INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE

Java

benjamin.nguyen@insa-cvl.fr

d’après J.-M. Doudoux

(Flux) et Fichiers

De java.io à java.nio

But du cours (compétences)

- Savoir coder une application complexe en Java (avec des tests unitaires)
- Savoir débugger une application complexe en Java
- Savoir gérer les fichiers en Java
- Savoir gérer la persistence en Java avec la sérialisation
- Savoir utiliser les threads en Java
- Savoir créer une interface graphique simple en Java (approche modèle MVC)

Si le temps le permet …

- Connaître les bibliothèques cryptographiques en Java
- Comprendre et avoir utilisé le mécanisme d'introspection en Java
- Savoir gérer la persistence en Java avec une base de données

java.io

depuis java 1.1 (1997)

Les fichiers : `File`

- Définis dans la bibliothèque java.io

Méthode	**Rôle**
boolean canRead()	Indiquer si le fichier peut être lu
boolean canWrite()	Indiquer si le fichier peut être modifié
boolean createNewFile()	Créer un nouveau fichier vide
File createTempFile(String, String)	Créer un nouveau fichier dans le répertoire par défaut des fichiers temporaires. Les deux arguments sont le nom et le suffixe du fichier
File createTempFile(String, String, File)	Créer un nouveau fichier temporaire. Les trois arguments sont le nom, le suffixe du fichier et le répertoire
boolean delete()	Détruire le fichier ou le répertoire. Le booléen indique le succès de l'opération
deleteOnExit()	Demander la suppression du fichier à l'arrêt de la JVM
boolean exists()	Indique si le fichier existe physiquement
String getAbsolutePath()	Renvoyer le chemin absolu du fichier
String getPath	Renvoyer le chemin du fichier
boolean isAbsolute()	Indiquer si le chemin est absolu
boolean isDirectory()	Indiquer si le fichier est un répertoire
boolean isFile()	Indiquer si l'objet représente un fichier
long length()	Renvoyer la longueur du fichier
String[] list()	Renvoyer la liste des fichiers et répertoires contenus dans le répertoire
boolean mkdir()	Créer le répertoire
boolean mkdirs()	Créer le répertoire avec création des répertoires manquants dans l'arborescence du chemin
boolean renameTo()	Renommer le fichier

Petits tests introductifs

- Ces méthodes permettent de naviguer dans la hiérarchie, obtenir des informations au sujet des fichiers ou répertoires, et effectuer les opérations “de base” d’un système d’exploitation.

- Exemple d’affichage du contenu d’un répertoire, ou des informations d’un fichier : `MainFichiers`

Quelques exemple de classes :

- `FileReader`
- `BufferedInputStream`
- `RandomAccessFile`

- Pourquoi tant de classes ?
- A quoi servent-elles ?
- Comment choisir la bonne classe ?

Caractéristiques

- Choisir le sens du flux (entrée ou sortie)
- Choisir le type d’entrée ou de sortie
- Choisir un traitement à effectuer sur le flux (filtre)

Des flux et des fichiers

- Les programmes doivent gérer des entrées et des sorties, en particulier : Fichiers (mais aussi sockets réseau, clavier, écran, etc)
- 2 grands types de flux
  - Entrée
  - Sortie
- 2 grands types de données
  - traitement d'octets (binaire)
  - traitement de caractères

SUFFIXE DE LA CLASSE

	Flux d'octets	Flux de caractères
Flux d'entrée	InputStream	Reader
Flux de sortie	OutputStream	Writer

- On s'intéresse dans ce cours à des entrées et sorties de *fichiers* uniquement.

Ces (nombreuses) classes sont présentes dans la bibliothèque `java.io`

Préfixe de la classe :

- Pour les flux : contient la source ou la destination, selon le sens du flux : e.g. `FileReader`

Préfixe du flux	**source ou destination du flux**
ByteArray	tableau d'octets en mémoire
CharArray	tableau de caractères en mémoire
File	fichier
Object	objet
Pipe	pipeline entre deux threads
String	chaîne de caractères

Préfixe de la classe :

- Pour les filtres : contient le type d'opération effectué

Type de traitement	Préfixe de la classe	En entrée	En sortie
Mise en tampon	Buffered	Oui	Oui
Concaténation de flux	Sequence	Oui pour flux d'octets	Non
Conversion de données	Data	Oui pour flux d'octets	Oui pour flux d'octets
Numérotation des lignes	LineNumber	Oui pour les flux de caractères	Non
Lecture avec remise dans le flux des données	PushBack	Oui	Non
Impression	Print	Non	Oui
Sérialisation	Object	Oui pour flux d'octets	Oui pour flux d'octets

