Introduction à la programmation et à la modélisation en Java

Flux et Fichiers

Partie 1 : Notion de flux

C'est une notion très générale.

- Un flux de sortie (comme dans « System.out ») désigne n'importe que système susceptible de recevoir de l'information sous forme d'une suite d'octets (périphérique d'affichage, fichier, connexion à un site distant, emplacement en mémoire centrale)
- Un flux d'entrée délivre de l'information sous forme d'une suite d'octets (clavier, fichier, connexion, emplacement en mémoire centrale)
- Flux binaire VS texte : un flux binaire correspond à de l'information brute, non transformée. Dans un flux de texte, les octets sont formatés sous la forme de caractère (comme dans « System.out.print »).
- Attention, la classe Stream introduite en Java 8, n'est pas liée aux flux de ce cours.

Accès aux fichiers : Séquentiel VS Direct

- Séquentiel : on traite l'information dans l'ordre
- Direct : on se place directement sur l'information voulue

Partie 2 : Quelques classes de flux

I. Flux binaires

A. Sortie - Ecriture

- OutputStream : classe abstraite commune à tous les flux binaires de sortie
- FileOutputStream : héritée pour l'écriture de fichiers binaires
- DataOutputStream : flux « haut niveau » pour écrire des données (par exemple dans un fichier)
- BufferedOutputStream : tampon entre le flux de fichier et le flux de données pour optimiser les écritures.

B. Entrée - Lecture

- InputStream : classe abstraite commune à tous les flux binaires d'entrée
- FileInputStream : héritée pour la lecture de fichiers binaires
- DataInputStream : flux « haut niveau » pour lire des données binaires (pas forcément depuis un fichier)
- BufferedInputStream : tampon entre le flux de fichier et le flux de données pour optimiser la lecture.

Remarques :

- Ne pas oublier de fermer le flux de données à la fin !
- Attention : flux sur les types primitifs uniquement !

II. Flux textes

A. Ecriture

- Writer : classe abstraite commune à tous les flux textes de sortie.
- OutputStreamWriter : héritée et concrète.
- FileWriter : héritée de OutputStreamWriter pour les fichiers.
- PrinterWriter : formate le flux texte (via la localisation de l'OS)
- BufferedWriter : introduit un tampon entre le flux-fichier et le flux de texte.

B. Lecture

- Reader : classe abstraite commune à tous les flux textes d'entrée
- FileReader : héritée, pour les fichiers (permet juste de lire des caractères)
- BufferedReader : tampon (en plus permet de lire des lignes)
- Pas de formatage, pas de « PrinterReader »

Partie 3 : La classe File

Au lieu de créer un flux directement avec un nom de fichier, il est possible d'utiliser la classe « File » :

- File f = new File(« donnees.dat ») ;
- FileInputStream flux = new FileInputStream(f) ;

Un objet de la classe « File » n'est pas un fichier sur le disque, c'est un objet en mémoire permettant la manipulation de fichiers ou de répertoires.

Cette classe permet de gérer les chemins (File.separator permet de résoudre l'incompatibilité entre Windows et Linux!), les métadonnées (droits, dates, etc), l'espace disque, etc.

JDK7 : une nouvelle API d'acronyme NIO.2 apporte des fonctionnalités supplémentaires, en particulier via la classe « Path », remplaçante de la classe « File ». Cette classe dépasse le cadre de ce cours.

Partie 4 : Exemples

1. Flux séquentiel binaire de sortie

Exemple : FluxSequentielSortie01.java
Visualisez le contenu avec un « cat » … c'est un flux binaire !

2. Flux séquentiel binaire d'entrée

Exemple : FluxSequentielEntree02.java

3. Écriture d'un fichier texte

Exemple : FluxTexteSortie03.java

4. Lecture d'un fichier texte

Exemples :

1. « ligne à ligne » : FluxTexteEntree04.java
2. « lexème par lexème » (token) : FluxTexteEntree05.java (utilisation de la classe StringTokenizer)