Chapitre 12

L’archivage des données

L’archivage des données

- Objectif : stocker les informations sous forme de fichiers (sur disque dur, clef USB, carte mémoire, etc.)

- Le langage Java utilise le concept de flux (***stream***).

1. La notion de flux

- Cela revient à considérer toute opération en entrée (vers le programme) et en sortie (du programme) comme un même ensemble d'opérations
  - flux entrant : clavier, fichier, tablette graphique, …
  - flux sortant : écran, imprimante, …

- Nous nous intéressons aux flux liés aux lectures et écritures vers ou dans un fichier.

- Existence en java d'objets pré définis pour gérer les flux (package java.io → utiliser l'instruction **`import java.io.*)`**

- Nous présentons dans ce chapitre deux techniques d'archivage afin d'en comprendre les différents mécanismes

2. Les fichiers texte

Deux déclarations possibles :

- **`BufferedWriter fW;`**
  - définit un objet fW de type **`BufferedWriter`**, utilisé pour enregistrer des données dans un fichier (Writer)
- **`BufferedReader fR;`**
  - définit un objet fR de type **`BufferedReader`**, utilisé pour lire (Reader) les données contenues dans un fichier afin de les placer dans des variables.

Ouverture d'un fichier texte en lecture
et lecture d'une chaine de caractères

```
import java.io.*;
...
BufferedReader fR;

try
{
    File f = new File("un_fichier.txt");
    fR = new BufferedReader (new FileReader(f));
    String chaine = fR.readLine();
    fR.close() ;
}
catch (IOException e)
{
    System.out.println("Erreur : " + e.getMessage() );
}
```

- L'ouverture du fichier est réalisé en lecture grâce à l'instruction

```
fR  = new BufferedReader(new FileReader(f));
```

  - l'opération **`new File(nomDuFichier)`** permet de lier à un nom physique un nom logique
  - L'appel au constructeur **`FileReader()`** permet l'ouverture du fichier en lecture caractère par caractère. Il fournit en retour l'adresse du début de fichier
  - **`BufferedReader()`** permet ensuite la lecture ligne par ligne. L'adresse du début de fichier est alors mémorisé dans l'objet fR

Ouverture d'un fichier texte en écriture

et écriture d'une chaine de caractères

```
import java.io.*;
...
BufferedWriter fW;

try
{
    File f = new File("un_fichier.txt");
    String chaine = "un peu de texte";
    fW = new BufferedWriter(new FileWriter(f));
    fW.write(chaine,0,chaine.length());
    fW.newLine();
    fW.close() ;
}
catch (IOException e)
{
    System.out.println("Erreur" + e.getMessage() );
}
```

- L’ouverture du fichier est réalisée en écriture grâce à l’instruction

```
fW = new BufferedWriter(new FileWriter(f));
```

  - Si le fichier spécifié en paramètre n’existe pas, et :
    - Si le chemin d’accès à ce fichier dans l’arborescence du disque est valide, alors le fichier est créé
    - Sinon erreur du type FileNotFoundException
  - Si le fichier existe, il est ouvert, et son contenu est totalement effacé.

3. Les fichiers binaires

Permet d'écrire ou de lire des données de n'importe quelle type dans un fichier, et pas uniquement que du texte.

Ecriture

Utilisation de la classe **`DataOutputStream`**

Lecture

Utilisation de la classe **`DataInputStream`**

3.1 Écriture dans un fichier binaire

```
import java.io.*;

class EcrireFichierBinaire
{
   public static void main(String[] argv) throws IOException
   {
      DataOutputStream fW;

      fW = new DataOutputStream(
            new FileOutputStream("fichier.dat"));
```

```
        fW.writeUTF("bonjour");
        fW.writeInt(3);
        fW.writeLong(100000);
        fW.writeFloat(2.0f);
        fW.writeDouble(3.5);
        fW.writeChar('a');
        fW.writeBoolean(false);
        System.out.println("octets écrits:"+fW.size());
        fW.close();
    }
}
```

Remarque

Pour accélérer les temps d'accès au fichier, on peut utiliser une mémoire tampon :

```
fW = new DataOutputStream(
     new BufferedOutputStream(
     new FileOutputStream("fichier.dat")));
```

Au lieu de :

```
fW = new DataOutputStream(
     new FileOutputStream("fichier.dat"));
```