Exemples de classes

	Flux en lecture	**Flux en sortie**
Flux de caractères	BufferedReader CharArrayReader FileReader InputStreamReader LineNumberReader PipedReader PushbackReader StringReader	BufferedWriter CharArrayWriter FileWriter OutputStreamWriter PipedWriter StringWriter
Flux d'octets	BufferedInputStream ByteArrayInputStream DataInputStream FileInputStream ObjectInputStream PipedInputStream PushbackInputStream SequenceInputStream	BufferedOutputStream ByteArrayOutputStream DataOuputStream FileOutputStream ObjetOutputStream PipedOutputStream PrintStream

Flux de caractères

- Gestion de l'encodage ? Par défaut sur 16 bits (de 0 à 65535)
- le char en java utilise un codage de longueur variable sur 16 bits

```
PS C:\tmp> Format-Hex .\source-utf8.txt

           Path: C:\tmp\source-utf8.txt

           00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   41 42 43 44 C3 88 46 47 48 49 4A 4B              ABCDÃ▯FGHIJK
```

Le fichier `source-utf8.txt` qu'on va utiliser

Exemple de lecture de fichier (`MainExample`)

- Approche caractère `TestFileReader`
- Approche binaire `TestFileInputStream`

- Utilisation de la méthode `read()` qui retourne -1 lorsqu'on est arrivé à la fin du fichier.

Autres méthodes de la classe `Reader`

Méthodes	**Rôles**
boolean markSupported()	indique si le flux supporte la possibilité de marquer des positions
boolean ready()	indique si le flux est prêt à être lu
close()	ferme le flux et libère les ressources qui lui étaient associées
int read()	renvoie le caractère lu ou -1 si la fin du flux est atteinte.
int read(char[])	lire plusieurs caractères et les mettre dans un tableau de caractères
int read(char[], int, int)	lire plusieurs caractères. Elle attend en paramètre : un tableau de caractères qui contiendra les caractères lus, l'indice du premier élément du tableau qui recevra le premier caractère et le nombre de caractères à lire. Elle renvoie le nombre de caractères lus ou -1 si aucun caractère n'a été lu. Le tableau de caractères contient les caractères lus.
long skip(long)	saute autant de caractères dans le flux que la valeur fournie en paramètre. Elle renvoie le nombre de caractères sautés.
mark()	permet de marquer une position dans le flux
reset()	retourne dans le flux à la dernière position marquée

Note : la création d'un objet `Reader` ouvre le flux (et lance une exception en cas de problème)

Méthodes de la classe `Writer`

Méthodes	**Rôles**
close()	ferme le flux et libère les ressources qui lui étaient associées
write(int)	écrire le caractère en paramètre dans le flux. (note : mieux que unicode 16 bits)
write(char[])	écrire le tableau de caractères en paramètre dans le flux.
write(char[], int, int)	écrire plusieurs caractères. Elle attend en paramètres : un tableau de caractères, l'indice du premier caractère et le nombre de caractères à écrire.
write(String)	écrire la chaîne de caractères en paramètre dans le flux
write(String, int, int)	écrire une portion d'une chaîne de caractères. Elle attend en paramètre : une chaîne de caractères, l'indice du premier caractère et le nombre de caractères à écrire.

Note : la création d'un objet `Writer` ouvre le flux (et lance une exception en cas de problème)

Exemple d’écriture de fichier

- **Classes :** `TestFileWriter` **et** `TestPrintWriter`

Utilisation de classes `BufferedReader` et `BufferedWriter`

- Intérêt
  - en lecture : possibilité de “lire à l’avance” (sur le disque) dans un buffer de taille contrôlable (RAM) qui permet d’améliorer la vitesse de lecture. Note : de base le `Reader` utilise un buffer de 8K.
  - en écriture : utilisation d’un buffer (mémoire) pour écriture (disque) uniquement lorsqu’on le demande (`flush` ou `close`) cf cours de MIBD
- En “surcouche” sur un `Reader` déjà existant (pas forcément sur un fichier).

Constructeur	**Rôle**
BufferedReader(Reader)	le paramètre fourni doit correspondre au flux à lire.
BufferedReader(Reader, int)	l'entier en paramètre permet de préciser la taille du buffer. Il doit être positif sinon une exception de type IllegalArgumentException est levée.
BufferedWriter(Writer)	le paramètre fourni doit correspondre au flux dans lequel les données sont écrites.
BufferedWriter(Writer, int)	l'entier en paramètre permet de préciser la taille du buffer. Il doit être positif sinon une exception IllegalArgumentException est levée.

