Programmation Orientée Objet Java

I/O

v1.0 - 28.02.2023

Plan

- Introduction
- Interface Path et Classe Paths
- Classe Files
- Classe File
- Classes Reader/Writer
- Classes Stream

Introduction

Introduction

La gestion des flux d'entrées/sorties en Java est gérée par les classes du packages java.io et java.nio.

Ces classes vont permettre de parcourir le système de fichiers, de lire des fichiers et d'en écrire.

Le package java.io comprend les classes et interfaces :

- File
- Reader/Writer
- Stream

Introduction

Le package java.nio (nio = New I/O) est apparu dans **Java 1.4**.
Il propose une API plus moderne et plus complète pour gérer l'accès aux répertoires et fichiers avec notamment l'apparition des **channels**.

Introduction

Le package java.nio a été considérablement enrichi dans **Java 7**.
Il est composé des classes et interfaces principales qui sont regroupées dans le package java.nio.file :

- Path (interface) et Paths : encapsule un chemin dans le système de fichiers
- Files : permet de manipuler les éléments du système de fichiers
- FileSystemProvider : interagit avec le système de fichiers
- FileSystem : encapsule un système de fichiers
- FileSystems : fabrique une instance de FileSystem

Classe Paths
Interface Path

Classe Paths

La classe java.io.file.Paths permet de récupérer un Path à partir d'un chemin d'un répertoire ou d'un fichier défini par une String ou une URI (=Uniform Resource Identifier)

Exemple pour avoir le chemin d'un fichier :

```
Path path1 = Paths.get("C:\\Users\\odenier\\Documents\\ESGI\\monfichier.txt");
Path path2 = Paths.get(URI.create("file:///C:/Users/odenier/Documents/ESGI/monfichier.txt"));
```

Interface Path

L'interface java.io.file.Path permet de représenter le chemin d'un répertoire ou d'un fichier.

Comme Path est une interface, il faut récupérer une classe de type Path pour pouvoir l'utiliser concrètement.

Exemple pour avoir le chemin du fichier "logs/access.log" :

```
Path path = FileSystems.getDefault().getPath("logs", "access.log");
```

Exemple pour avoir le chemin d'un File :

```
File file = new File("monFichier.txt");
Path myPath = file.toPath();
```

Interface Path

Voici quelques méthodes courantes de Path :

- compareTo()
- equals()
- getFileName()
- getParent() et getRoot()
- isAbsolute()
- startsWith(..) et endsWith(..)
- toAbsolutePath()
- toFile()
- toString()
- toUri()

Classe Files

Classe Files

La classe java.io.file.Files permet de réaliser des traitements sur les fichiers ou les répertoires définis par un Path

Exemple pour avoir des infos sur un fichier :

```
Path myFilePath = Paths.get("C:\\Users\\odenier\\Documents\\ESGI\\monfichier.txt");
boolean isReadable = Files.isReadable(myFilePath);
```

Classe Files

Voici quelques **méthodes statiques** de Files :

- copy(..)
- createDirectory(..), createFile(..), createLink(..), createSymbolicLink(..), createTempFile(..), createTempDirectory(..), …
- delete(..), deleteIfExists(..)
- exists(..)
- getAttribute(..), getLastModifiedTime(..), getOwner(..), …
- isReadable(Path), isWritable(Path), isHidden(Path), isExecutable(Path), isRegularFile(Path), isDirectory(Path), isSymbolicLink(Path), …
- move(..)
- readAllBytes(Path), readAllLines(..), …
- setAttribute(..), setLastModifiedTime(..), setOwner(..), …
- size(Path)
- write(..)

Classe File

Classe File

La classe java.io.File permet de gérer les répertoires et fichiers.

Le séparateur de répertoires est différent suivant les OS :

- “/” sur Unix, Linux, MacOS
- “\” sur Windows

En Java, il est conseillé d’utiliser File.separator lorsqu’on construit un nom complet de fichier.

Exemple :

```
String nomComplet = “Mes documents” + File.separator + “readme.txt”;
String nomComplet = “Mes documents” + System.getProperty(“file.separator”) + “readme.txt”;
```

Classe File

Voici quelques méthodes courantes de File :

- boolean canExecute(), canRead() et canWrite()
- delete()
- exists()
- getAbsoluteFile() et getAbsolutePath()
- getName() et getPath()
- isDirectory(), isFile(), isHidden()
- lastModified()
- length()
- list() et listFiles()
- mkdir() et mkdirs()
- renameTo(File dest)
- setExecutable(), setLastModified(long time), setReadable(..), setReadOnly(), setWritable(..)

TD07.01-File

Exercice : Afficher la liste des fichiers d'un répertoire

1. Dans votre IDE, créez un nouveau projet Java nommé par exemple "MyFileLister"
2. Définissez un nom de package
3. Créez une classe et y ajouter la méthode public static void main(String[ ] args) mais laissez-la vide
4. Créez une autre classe nommée MyDir qui aura :
    a. Un constructeur pour définir un répertoire d'entrée
    b. Une méthode pour lister et afficher
        i. les fichiers du répertoire avec leur nom et taille,
        ii. les répertoires avec leur nom
5. Dans la méthode main, instanciez un objet MyDir pour afficher les fichiers du répertoire spécifié

Classes “Reader/Writer”

Classes “Reader/Writer”

Les classes “Reader/Writer” permettent de lire/écrire dans des fichiers textes.

