0-M02 – Introduction à la programmation

Cours n°6

Fichiers et flux de données

Camille Coti
camille.coti@iutv.univ-paris13.fr

IUT de Villetaneuse, département R&T

2011 – 2012

Flux

Définition

Un **flux** est une séquence d'informations.

- Il peut être en lecture ou en écriture
- Ces informations peuvent être des octets (binaire) ou des caractères

En Java :

- Un **flux d'entrée** est caractérisé par une **source** (entrée) à partir de laquelle on **lit** des informations
- Un **flux de sortie** est caractérisé par une **destination** (sortie) sur laquelle on **écrit** des informations

Exemples : fichier, socket vers une machine distante, périphérique...

Flux d'octets :

- 1 octet = 8 bits
- Représente des données binaires
- `input stream` : flux d'entrée d'octets
- `output stream` : flux de sortie d'octets

Flux de caractères :

- Caractères unicodes = 16 bits
- Représente des textes (lisibles)
- `reader` : flux d'entrée de caractères
- `writer` : flux de sortie de caractères

Utilisation d'un flux

Opérations obligatoires sur un flux :

1. Créer et ouvrir le flux
2. Lire (flux d'entrée) ou écrire (flux de sortie) les informations à partir du (ou sur le) flux
3. Fermer le flux

Classes disponibles :

Flux binaires

- FileInputStream
- FileOutputStream
- DataInputStream
- DataOutputStream

Flux de caractères

- FileWriter
- FileReader
- PrintWriter

Écriture dans un fichier (caractères)

Les flux de sortie en mode caractères **héritent de la classe abstraite Writer**

- Exemple : FileWriter

Méthodes :

- `public abstract void close() throws IOException`
  - Ferme le flux de sortie courant
- `public void write( int c ) throws IOException`
  - Écrit le caractère `c` sur le flux de sortie courant
- `public void write( char[] tab ) throws IOException`
  - Écrit le tableau de caractères `tab` sur le flux de sortie courant
- `public void write( String s ) throws IOException`
  - Écrit la chaîne de caractères `s` sur le flux de sortie courant

Remarque : ces méthodes lèvent une exception du type `IOException`

Écriture dans un fichier (caractères) : exemple

```
1  import java.io.*;
2
3  public class textFileWriter {
4
5      public static void main( String[] args ) {
6          FileWriter fw;
7
8          try{
9              fw = new FileWriter( "essai.txt" );
10             fw.write( "toto" );
11             fw.close();
12         }
13         catch( IOException e ) {
14             System.out.println( e );
15             fw = null;
16         }
17     }
18
19 }
```

Lecture dans un fichier (caractères)

Les flux d'entrée en mode caractères **héritent de la classe abstraite Reader**

- Exemple : FileReader

Méthodes :

- `public abstract void close() throws IOException`
  - Ferme le flux d'entrée courant
- `public int read( ) throws IOException`
  - Lit un caractère depuis le flux d'entrée courant. Retourne le caractère lu ou -1 si rien n'a pu être lu
- `public int read( char[] tab ) throws IOException`
  - `tab` doit être initialisé avec une longueur non nulle `tab.length`
  - Tente de lire `tab.length` caractères depuis le flux d'entrée courant et les stocke dans `tab`
  - Retourne le nombre de caractères réellement lus ou -1 si rien n'a pu être lu

Remarque : ces méthodes lèvent une exception du type `IOException`

Lecture dans un fichier (caractères) : exemple

```
1  import java.io.*;
2  public class textFileReader {
3      public static void main( String[] args ) {
4          FileReader fr;
5          int c;
6          try{
7              fr = new FileReader( "essai.txt" );
8              c = fr.read( );
9              while( c != -1 ) {
10                 System.out.print( (char) c );
11                 c = fr.read( );
12             }
13             fr.close();
14         }
15         catch( IOException e ) {
16             System.out.println( e );
17             fr = null;
18         }
19     }
20 }
```

Écriture dans un fichier (binaire)

Les flux de sortie en mode binaire **héritent de la classe abstraite OutputStream**