BufferedReader et BufferedWriter

- Ne rajoute que peu de méthodes :

Méthode	**Rôle**
String readLine()	lire une ligne de caractères dans le flux. Une ligne est une suite de caractères qui se termine par un retour chariot '\r' ou un saut de ligne '\n' ou les deux.

Méthode	**Rôle**
flush()	vide le tampon en écrivant les données dans le flux.
newLine()	écrire un séparateur de ligne dans le flux
write(String)	écrire la String dans le flux (méthode héritée de `Writer`)

Flux binaires (octets) : `InputStream`

Méthode	**Rôle**
long skip(long)	saute autant d'octets dans le flux que la valeur fournie en paramètre. Elle renvoie le nombre d'octets sautés.
close()	ferme le flux et libère les ressources qui lui étaient associées
int available()	retourne une estimation du nombre d'octets qu'il est encore possible de lire dans le flux
int read()	Cette méthode envoie la valeur de l'octet lu ou -1 si la fin du flux est atteinte.

Flux binaires (octets) : `OutputStream`

Méthode	**Rôle**
write(int)	Cette méthode écrit l'octet en paramètre dans le flux
write(byte[])	Cette méthode écrit plusieurs octets. Elle attend en paramètre : un tableau d'octets qui contient les octets à écrire : tous les éléments du tableau sont écrits
write(byte[], int, int)	Cette méthode écrit plusieurs octets. Elle attend en paramètre : un tableau d'octets qui contient les octets à écrire, l'indice du premier élément du tableau d'octets à écrire et le nombre d'octets à écrire

Fichiers à accès direct : `RandomAccessFile`

- Accès direct à un enregistrement du fichier via la méthode `seek`()
- Idéalement : tous les enregistrements ont la même taille
- Constructeur : prend un paramètre String qui contien r, w ou rw indiquant le type d'accès autorisé sur le fichier.

Méthode	**Rôle**
seek(long pos)	Se positionne à l'offset pos dans le fichier (en octets)
long getFilePointer()	Retourne la position du pointeur de lecture du fichier
int readInt()	Lit un entier à la position du pointeur
void writeInt()	Ecrit un entier à la position du pointeur
String readUTF()	Permet de lire une chaîne UTF-8
void writeUTF()	Ecrit une chaine en UTF-8. Les 2 premier bytes sont codés en unsigned short et donnent la longueur de la chaîne (voir exemple)

java.nio

Conseillé depuis Java 7 (2011)

Intérêts de java.nio

- Simplification de la gestion d'un système de fichiers
- Lecture et écriture asynchrones (performances)

CARACTERISTIQUES

- **Séparation des responsabilités :** un chemin (Path) représente un élément du système de fichiers (FileSystem) stocké dans un système de stockage (FileStorage) et est manipulé en utilisant la classe Files
- Gestion de toutes les erreurs avec des **exceptions**
- **Utilisation de fabriques** pour créer les différentes instances de l'API et de la rendre extensible

Quelques ajouts pratiques

- le support des liens physiques et symboliques s'ils sont pris en charge par le système de fichiers
- la gestion des attributs sur les fichiers des systèmes Dos et POSIX (Portable Operating System)
- Le support de notifications en cas de changement dans le contenu d'un répertoire (ajout, suppression, modification d'un fichier du répertoire) en utilisant l'API WatchService
- le support du parcours d'un répertoire avec la possibilité de filtrer les fichiers obtenus
- l'utilisation de channels asynchrones avec lesquels les opérations de lecture/écriture sont réalisées en utilisant un pool de threads
- l'ajout de fonctionnalités de base comme la copie ou le déplacement de fichiers
- l'utilisation de fabriques pour permettre à l'API d'être extensible : il est par exemple possible de créer sa propre implémentation d'un système de fichiers. Une implémentation permettant de gérer les fichiers zip est d'ailleurs fournie en standard.

