JAVA
FONDAMENTAUX

Présenté par :

Xavier TABUTEAU

Les entrées / sorties de fichiers

La gestion des fichiers se fait par l'API « IO » situé dans le package « java.io.* » qui existe depuis Java 1.0 (1995). Cette API permet les accès disques et réseaux (web / BDD) et mémoire. A partir de Java 4 (2002) une nouvelle version de « IO » est sortie et est nommée « NIO » pour « New IO ». En Java 7 (2011) une extension de « NIO » sort et s'appel « NIO2 ». Ce cours présente l'API « IO ».

IO utilise la classe « File » et l'interface « Path » (« Path » est arrivé avec « NIO2 » mais fait parti du package de l'API « IO »). Ils permettent de modéliser des chemins vers le « FileSystem ». L'interface permet d'être implémentée par des classes ayant les mêmes fonctionnalités en fonction du système d'exploitation. Les attributs de sécurité peuvent être différents selon le système d'exploitation et donc en fonction du « FileSystem ».

Création d'une instance de File :

File f = new File(« chemin/fichier »);

Le chemin dans File peut être relatif ou complet. Quelque soit le système d'exploitation, on utilise le « / » pour séparer les dossiers et fichiers. Java adaptera le « / » en « \ » si besoin.
Si on veut vraiment mettre le chemin avec des antislash alors il faut les doubler « dossier\\fichier » comme pour d'autre caractères spéciaux tel que « \n »ou « \t ».

Les entrées / sorties de fichiers

Pour un créer un Path du fait que c'est une interface et pas une classe. On n'utilise pas « new » mais une méthode statique fournie.

En Java 11 :
Path p = Path.of(« chemin/fichier »);

Entre Java 7 et 10 on utilise une méthode Factory appelée « Paths » :
Path p = Paths.of(« chemin/fichier »);

« File » et « Path » ne sont que des objets en mémoire modélisant un chemin. Il ne font pas d'accès disque, ne créer pas de fichier et ne vérifie pas si le chemin existe.

Pour obtenir des informations sur le chemin passé dans « File » on peut utiliser plusieurs méthodes.

getName() : Donne le nom du fichier.
getParent() : Donne le dom du dossier parent.
getPath() : Donne le chemin et le nom du fichier complet.

Avec ces trois méthodes, aucune vérification n'est fait sur le chemin réel du disque. On se base que sur la chaine de caractères passée en paramètre de « File ».

Les entrées / sorties de fichiers

Les méthodes suivantes font un accès au disque avec un « FileSystem ».
Une exception « IOException » est générée en cas d'erreur.

getCanonicalPath()	: Essai de donner le nom du chemin complet sur le disque.
inFile()	: Retourne true si le fichier existe.
inDirectory()	: Retourne true si le répertoire existe.
canRead()	: Retourne true si la lecture du fichier est possible.
canWrite()	: Retourne true si l'écriture du fichier est possible.
canExectue	: Retourne true si le fichier peut être exécuté.
exists()	: Retourne true si le chemin du fichier existe.
createNewFile()	: Si le fichier n'existe pas, il est créé.
mkdir()	: Créer le dossier final du chemin indiqué.
mkdirs()	: Créer le dossier principal et tout les sous dossiers du chemin indiqué.
delete()	: Supprime le fichier.
renameTo()	: Renomme un fichier.

Les entrées / sorties de fichiers

L’API « IO » divise les flux en deux types. Les textes (chaine de caractères) et les binaires (octets). Ces deux types de flux sont subdivisés en deux catégories. La lecture et l’écriture.

	Texte	Binaire
Lecture	Reader	InputStream
Ecriture	Writer	OutputStream

Ces quatre classes sont abstraites. Le JDK va étendre ces classes. Ces classes définissent la façon dont on peut lire ou écrire du texte ou des données binaires mais elles ne définissent pas le medium de sortie (fichier, socket ou buffer en mémoire).

Ecriture de texte (Writer)

• La classe abstraite « Writer » possède les méthodes :

write() : Permet d'écrire du texte. Elle ne retourne rien.
append() : Ajoute du texte.
flush() : Vide le contenu restant dans le « Writer » vers sa destination (fichier, etc …). En cas de try with ressource, flush() est exécutée à la fermeture.
close() : Ferme la ressource.

Pour écrire ou ajouter un texte dans un fichier, on utilise la classe concrète « FileWriter » qui définie les méthodes abstraites de « Writer » pour être utiliser avec un fichier. Le deuxième argument de « FileWriter » est un Boolean (true pour ajouter, false par défaut pour créer ou écraser).

• « BufferedWriter » est une classe fille de « Writer » prenant comme paramètre une des autres classes filles de « Writer » telle que « FileWriter ». C'est un décorateur de « Writer ». Elle possède une méthode supplémentaire intéressante :

newLine() : Ecrit un saut de ligne à la fin du write() qui précède.

