Flots d'octets

Table des matières

Copie d'un fichier

Nous allons écrire un programme qui copiera le contenu d'un fichier à l'écran ou dans un autre fichier.

Définition des flots d'entrée et de sortie

Les flots d'entrée et de sortie seront représentés par les variables in de type InputStream et out de type OutputStream. Le programme exigera au moins un paramètre qui sera le nom d'un fichier associé au flot d'entrée. Si un second paramètre est présent il représentera le nom du fichier associé au flot de sortie, sinon on utilisera la console écran représentée par System.out.

On déclare donc :

```
InputStream in;
OutputStream out=System.out;
```

Pour associer un fichier à un flot on utilise les classes **FileInputStream** et **FileOutputStream**.

Ceci nous donne :

```
if (args.length>0) in=new FileInputStream(args[0]);
if (args.length>1) out=new FileOutputStream(args[1]);
```

La copie

Lorsque les flots ont été initialisés, on réalise la copie du flot entrant dans le flot sortant en utilisant les méthodes **int read()** qui renvoie l'octet suivant ou -1 si le flot est terminé et **void write(int c)** qui écrit dans le flot sortant. La copie se fera donc avec l'instruction :

```
while ((c=in.read()) != -1) out.write(c);
```

Fermeture des flots

Les deux flots utilisés doivent être fermés après utilisation avec leur méthode **close()**. On termine donc avec :

```
in.close();
out.close();
```

Code du programme

On obtient le programme suivant :

```
import java.io.*;

public class copy {
```

```
 public static void main(String args[]) {
  InputStream in;
  OutputStream out=System.out;
  int c=0;
  if (args.length==0) {
   System.out.println("Passer un nom de fichier en argument.");
   System.exit(1);
  }
  try {
   if (args.length>0) in=new FileInputStream(args[0]);
   if (args.length>1) out=new FileOutputStream(args[1]);
   while ((c=in.read())!=-1) out.write(c);
   in.close();
   out.close();
  } catch (IOException e) {
   System.out.println(e.toString());
  }
 }

}
```

Notes :

- nous avons importé le package java.io qui contient les classes représentant des flots.
- les opérations sur les flots sont susceptibles de provoquer des exceptions de type IOException, nous devons donc les capturer dans une structure try...catch...

Utilisation du programme

Après avoir compilé copy.java, entrer la ligne de commande suivante:
`java copy copy.java`
Le contenu de copy.java s'affiche à l'écran.

Vous pouvez aussi utiliser la commande:
`java copy copy.java sortie.txt`
Un nouveau fichier nommé sortie.txt est créé, il contient la même chose que copy.java.

Fichiers de type texte

Pour gérer les fichiers de type texte on utilise des objets des classes **InputStreamReader** et **OutputStreamWriter**. Ces objets sont respectivement associés à des flots d'octets de type InputStream et OutputStream, ils ont pour tâche de convertir les octets en caractères. Pour pouvoir lire des lignes entières de texte on utilisera un objet de la classe **BufferedReader** associé à un InputStreamReader. Cet objet utilise un tampon (buffer) qui minimise le nombre d'accès au fichier.

Afficher le contenu d'un fichier

Nous allons écrire un programme qui affiche le contenu du fichier texte dont le nom est passé en paramètres (commande "type" du DOS).

Initialisation du flot

Nous construisons successivement un InputStream associé au fichier, un InputStreamReader qui effectue la conversion octets/caractères et un BufferedReader qui utilise un tampon et permet de lire des lignes entières. Cela nous donne :

```
InputStream ips=new FileInputStream(args[0]);
InputStreamReader ipsr=new InputStreamReader(ips);
BufferedReader br=new BufferedReader(ipsr);
```

Ces 3 instructions peuvent être réduites à une seule de la façon suivante:

```
BufferedReader br=new BufferedReader(
 new InputStreamReader(
 new FileInputStream(args[0])));
```

Lecture des lignes

Nous utilisons la méthode **readLine()** de la classe BufferedReader. Cette méthode renvoie une chaîne de caractères ou null lorsque la fin du flot est atteinte. Cela nous donne le code suivant pour afficher chaque ligne sur la console DOS:

```
String ligne;
while ((ligne=br.readLine())!=null)
 System.out.println(ligne);
```

Programme complet

On commence par vérifier qu'un nom de fichier a bien été passé en argument.

```
import java.io.*;

public class type {

 public static void main(String args[]) {
  if (args.length==0) {
   System.out.println("Passer un nom de fichier en paramètre.");
   System.exit(1);
  }
  try {
   InputStream ips=new FileInputStream(args[0]);
   InputStreamReader ipsr=new InputStreamReader(ips);
   BufferedReader br=new BufferedReader(ipsr);
   String ligne;
   while ((ligne=br.readLine())!=null)
    System.out.println(ligne);
   br.close();
  }
  catch (Exception e) {
   System.out.println(e.toString());
  }
 }

}
```

Utilisation

Après avoir compilé le programme, il suffit d'entrer la commande :
`java type type.java`
pour afficher le source du programme.