Principales classes/interfaces

- Path : encapsule un chemin dans le système de fichiers
- Files : contient des méthodes statiques pour manipuler les éléments du système de fichiers
- FileSystemProvider : service provider qui interagit avec le système de fichiers sous-jacent
- FileSystem : encapsule un système de fichiers
- FileSystems : fabrique qui permet de créer une instance de FileSystem

Path

Peut être :

- Un fichier
- Un répertoire
- Un lien symbolique : permet de faire référence à un fichier ou un autre répertoire
- Un sous-chemin (chemin absolu ou chemin relatif calculé par rapport au chemin courant)

Obtenu via :

- la méthode getPath() d'une instance de type FileSystem
- la méthode Paths.get() qui invoque la méthode FileSystems.getDefault().getPath()
- la méthode toPath() sur un objet de type java.io.File

On ne vérifie pas qu’il existe sur le système de fichiers ! (Files va faire ça)

Path

Méthode	Rôle
String getFileName()	Retourner le nom du dernier élément du chemin. Si le chemin concerne un fichier alors c'est le nom du fichier qui est retourné
Path getName(int index)	Retourner l'élément du chemin dont l'index est fourni en paramètre. Le premier élément possède l'index 0
int getNameCount()	Retourner le nombre d'éléments du chemin
Path getParent()	Retourner le chemin parent ou null s'il n'existe pas (dans ce cas, le chemin correspond à une racine)
Path getRoot()	Retourner la racine d'un chemin absolu (par exemple C:\ sous Dos ou / sous Unix) ou null pour un chemin relatif
String toString()	Retourner le chemin sous la forme d'une chaîne de caractères
Path subPath(int beginIndex, int endIndex)	Retourner un sous-chemin correspondant aux deux index fournis en paramètres

Fichier ExemplePath.java

Path

Méthode	Rôle
Path normalize()	Nettoyer le chemin en supprimant les éléments « . » et « .. » qu'il contient
Path relativize(Path other)	Retourner le chemin relatif à celui fourni en paramètres
Path resolve(Path)	Combiner deux chemins

Path : comparaison de chemins

Méthode	Rôle
int compareTo(Path other)	Comparer le chemin avec celui fourni en paramètre
boolean endsWith(Path other)	Comparer la fin du chemin avec celui fourni en paramètre
boolean endsWith(String other)	Comparer la fin du chemin avec celui fourni en paramètre
boolean startsWith(Path other)	Comparer le début du chemin avec celui fourni en paramètre
boolean startsWith(String other)	Comparer le début du chemin avec celui fourni en paramètre

Recherche de fichiers : le GLOB (global command)

- Expression régulière simple

Motif	Rôle
*	Aucun ou plusieurs caractères
**	Aucun ou plusieurs sous-répertoires
?	Un caractère quelconque
{}	Un ensemble de motifs exemple : {htm, html}
[]	Un ensemble de caractères. Exemple : [A-Z] : toutes les lettres majuscules [0-9] : tous les chiffres [a-z,A-Z] : toutes les lettres indépendamment de la casse Chaque élément de l'ensemble est séparé par un caractère virgule Le caractère - permet de définir une plage de caractères A l'intérieur des crochets, les caractères *, ? et / ne sont pas interprétés
\	Il permet d'échapper des caractères pour éviter qu'ils ne soient interprétés. Il sert notamment à échapper le caractère \ lui-même
Les autres caractères	Ils se représentent eux-mêmes sans être interprétés

Files

Méthode	Rôle
boolean exists(Path)	vérifier l'existence sur le système de fichiers de l'élément dont le chemin est encapsulé dans le paramètre de type Path fourni
boolean notExists(Path)	vérifier que l'élément dont le chemin est encapsulé dans l'instance de type Path fournie en paramètre n'existe pas sur le système de fichiers
boolean isReadable(Path path)	Retourner true si le fichier peut être lu
boolean isWritable(Path path)	Retourner true si le fichier peut être modifié
boolean isHidden(Path path)	Retourner true si le fichier est caché
boolean isExecutable(Path path)	Retourner true si le fichier est exécutable
boolean isRegularFile(Path path)	Retourner true si l'objet encapsulé dans le Path est un fichier
boolean isDirectory(Path path)	Retourner true si l'objet encapsulé dans le Path est un répertoire
boolean isSymbolicLink(Path path)	Retourner true si l'objet encapsulé dans le Path est un lien symbolique

Files : création de fichiers (y compris temporaires)

