès à la ressource.

Méthode static get(String) :

```
Path path = Paths.get("C:/Temp/recensement.csv");
```

Les informations fournies par le Path

```
Path path = Paths.get("C:/Temp/recensement.csv");

System.out.println(path.getRoot());
System.out.println(path.getParent());
System.out.println(path.getFileName());
```

```
C:\
C:\Temp
recensement.csv
```

La classe Files

La classe **java.nio.file.Files** permet :

- de fournir des informations sur un fichier, ou un répertoire
- de réaliser l'ensemble des manipulations possibles
  - lecture
  - création
  - modification
  - copie
  - déplacement
  - suppression

Vérifier si la ressource existe

Vérifier si la ressource (fichier ou répertoire) existe :

```
boolean exists = Files.exists(path);
```

Vérification des propriétés de la ressource

Vérifier si la ressource est un fichier et si ce fichier est lisible :

```
Path path = Paths.get("C:/Temp/recensement.csv");
boolean estFichier = Files.isRegularFile(path);
boolean estLisible = Files.isReadable(path);
```

Vérifier si la ressource est un répertoire et si ce répertoire est lisible :

```
Path path = Paths.get("C:/Temp");
boolean estFichier = Files.isDirectory(path);
boolean estLisible = Files.isReadable(path);
```

Copie d'un fichier

Il faut utiliser 2 « paths », le path d'origine qui correspond au fichier à copier, et le path de destination qui correspond au futur fichier.

```
Path pathOrigine = Paths.get("C:/Temp/recensement.csv");
Path pathDestination = Paths.get("C:/Temp/Work/recensement2.csv");

Files.copy(pathOrigine, pathDestination);
```

Attention, si le path de destination correspond à un fichier existant :

```
Exception in thread "main" java.nio.file.FileAlreadyExistsException: C:\Temp\Work\recensement2.csv
```

Copie d’un fichier avec option

Une option permet d’écraser la destination si elle existe :

```
Path pathOrigine = Paths.get("C:/Temp/recensement.csv");
Path pathDestination = Paths.get("C:/Temp/Work/recensement2.csv");

Files.copy(pathOrigine, pathDestination, StandardCopyOption.REPLACE_EXISTING);
```

Déplacement d'un fichier

Il faut utiliser 2 « paths », le path d'origine qui correspond au fichier à copier, et le path de destination qui correspond au futur fichier.

```
Path pathOrigine = Paths.get("C:/Temp/recensement.csv");
Path pathDestination = Paths.get("C:/Temp/Work/recensement2.csv");

Files.move(pathOrigine, pathDestination);
```

Attention, si le path de destination correspond à un fichier existant :

```
Exception in thread "main" java.nio.file.FileAlreadyExistsException: C:\Temp\Work\recensement2.csv
```

Déplacement d'un fichier

Pour écraser le fichier de destination s'il existe :

```
Path pathOrigine = Paths.get("C:/Temp/recensement.csv");
Path pathDestination = Paths.get("C:/Temp/Work/recensement2.csv");

Files.move(pathOrigine, pathDestination, StandardCopyOption.REPLACE_EXISTING);
```

Déplacement d'un répertoire avec ses fichiers

Pour écraser le fichier de destination s'il existe :

```
Path pathOrigine = Paths.get("C:/Temp/monRepertoire");
Path pathDestination = Paths.get("C:/Temp/Work/monRepertoire");

Files.move(pathOrigine, pathDestination, StandardCopyOption.REPLACE_EXISTING);
```

Attention, si le répertoire de destination n'est pas vide :

```
Exception in thread "main" java.nio.file.DirectoryNotEmptyException: C:\Temp\Work\monRepertoire
```

Suppression d'un fichier

Pour supprimer un fichier:

```
Path path = Paths.get("C:/Temp/monFichier.txt");
Files.delete(path);
```

Attention, si la ressource cible n'existe pas :

```
Exception in thread "main" java.nio.file.NoSuchFileException: C:\Temp\monFichier.txt
```

Suppression d’un fichier s’il existe

Pour supprimer un fichier seulement s’il existe:

```
Path path = Paths.get("C:/Temp/monFichier.txt");
Files.deleteIfExists(path);
```

Lister les fichiers d'un répertoire (1/2)

Pour lister le contenu d'un répertoire :

```
Path pathDirectory = Paths.get("C:/Temp/");

DirectoryStream<Path> stream = Files.newDirectoryStream(pathDirectory);
for (Path path: stream) {
    System.out.println(path);
}
```

Lister les fichiers d'un répertoire (2/2)

Pour lister le contenu d'un répertoire :

```
Path pathDirectory = Paths.get("C:/Temp/");

DirectoryStream<Path> stream = Files.newDirectoryStream(pathDirectory);
Iterator<Path> iterator = stream.iterator();
while (iterator.hasNext()) {
     Path path = iterator.next();
     System.out.println(path);
}
```

Pour lire le contenu d'un fichier

Exemple pour lire dans un fichier:

```
Path pathFile = Paths.get("C:/Temp/monFichier.txt");
List<String> lines = Files.readAllLines(pathFile, StandardCharsets.UTF_8);
```

Pour écrire dans un fichier

Exemple pour écrire dans un fichier:

```
List<String> lines = new ArrayList<>();
lines.add("Coucou");
lines.add("Hello");

Path pathCible = Paths.get("C:/Temp/monFichier.txt");
Files.write(pathCible, lines);
```

Si le fichier a déjà un contenu, ce dernier est écrasé.

Pour compléter un fichier

Exemple pour ajouter des lignes en fin d'un fichier:

```
List<String> lines = new ArrayList<>();
lines.add("Coucou");
lines.add("Hello");

Path pathCible = Paths.get("C:/Temp/monFichier.txt");
Files.write(pathCible, lines, StandardOpenOption.APPEND);
```

Si le fichier a déjà un contenu, ce dernier est complété.

Pour écrire dans un fichier en UTF-8

Les options sont cumulables, je peux indiquer à la fois une complétion de fichier et une écriture en UTF-8:

```
List<String> lines = new ArrayList<>();
lines.add("Coucou");
lines.add("Bienvenue sur mon île");

Path pathCible = Paths.get("C:/Temp/monFichier.txt");
Files.write(pathCible, lines, StandardCharsets.UTF_8,
StandardOpenOption.APPEND);
```

La gestion des exceptions (non encore vue)

La plupart des méthodes vues précédemment génère une exception de type **IOException**.

A ce stade, nous nous contenterons d'ajouter une clause à la méthode main

```
public static void main(String[] args) throws IOException {
     Path pathFile = Paths.get("C:/Temp/recensement.csv");

     List<String> lines = Files.readAllLines(pathFile, StandardCharsets.UTF_8);
     for (String line: lines) {
          System.out.println(line);
     }
}
```

Atelier (TP)

Objectifs du TP: lire le contenu d'un fichier et généré un nouveau fichier

Description du TP:

Dans ce TP, vous allez apprendre à lire le contenu d'un fichier, en extraire des informations et générer un nouveau fichier avec les données utiles.