Applets et fichiers

Pour des raisons de sécurité les applets ne peuvent pas lire ou écrire des fichiers sur le système où elles s'exécutent. Par contre elles peuvent lire des fichiers existants sur le serveur html qui les a envoyées. Nous allons écrire une applet qui télécharge un fichier texte et l'affiche.

Les URL

Les URL sont gérés par la classe URL du package java.net.

Constructeur

La classe URL fournit un constructeur qui attend deux arguments:
- objet de type URL donnant l'URL du dossier de base
- chaîne de caractères donnant le nom du fichier

Nous utiliserons la méthode **getCodeBase()** de l'applet pour obtenir l'URL de base. La création de l'URL associée au fichier à lire prendra la forme :

```
URL url=new URL(getCodeBase(),nomFichier);
```

Cette création peut générer une exception de type MalformedURLException, elle devra donc se faire dans une structure try...catch...

La méthode openStream()

La classe URL permet d'associer le fichier pointé à un flot d'entrée par l'intermédiaire de la méthode openStream(). Celle-ci renvoie un flot de type InputStream. Nous aurons donc :

```
InputStream ips=url.openStream();
```

Lecture du fichier

La lecture du fichier se fera en utilisant un objet de la classe BufferedReader branché sur un InputStreamReader associé au flot d'octets fourni par l'URL.

Les lignes lues seront ajoutées à l'objet zoneTexte de type TextArea. Ceci nous donne le code suivant :

```
BufferedReader in=new BufferedReader(new InputStreamReader(ips));
String ligne;
while ((ligne=in.readLine())!=null)
```

```
 zoneTexte.append(ligne+"\n");
in.close();
```

Source complet de l'applet

Nous importons les packages java.net et java.io en plus des packages habituels. La méthode init installe le TextArea zoneTexte et lit un éventuel nom de fichier passé en paramètre. La méthode start se charge de la lecture du fichier.

```
import java.applet.*;
import java.awt.*;
import java.net.*;
import java.io.*;

public class textapplet extends Applet {

 String nomFichier;
 TextArea zoneTexte=new TextArea();

 public void init() {
  nomFichier=getParameter("fichier");
  if (nomFichier==null) nomFichier="textapplet.java";
  setLayout(new BorderLayout());
  add(zoneTexte,BorderLayout.CENTER);
 }

 public void start() {
  try {
   URL url=new URL(getCodeBase(),nomFichier);
   //flot associé à l'url
   InputStream ips=url.openStream();
   BufferedReader in=new BufferedReader(
    new InputStreamReader(ips));
   String ligne;
   while ((ligne=in.readLine())!=null)
    zoneTexte.append(ligne+"\n");
   in.close();
  } catch (Exception e) {
   zoneTexte.append("\n"+e.toString()+"\n");
  }
 }

}
```

La classe File

La classe File permet de gérer les fichiers et les dossiers de façon externe (nom, taille, existence, type, etc...).

Constructeurs et méthodes

Constructeurs

- File(File d, String n)
  d est le dossier du fichier, n est son nom.
- File(String n)
  n est le nom du fichier.

Méthodes principales

- boolean exists()
  indique si le fichier existe.
- String getCanonicalPath()
  renvoie le chemin complet d'accès au fichier sous forme canonique (sans . ou ..).
- String getName()
  renvoie le nom du fichier.
- boolean isDirectory()
  indique s'il s'agit d'un dossier
- boolean isFile()
  indique s'il s'agit d'un vrai fichier
- int length()
  renvoie la taille du fichier
- String[] list()
  renvoie la liste des fichiers d'un dossier sous forme de tableau de chaînes.

