Les Flux : principes de base

Introduction

Les flux permettent de travailler avec des données – des fichiers – provenant de l'extérieur. Toutes les commandes de type Ouvrir, Enregistrer, Enregistrer sous … des programmes classiques font appel aux flux. Les flux sont aussi appelés I/O : en anglais, Input/Output, entrées/sorties. Nous allons voir dans ce chapitre les flux de base, permettant de lire et d'écrire du texte (On différencie les caractères et le texte en général). Nous verrons également comment utiliser la mise en mémoire cache du contenu d'un flux (en fait des données qu'il transporte) pour accélérer ses performances (vitesse, fiabilité). Le programme final définira une petite interface Swing, un gros bouton qui, quand on le pressera, fera charger un fichier dans un JTextArea.

Ligne de commande

Vous connaissez déjà 1 type de flux basique de ligne de commande : System.out. Sachez qu'il existe pour les entrées System.in et pour les erreurs System.err. Dans la mesure où les applications de ligne de commande uniquement se font très rares, nous n'étudierons pas ces méthodes.

Flux d'entrée / Flux de sortie

La création d'un flux de base se fait en 2 étapes : création d'un flux de *support* d'entrée ou de sortie, création d'un flux adapté à l'opération que l'on désire obtenir.

```
// flux de support (valable dans tous les cas)
FileInputStream fis = new FileInputStream(" fichier.extension ") ;
// flux particulier (exemple)
*InputStream zis = new *InputStream(fis);
```

On crée un flux de support fis grâce à FileInputStream, ce flux (ici, d'entrée) travaille avec le fichier fichier.extension (exemple : Readme.txt), à ce flux, on associe un flux particulier. L'étoile représente tous les types de flux : ZipInputStream, DataInpuStream, GzipInputStream …
Ici sont présentés des flux d'entrée, pour les flux de sortie, on renplace 'Input' par 'Output'.
Exemple concret :

```
FileOutputStream fos = new FileOutputStream("fichier.txt");
ZipOutputStream zout = new ZipOutputStream(fos);
```

Classes de flux d'entrée/sortie

1) Entrée

Public int available ( ) throws IOException
Pernet de retourner le nombre d'octets que le flux peut lire. (rarement nécessaire)

Public long skip (long n) throws IOException
Permet d'ignorer les n prochains caractères. (rarement nécessaire)

2) Sortie

Public void flush ( )
Vide immédiatement le contenu d'un flux (vide le tampon et distribue ce qu'il contient). (utile)

Public void close ( ) throws IOException
Permet de fermer un flux. Même si vous ne fermez pas un flux, aucune erreur ne sera (normalement) provoquée. Cependant, chaque flux devrait être fermé.

Mise en mémoire tampon

La mise ne mémoire tampon d'un flux permet d'accélérer le performances, voici la procédure à adopter :

```
// création du support
FileInputStream fis = new FileInputStream( " fichier.txt ");
// création du flux de mise en tampon
BufferedInputStream bis = new BufferedInputStream(fis);
// on joint le BufferedInputStream à un flux particulier, par exemple:
DataInputStream dis = new DataInputStream(bis) ;
```

Readers/Writers

Les readers et writers sont un sujet assez long à traiter. Il serait hors-sujet de les traiter dans leur intégralité dans ce chapitre présentant les concepts clé de I/O. Nous allons donc voir uniquement une partie basique.
Les Readers et Writers permettent de lire et d'écrire des données dans des fichiers. Ils sont très utiles pour travailler avec des fichiers de texte. Ils sont utilisés en remplacement des méthodes FileInputStream et FileOutputStream. Par exemple, pour utiliser un FileWriter et un FileReader :

```
// FileWriter:
// déterminer le texte à écrire dans le fichier en le prenant, par exemple, dans un TextArea
    String texte = new String(jTextArea1.getText());
// ouvrir le FileWriter avec pour argument le nom du fichier de sortie
    FileWriter lu = new FileWriter("fichier.txt");
// procédure de mise en cache
    BufferedWriter out = new BufferedWriter(lu);
// écrire dans le FileWriter les informations du String texte
    out.write(texte);
// fermer le flux
    out.close();

// FileReader :
//ouvre un FileReader
FileReader fr = new FileReader(" fichier.txt ") ;
// procédure de lecture des caractères…
While (true) {
// lire les caractères
    int i = fr.read();
// -1 représente le moment où il n'y a plus de caractères à lire, le boucle while doit alors s'
// arrêter
    if (i == -1) break;
}
```

Projet: afficheur

Ce programme permet de capturer un texte d'une zone de texte et de l'écrire dans un fichier, de le lire d'un fichier pour l'écrire dans la zone de texte.
Il utilise 2 classes, Swing, un FileWriter et un FileInputStream …