Les classes Reader et Writer sont des classes abstraites.
Ce sont leurs sous-classes concrètes respectives qui pourront être instanciées.

Les classes “Reader”

Les classes “Reader” sont définies par plusieurs classes Java :

- Reader
  - BufferedReader
    - LineNumberReader
  - CharArrayReader
  - FilterReader
    - PushbackReader
  - InputStreamReader
    - FileReader
  - PipedReader
  - StringReader

Méthodes de Reader

Voici quelques méthodes courantes de Reader :

- close()
- read(....)
- ready()
- reset()

Les classes “Writer”

Les classes “Writer” sont définies par plusieurs classes Java :

- Writer
  - BufferedWriter
  - CharArrayWriter
  - FilterWriter
  - OutputStreamWriter
    - FileWriter
  - PipedWriter
  - PrintWriter
  - StringWriter

Méthodes de Writer

Voici quelques méthodes courantes de Writer :

- append(....)
- close()
- flush()
- write(....)

TD07.02-Reader/Writer

Exercice : Ecrire un fichier texte et le lire

1. Dans votre IDE, créez un nouveau projet Java nommé par exemple “TextStorage”
2. Définissez un nom de package
3. Créez une classe et y ajouter la méthode public static void main(String[ ] args) mais laissez-la vide
4. Créez une autre classe nommée Storage qui va permettre de :
   a. Définir un nom de fichier
   b. Définir une méthode read() de lecture de données du fichier
   c. Définir une méthode write() d’écriture de données dans le fichier
5. Dans la méthode main, instanciez un objet Storage pour :
   a. Ecrire du texte dans un fichier
   b. Lire le texte dans un fichier et l’afficher dans la console

Classes Stream

Classes “Stream”

Les classes “Stream” permettent de lire/écrire des flux binaires.
Les informations sont donc véhiculées sous forme d’octets.

La classe abstraite InputStream et ses sous-classes permettent de lire des flux binaires.
La classe abstraite OutputStream et ses sous-classes permettent d’écrire des flux binaires.

Les classes “InputStream”

Les classes “InputStream” sont définies par plusieurs classes Java :

- InputStream
  - AudioInputStream
  - ByteArrayInputStream
  - FileInputStream
    - BufferedInputStream, CheckedInputStream, CipherInputStream, DataInputStream, DeflaterInputStream, DigestInputStream, InflaterInputStream, LineNumberInputStream, ProgressMonitorInputStream, PushbackInputStream
  - FilterInputStream
  - InputStream
  - ObjectInputStream
  - PipedInputStream
  - SequenceInputStream
  - StringBufferInputStream

Méthodes de InputStream

Voici quelques méthodes courantes de InputStream :

- available()
- close()
- read(....)
- reset()

Les classes “OutputStream”

Les classes “OutputStream” sont définies par plusieurs classes Java :

- OutputStream
  - ByteArrayOutputStream
  - FileOutputStream
  - FilterOutputStream
    - BufferedOutputStream, CheckedOutputStream, CipherOutputStream, DataOutputStream, DeflaterOutputStream, DigestOutputStream, InflaterOutputStream, PrintStream
  - ObjectOutputStream
  - OutputStream
  - PipedOutputStream

Méthodes de OutputStream

Voici quelques méthodes courantes de OutputStream :

- close()
- flush()
- write(....)

TD07.03-InputStream/OutputStream

Exercice : Copier un fichier binaire

1. Dans votre IDE, créez un nouveau projet Java nommé par exemple “Copier”
2. Définissez un nom de package
3. Créez une classe et y ajouter la méthode public static void main(String[ ] args) mais laissez-la vide
4. Créez une autre classe nommée FileCopy qui va permettre de :
   a. Définir un nom de fichier à lire, un nom de fichier à écrire
   b. Définir une méthode copy() qui va lire le fichier de lecture et écrire les données dans le fichier d’écriture
5. Dans la méthode main, instanciez l’objet FileCopy afin de dupliquer un fichier.

Bonnes pratiques

A retenir

- **Privilégier l'utilisation des API java.nio à java.io**
- Pour définir des chemins de répertoire/fichier, utiliser Path
- Ne pas utiliser de chemin en dur avec des "/" ou "\"
  → MAIS utiliser File.separator
- Utiliser le type de classe IO correspondant à vos données
  → Reader/Writer pour des flux textes
  → InputStream/OutputStream pour des flux binaires
- Penser à toujours fermer vos flux
  → Utiliser pour cela le bloc finally
  → Utiliser le **try-with-resources**
- Attention à l'encoding de vos flux
  → Privilégier UTF-8

Questions/Réponses