Le programme dir

Ecrivons le programme **dir** qui affiche le contenu d'un dossier.

```
import java.io.*;

public class dir {

 public static void main(String[] args) {
  File f=new File(".");
  System.out.println(f.getAbsolutePath());
  String[] liste=f.list();
  for (int i=0; i<liste.length; i++) {
   File ff=new File(liste[i]);
   if (ff.isDirectory()) System.out.println("Dossier \t"+liste[i]);
   else System.out.println(""+ff.length()+" \t"+liste[i]);
  }
 }

}
```

Le dossier de départ est nommé ".", c'est le dossier courant. La liste des fichiers contenus dans ce dossier est obtenue par la méthode **list** sous forme de tableau de chaînes. Les fichiers représentant un sous dossier (méthode **isDirectory()**) sont affichés précédés du mot dossier, les autres sont affichés précédés par leur taille (méthode **length**).

Résumé Fichiers

Lecture

On utilise un objet de type InputStream, ou plutôt un de ses dérivés, qui va représenter la source des données.

Hiérarchie :

```
· class java.io.InputStream
   · class java.io.ByteArrayInputStream
   · class java.io.FileInputStream
   · class java.io.FilterInputStream
      · class java.io.BufferedInputStream
      · class java.util.zip.CheckedInputStream
      · class java.io.DataInputStream
      · class java.security.DigestInputStream
      · class java.util.zip.InflaterInputStream
         · class java.util.zip.GZIPInputStream
         · class java.util.zip.ZipInputStream
      · class java.io.LineNumberInputStream
      · class java.io.PushbackInputStream
   · class java.io.ObjectInputStream
   · class java.io.PipedInputStream
   · class java.io.SequenceInputStream
   · class java.io.StringBufferInputStream
```

L'objet InputStream lit une suite d'octets (Byte).
Exemple : System.in est un InputStream.

Pour obtenir un InputStream

- L'InputStream est déjà construit comme System.in
- L'InputStream est fourni par un autre objet, un objet URL fournit un InputStream par sa méthode openStream().
- On utilise un objet dérivé : FileInputStream permet d'associer l'InputStream à un fichier, ByteArrayInputStream permet d'associer l'InputStream à un tableau d'octets.

Fichiers texte

Pour un fichier texte, on utilise ensuite un objet Reader pour filtrer les octets reçus et les traduire en caractères.

Hiérarchie

```
· class java.io.Reader
   · class java.io.BufferedReader
      · class java.io.LineNumberReader
   · class java.io.CharArrayReader
   · class java.io.FilterReader
      · class java.io.PushbackReader
   · class java.io.InputStreamReader
      · class java.io.FileReader
   · class java.io.PipedReader
   · class java.io.StringReader
```

Les objets dérivés de Reader ont une méthode read() qui renvoie un caractère.

Reader étant une classe abstraite, c'est un InputStreamReader qu'on utilise. On construit donc en utilisant un InputStream in un objet InputStreamReader inSR=new InputStreamReader(in).

Enfin pour lire des lignes de texte, on utilise comme Reader un BufferedReader associé à l'InputStreamReader. La classe BufferedReader fournit la méthode readLine() qui renvoie un String. Lorsqu'il n'y a plus de chaînes à lire, cette méthode renvoie null.

Exemples

Pour lire à partir de la console DOS

On utilise System.in.

```
BufferedReader br=new BufferedReader(
    new InputStreamReader(System.in));
```

On a associé le BufferedReader nommé in à l'InputStreamReader associé à l'InputStream in de la classe System pour gérer l'entrée clavier.

Ainsi :

```
String S=br.readLine();
```

permet de récupérer une chaîne entrée sur la console.

Pour lire un fichier texte local

On construit comme InputStream un dérivé de type FileInputStream en donnant le nom de fichier :

```
FileInputStream fin=new FileInputStream(nomfichier);
BufferedReader br=
 new BufferedReader(new InputStreamReader(fin));
```

Pour lire un fichier distant associé à une URL

On obtient l'InputStream avec la méthode openStream() d'un objet URL.

```
URL source=new URL(nomfichier);
BufferedReader br=new BufferedReader(
 new InputStreamReader(source.openStream()));
```

Exercices

1. Conversion Euro
   Ecrire une applet convertissant plusieurs monnaies en Euro, les taux de changes étant donnés dans le fichier euro.txt.

2. Afficheur d'images
   Reprendre l'applet Visio vue dans le TD Images pour qu'elle affiche des images dont le nom est contenu dans le fichier liste.txt.

3. Coupeur de fichiers
   Ecrire les programmes **coupe** et **recolle** qui permettent de couper un gros fichier en fichiers plus petits, puis de reconstituer le fichier d'origine.

4. Liste de fichiers
   Ecrire le programme fenêtré ListeFic qui permet de naviguer dans les dossiers d'un lecteur et affiche la liste des fichiers contenus dans chaque dossier.