1° classe :

```
package afficheur;

import javax.swing.UIManager;

public class Application {
  boolean packFrame = false;

  //Construire l'application
  public Application() {
    // appel à la classe 'afficheur'
    afficheur frame = new afficheur();
    //Valider les cadres ayant des tailles prédéfinies
    //Compacter les cadres ayant des infos de taille préférées - ex. depuis leur
disposition
    if (packFrame) {
      frame.pack();
    }
    else {
      frame.validate();
    }
    frame.setVisible(true);
  }

  //Méthode principale
  public static void main(String[] args) {
    try {
      // pour l'apparence 'Windows'
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    new Application();
  }
}
```

2 nd classe :

```
// le programme comprend 2 classes regroupées dans 1 package:
package afficheur;

// importation de packages nécessaires
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

// début du programmes, la classe afficheur doit étendre (extends) de JFrame
// pour définir un cadre Swing à l'écran
public class afficheur extends JFrame {
  // Conteneur
  JPanel contentPane;
  // Layout
  BorderLayout borderLayout1 = new BorderLayout();
  // Composant 1 : le bouton
  JButton jButton1 = new JButton();
  // barres de défilement pour le composant 2
  JScrollPane jScrollPane1 = new JScrollPane();
  // Composant 2 : la zone de texte
  JTextArea jTextArea1 = new JTextArea();
  JButton jButton2 = new JButton();

  //Construire le cadre
  public afficheur() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
      // essayer d'exécuter le méthode jbInit()
      jbInit();
    }
    catch(Exception e) {
```

```
    e.printStackTrace();
  }
}

//Initialiser le composant
private void jbInit() throws Exception  {
  // texte du bouton
  jButton1.setText("Charger");
  // écouteur d'actions (fait partie du gertionnaire d'evènements)
  jButton1.addActionListener(new afficheur_jButton1_actionAdapter(this));
  // définission du conteneut courant
  contentPane = (JPanel) this.getContentPane();
  // assignation à ce conteneur d'un layout précis
  contentPane.setLayout(borderLayout1);
  // taille de la fenêtre
  this.setSize(new Dimension(400, 300));
  // label de la fenêtre
  this.setTitle("Exemple pour les flux");
  // texte qui apparaît quand on laisse la souris immobile un certain temps
  // su la zone de texte
  jTextArea1.setToolTipText("Exemple d'un chapitre de JGFL");
  // ajout des composants
  jButton2.setText("Enregistrer");
  jButton2.addActionListener(new afficheur_jButton2_actionAdapter(this));
  contentPane.add(jButton1, BorderLayout.SOUTH);
  contentPane.add(jScrollPane1, BorderLayout.CENTER);
  contentPane.add(jButton2, BorderLayout.NORTH);
  jScrollPane1.getViewport().add(jTextArea1, null);
}




//Remplacé, ainsi nous pouvons sortir quand la fenêtre est fermée
protected void processWindowEvent(WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    // pour fermer 'proprement' la fenêtre au clic sur la croix (sous Win95)
    System.exit(0);
  }
}

// gestionnaire d'evènements, 2nd partie
void jButton1_actionPerformed(ActionEvent e) {
     // appeler cette méthode
     charger();
}

// méthode qui permet de prendre le texte dans la zone de texte et de le
// transmettre au FileWriter
public void enregistrer() {
try {
   String texte = new String(jTextArea1.getText());
   FileWriter lu = new FileWriter("fichier.txt");
   BufferedWriter out = new BufferedWriter(lu);
   out.write(texte);
   out.close();
} catch (Exception err) {;}
}


// ma méthode qui permet de lire les informations du fichier 'fichier.txt'
// et de les imprimer dans la zone de texte
public void charger() {
  // les exceptions doivent être interceptées ('catchées')
    try {
       // Flux d'entrée
       FileInputStream fis = new FileInputStream("fichier.txt");
       // nbre de caractères dans le fichier
       int n;
       // tant que ce nombre de caractères est supérieur à 0...
       while ((n = fis.available()) > 0) {
```

```
            // chaque caractère associé à 1 byte
            byte[] b = new byte[n];
            int result = fis.read(b);
            // fin du flux = plus rien à lire = sortie de la boucle
            if (result == -1) break;
            String s = new String(b);
            jTextArea1.setText(s);
          }
        } catch (IOException err) { System.out.println("Erreur: " + err);
    }


}
// classes pour la gestion d'évènements.
class afficheur_jButton1_actionAdapter implements java.awt.event.ActionListener {
  afficheur adaptee;

  afficheur_jButton1_actionAdapter(afficheur adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jButton1_actionPerformed(e);
  }
}

  void jButton2_actionPerformed(ActionEvent e) {
      enregistrer();
  }
}

class afficheur_jButton2_actionAdapter implements java.awt.event.ActionListener {
  afficheur adaptee;

  afficheur_jButton2_actionAdapter(afficheur adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jButton2_actionPerformed(e);
  }
}
```