Méthode	Rôle
Path createFile(Path path, FileAttribute<?>... attrs)	Créer un fichier dont le chemin est encapsulé par l'instance de type Path fournie en paramètre
Path createDirectory(Path dir, FileAttribute<?>... attrs)	Créer un répertoire dont le chemin est encapsulé par l'instance de type Path fournie en paramètre
Path createDirectories(Path dir, FileAttribute<?>... attrs)	Créer dans le répertoire dont le chemin est fourni en paramètre un sous-répertoire avec les attributs fournis
Path createTempDirectory(Path dir, String prefix, FileAttribute<?>... attrs)	Créer dans le répertoire dont le chemin est fourni en paramètre un sous-répertoire temporaire dont le nom utilisera le préfixe fourni
Path createTempDirectory(String prefix, FileAttribute<?>... attrs)	Créer dans le répertoire temporaire par défaut du système, un sous-répertoire temporaire dont le nom utilisera la préfixe fourni
Path createTempFile(Path dir, String prefix, String suffix, FileAttribute<?>... attrs)	Créer dans le répertoire dont le chemin est fourni en paramètre un fichier temporaire dont le nom utilisera le préfixe fourni
Path createTempFile(String prefix, String suffix, FileAttribute<?>... attrs)	Créer dans le répertoire temporaire par défaut du système un fichier temporaire dont le nom utilisera le préfixe et le suffixe fournis

Files : copie, déplacement etc

- avec les méthodes
  - copy
  - move
  - delete
  - probeContentType : donne le type de fichier

Files : copie de fichier

Valeur	Rôle
StandardCopyOption.COPY_ATTRIBUTES	La copie se fait en conservant les attributs du fichier : ceux-ci sont dépendants du système sous-jacent
StandardCopyOption.REPLACE_EXISTING	Remplacer le fichier cible s'il existe. Si le chemin cible est un répertoire non vide, une exception de type FileAlreadyExistsException est levée
LinkOption.NOFOLLOW_LINKS	Ne pas suivre les liens symboliques. Si le chemin à copier est un lien symbolique, c'est le lien lui-même qui est copié

Ecriture et lecture dans des fichiers

IO	**NIO2**
Java 1.0 et 1.1	Java 7
Synchrone bloquant	Asynchrone non bloquant
File InputStream OutputStream Reader Writer Socket RandomAccessFile	Path AsynchronousFileChannel AsynchronousByteChannel AsynchronousSocketChannel AsynchronousServerSocketChannel SeekableByteChannel

Les options d'ouverture indiquent si on est en READ, WRITE, etc !

Options d’ouverture du fichier

Valeur	Rôle
APPEND	Si le fichier est ouvert en écriture alors les données sont ajoutées au fichier. Cette option doit être utilisée avec les options CREATE ou WRITE
CREATE	Créer un nouveau fichier s'il n'existe pas sinon le fichier est ouvert
CREATE_NEW	Créer un nouveau fichier : si le fichier existe déjà alors une exception est levée
DELETE_ON_CLOSE	Supprimer le fichier lorsque son flux associé est fermé : cette option est utile pour des fichiers temporaires
DSYNC	Demander l'écriture synchronisée des données dans le système de stockage sous-jacent (pas d'utilisation des tampons du système)
READ	Ouvrir le fichier en lecture
SPARSE	Indiquer au système que le fichier est clairsemé ce qui peut lui permettre de réaliser certaines optimisations si l'option est supportée par le système de fichiers (c'est notamment le cas avec NTFS)
SYNC	Demander l'écriture synchronisée des données et des métadonnées dans le système de stockage sous-jacent
TRUNCATE_EXISTING	Si le fichier existe et qu'il est ouvert en écriture alors il est vidé. Cette option doit être utilisée avec l'option WRITE
WRITE	Ouvrir le fichier en écriture

Lectures et écriture

Méthodes de lecture chaîne de caractères:

Files.readString(Path p)

Files.readAllLines(Path p)

Files.readAllBytes(Path p) pour un fichier binaire

Voir ExempleFichierNIO

Fichier RandomAccess : classe `SeekableByteChannel`

Méthode	Rôle
long position()	Retourner la position courante dans le channel
SeekableByteChannel position(long newPosition)	Changer la position dans le channel
int read(ByteBuffer dst)	Lire un ensemble d'octets du channel dans le tampon fourni en paramètre. Retourne le nombre d'octets lus ou -1 si la fin du channel est atteinte
long size()	Retourner la taille en octets du flux auquel le channel est connecté
SeekableByteChannel truncate(long size)	Tronquer le contenu de l'élément sur lequel le channel est connecté à la taille fournie en paramètre. Cela permet de redimensionner la taille du flux associé au channel avec la valeur fournie en paramètre
int write(ByteBuffer src)	Ecrire les octets fournis en paramètre à la position courante dans le channel

ExampleSeekableByteChannel

Et on n’a pas discuté du côté asynchrone ! (voir plus tard)