3.2 Lecture dans un fichier binaire

```
import java.io.*;

class LireFichierBinaire
{
   public static void main(String[] argv) throws IOException
   {
      DataInputStream fR;

      fR = new DataInputStream(
            new FileInputStream("fichier.dat"));
```

```
        System.out.println(fR.readUTF());
        System.out.println(fR.readInt());
        System.out.println(fR.readLong());
        System.out.println(fR.readFloat());
        System.out.println(fR.readDouble());
        System.out.println(fR.readChar());
        System.out.println(fR.readBoolean());
        fR.close();
    }
}
```

Remarque

Pour accélérer les temps d'accès au fichier, on peut utiliser une mémoire tampon :

```
fR = new DataInputStream(
     new BufferedInputStream(
     new FileInputStream("fichier.dat")));
```

Au lieu de :

```
fR = new DataInputStream(
     new FileInputStream("fichier.dat"));
```

4. Utilisation de la classe `File`

La classe **`java.io.File`** permet de :

- déterminer si un fichier existe
- connaître la taille d'un fichier
- connaître le chemin complet d'un fichier
- lister les fichiers d'un répertoire
- renommer un fichier
- supprimer un fichier
- …

4.1 Déterminer si un fichier existe

On utilise la méthode **`exists()`**

```
File fichier = new File("fichier.dat");


if( fichier.exists() )
   System.out.println("Le fichier existe");
else
   System.out.println("Fichier introuvable");
```

4.2 Connaître la taille d'un fichier

La taille en octets d'un fichier s'obtient avec la méthode **`length()`**

```
File fichier = new File("fichier.dat");
long taille = fichier.length();
```

4.3 Connaître le chemin complet d'un fichier

```
File fichier = new File("fichier.dat");

// Nom complet avec chemin d'accès absolu
// (depuis la racine)
// ex : c:\dossier\fichier.dat
String nom_complet = fichier.getAbsolutePath());
```

4.4 Lister les fichiers d'un répertoire

```
File répertoire = new File("c:\temp");
String[] liste_fichiers;

liste_fichiers = répertoire.list();

for(int i=0; i<liste_fichiers.length; i++)
   System.out.println(liste_fichiers[i]);
```

4.5 Supprimer un fichier

```
File fichier = new File("fichier.dat");


if( fichier.delete() == false )
    System.out.print("Suppression impossible");
```

5. Les fichiers d'objets

- Le langage java propose des outils permettant le stockage ainsi que la lecture d'objets dans un fichier.
- Ces outils font appel à des **mécanismes de sérialisation**
- Utilisation de flux spécifiques du package java.io :
  - `ObjectOutputStream`
  - `ObjectInputStream`

- Un objet est **sérialisé** afin de pouvoir être transporté sur un flux de fichier.

  $\Rightarrow$ l'objet peut être stocké dans un fichier (écriture) et reconstruit à l'identique (lors de la lecture).

  $\Rightarrow$ Un objet peut ainsi exister entre deux exécutions d'un programme, ou entre deux programmes : c'est la **persistance objet**.

- Cette méthode est applicable à tous les **objets prédéfinis** du langage Java, tels que les String, Vector, Hashtable, …

- Dans les autres cas, il faut rendre l'objet sérialisable :

```
public class Exemple implements Serializable
{
   // données et méthodes
}
```

- Seules les variables d'instance seront pris lors de la sérialisation, alors que les variables de classes (définies en static) ne peuvent être sérialisées.

- On peut ensuite sauver les objets avec la méthode `writeObject()` et les lire avec `readObject()`

- Exemple : archiver une classe d'étudiants
  - Nous souhaitons stocker l'ensemble du dictionnaire dans un fichier
  - Nous devons donc d'abord rendre sérialisable les objets que nous souhaitons sauvegarder

```
public class Etudiant implements Serializable
{
   // ...
}

public class Classe_etudiants implements
  Serializable
{
   Vector<Etudiant> liste_etudiants ;
}
```

```
import java.io.*;

class TestSerialisation
{
   public static void main(String[] argv)throws IOException
   {
      Classe_etudiants classe = new Classe_etudiants();
```

- Ecriture dans le fichier

```
ObjectOutputStream fWo;

fWo = new ObjectOutputStream(
            new FileOutputStream("classe_etudiants.dat"));

fWo.writeObject(classe);

fWo.close();
```

- Lecture dans le fichier

```
ObjectInputStream fRo;

fRo = new ObjectInputStream(
            new FileInputStream("classe_etudiants.dat"));

classe = (Classe_etudiants) fRo.readObject();

fRo.close();
```