- Exemple : FileOutputStream

Méthodes :

- `public abstract void close() throws IOException`
  - Ferme le flux de sortie courant
- `public void write( int b ) throws IOException`
  - Écrit l'octet le poid faible de `b` en tant qu'octet sur le flux de sortie courant
- `public void write( byte[] tab ) throws IOException`
  - Écrit les octets contenus dans le tableau `tab` sur le flux de sortie courant

Remarque : ces méthodes lèvent une exception du type `IOException`

Écriture dans un fichier (binaire) : exemple

```
1  import java.io.*;
2
3  public class binFileOutputStream {
4
5      public static void main( String[] args ) {
6          FileOutputStream fo;
7          byte[] tab = {1, 2, 4};
8
9          try {
10             fo = new FileOutputStream( "essai2.txt" );
11             fo.write( tab );
12             fo.close();
13         }
14         catch( IOException e ) {
15             System.out.println( e );
16             fo = null;
17         }
18     }
19
20 }
```

Écriture dans un fichier (binaire) : classe avancée

Les méthodes de la classe FileOutputStream sont rudimentaires

- Méthodes plus avancées avec la classe **DataOutputStream**
- Le constructeur de `DataOutputStream` prend en paramètre un objet de la classe `FileOutputStream`

Méthodes :

- `public int size() throws IOException`
  - Retourne le nombre d'octets écrits sur le flux
- `public void writeBoolean( boolean b ) throws IOException`
  - Écrit le booléen `b` en tant qu'octet sur le flux de sortie courant
- `public void writeInt( int i ) throws IOException`
  - Écrit l'entier `i` en tant qu'une séquence de 4 octets sur le flux de sortie courant (octet de poids fort en premier)
- Et ainsi de suite pour tous les types primitifs de Java : `writeLong( )`, `writeShort()`, `writeChar()`, etc

Remarque : ces méthodes lèvent une exception du type `IOException`

Écriture avancée dans un fichier (binaire) : exemple

```
1  import java.io.*;
2
3  public class binDataOutputStream {
4      public static void main( String[] args ) {
5          FileOutputStream fo;
6          DataOutputStream dos;
7          try {
8              fo = new FileOutputStream( "essai2.txt" );
9              dos = new DataOutputStream( fo );
10             dos.writeChars( "totototititata" );
11             dos.writeInt( 42 );
12             dos.close();
13         }
14         catch( IOException e ) {
15             System.out.println( e );
16             fo = null;
17             dos = null;
18         }
19     }
20 }
```

Lecture dans un fichier (binaire)

Les flux d'entrée en mode binaire **héritent de la classe abstraite InputStream**

- Exemple : FileInputStream

Méthodes :

- `public abstract void close() throws IOException`
  - Ferme le flux d'entrée courant
- `public abstract int read( ) throws IOException`
  - Lit un caractère depuis le flux d'entrée courant. Retourne le caractère lu ou -1 si rien n'a pu être lu
- `public int read( char[] tab ) throws IOException`
  - `tab` doit être initialisé avec une longueur non nulle `tab.length`
  - Tente de lire `tab.length` caractères depuis le flux d'entrée courant et les stocke dans `tab`
  - Retourne le nombre de caractères réellement lus ou -1 si rien n'a pu être lu

Remarque : ces méthodes lèvent une exception du type `IOException`

Lecture dans un fichier (binaire) : classe avancée

Comme pour FileOutputStream, les méthodes de la classe FileInputStream sont rudimentaires

- Méthodes plus avancées avec la classe **DataInputStream**
- Le constructeur de `DataInputStream` prend en paramètre un objet de la classe `FileInputStream`

Méthodes :

- `public int size() throws IOException`
  - Retourne le nombre d'octets écrits sur le flux
- `public boolean readBoolean( ) throws IOException`
- `public int readInt( ) throws IOException`
- `public char readChar( ) throws IOException`
- Et ainsi de suite pour tous les types primitifs de Java : `writeLong( )`, `writeShort()`, `writeChar()`, etc

Remarque : ces méthodes lèvent une exception du type `IOException`