• « PrintWriter » est un autre décorateur qui donne accès à trois méthodes :
print() : Ecrit un texte sans retour chariot.
println() : Ecrit un texte avec retour chariot.
printf() : Ecrit une ligne formatée comme en C/C++ (voir la documentation).

FichiersIO.java

Lecture de texte (Reader)

• La classe abstraite « Reader » possède les méthodes :

read() : Lit un caractère.
skip(x) : Saute x caractères.
close() : Ferme le flux de lecture de fichier texte.

Pour lire du texte dans un fichier, on utilise la classe fille concrète « FileReader » qui définie les méthodes abstraites de « Reader » pour être utilisable avec un fichier.

• « BufferedReader » est un décorateur de « Reader » ajoutant les méthodes suivantes :

readLine() : Retourne une ligne de texte sous forme de String.
lines() : Retourne toutes les lignes sous forme de Stream<String>.

Elle permet de simplifier le code de lecture d'un texte.

• Il est possible d'utiliser la classe « LineNumberReader » qui prend en paramètre un « BufferedReader » et qui possède des méthodes pour compter les ligne lues, ou changer la valeur du compteur de lignes lues.

Ecriture binaire (OutputStream)

• La classe abstraite « OutputStream » possède les méthodes suivantes :

write() : Permet d'écrire des octets. Elle ne retourne rien.
flush() : Vide le contenu restant dans le « OutputStream » vers sa destination (fichier, etc…). En cas de try with ressource, flush() est exécutée à la fermeture.
close() : Ferme la ressource.

Pour écrire des octets dans un fichier, on utilise la classe concrète « FileOutputStream » qui définie les méthodes abstraites de « OutputStream ». Le deuxième argument de « FileOutputStream » est un Boolean (true pour ajouter, false par défaut pour créer ou écraser).

• « BufferedOutputStream » est décorateur de « OutputStream ». L'utilisation de cette classe au lieu d'utiliser directement « FileOutputStream » améliore les performances.

• « DataOutputStream » est un autre décorateur permettant l'écriture des types primitifs Java en octets. Les méthodes sont writeInt() etc… sauf writeUTF() pour le type String.

Remarque : Si on écrire avec « DataOutputStream » et « BufferedOutputStream » dans la même destination. En lecture il faudra lire avec « DataInputStream » et « BufferedInputStream » dans le même ordre et les mêmes types.

FichiersIO.java

Ecriture binaire (OutputStream)

- « ObjectOutputStream » est un autre décorateur de « OutputStream » permettant l'utilisation d'une méthode supplémentaire s'appelant writeObject() et qui permet d'écrit la totalité de l'objet dans la destination. L'objet doit être sérialisable, c'est-à-dire que la classe de l'objet doit implémenter l'interface « Serializable ». Si ce n'est pas le cas, la méthode writeObject() générera une exception.

Lecture binaire (InputStream)

• La classe abstraite « InputStream » possède les méthodes :

read() : Lit un octet. Retourne un « int ».
readAll() : Lit un tableau d'octets.
mark(x) : Pose un index à la valeur x.
reset() : Permet de revenir à la position de l'index posé avec mark().
close() : Ferme le flux de lecture de fichier texte.

Pour lire des octets dans un fichier, on utilise la classe fille concrète « FileInputStream » qui définie les méthodes abstraites de « InputStream ».

• « BufferedInputStream » est un décorateur de « InputStream ». L'utilisation de cette classe au lieu d'utiliser directement « FileInputStream » améliore les performances.

• « DataInputStream » est un autre décorateur de « InputStream » ajoutant la fonctionnalité de lecture des octets des types primitifs Java. Les méthodes sont readInt() etc… sauf readUTF() pour le type String.

Flux mixtes (InputStreamReader / OutputStreamWriter)

- « InputStreamReader » permet de lire des caractères sur un flux d'octets.
- « OutputStreamWriter » permet d'écrire des caractères sur un flux d'octets.

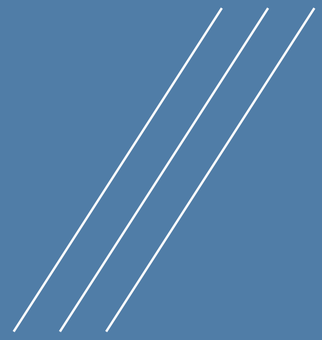
Remarque : Le format JPEG utilise un format mixte.

Ils s'utilisent et se décorent de la même façon que les autres flux.

Il existe aussi des classes pour faire des flux compressés.

- GzipInputStream / GzipOutputStream
- ZipInputStream / ZipOutputStream

JAVA
FONDAMENTAUX



Présenté par
Xavier TABUTEAU