Chapitre 3

Gestion des entrées/sorties simples

Le package `java.io` propose un ensemble de classes permettant de gérer la plupart des entrées/sorties d'un programme. Cette gestion consiste à créer un objet *flux* dans lequel transitent les données à envoyer ou à recevoir. Un flux connecte un objet Java à un autre élément. Deux cas sont illustrés dans ce chapitre : les interactions avec un utilisateur (entrée clavier et sortie écran) et les accès en lecture ou écriture à un fichier.

3.1 Flux d'entrée

Un flux d'entrée est une instance d'une sous-classe de `InputStream`. Chaque classe de flux d'entrée a son propre mode d'échange de données qui spécifie un format particulier de données ou un accès particulier. Les classes les plus couramment utilisées sont :

- **ByteArrayInputStream** permet de lire le flux d'entrée sous la forme d'octets (*byte*) ;
- **DataInputStream** permet de lire le flux d'entrée sous la forme de types de données primitifs de Java. Il existe des méthodes pour récupérer un entier, un réel, un caractère,…
- **FileInputStream** est utilisé pour lire le contenu d'un fichier. Les objets de cette classe sont souvent encapsulés dans un autre objet de classe `InputStream` qui définit le format des données à lire.
- **ObjectInputStream** permet de lire des objets (c-à-d des instances de classes Java) à partir du flux d'entrée, si ces objets implémentent les interfaces `java.io.Serializable` ou `java.io.Externalizable`.
- **Reader** n'est pas une sous-classe de `InputStream` mais représente un flux d'entrée pour chaînes de caractères. Plusieurs sous-classes de `Reader` permettent la création de flux pour chaînes de caractères.
- **Scanner** n'est pas une sous-classe de `InputStream`, mais un `Iterator` qui permet de lire un flux (fichier ou chaîne de caractères par exemple) "mot" par "mot" en définissant le délimiteur entre les mots (espace par défaut).

La lecture de données à partir d'un flux d'entrée suit le déroulement suivant :

1. **Ouverture du flux :** Elle se produit à la création d'un objet de la classe `InputStream`. Lors de l'appel au constructeur, on doit préciser quel élement externe est relié au flux (par exemple un nom de fichier ou un autre flux).
2. **Lecture de données :** Des données provenant du flux sont lues au moyen de la méthode `read()`) ou d'une méthode équivalente. La méthode précise à employer dépend du type de flux ouvert.
3. **Fermeture du flux :** Quand le flux n'est plus nécessaire, il doit être fermé par la méthode `close()`.

3.1.1 Lecture des entrées clavier

Les données provenant de l'utilisation du clavier sont transmises dans un flux d'entrée créé automatiquement pour toute application Java. On accède à ce flux par la variable statique de la classe `java.lang.System` qui s'appelle `in`. Ce flux est alors utilisé comme paramètre d'entrée du constructeur d'un autre flux d'entrée. Pour cet autre flux, on utilise généralement une sous-classe de `Reader` pour récupérer les entrées de l'utilisateur sous la forme d'une chaîne de caractères. La classe `Clavier` en donne un exemple :

```
import java.io.*;

public class Clavier {
  public static void main(String[] args) {
    try {
      BufferedReader flux = new BufferedReader(
         new InputStreamReader(System.in));
      System.out.print("Entrez votre prenom : ");
      String prenom = flux.readLine();
      System.out.println("Bonjour " + prenom);
      flux.close();
    } catch (IOException ioe) {
      System.err.println(ioe);
    }
  }
}
```

3.1.2 Lecture à partir d'un fichier

Un fichier est représenté par un objet de la classe `java.io.File`. Le constructeur de cette classe prend en paramètre d'entrée le chemin d'accès du fichier. Le flux d'entrée est alors créé à l'aide de la classe `FileInputStream` sur lequel on peut lire caractère par caractère grâce à la méthode `read()`. L'exemple suivant présente une méthode pour afficher le contenu d'un fichier :

```java
import java.io.*;

public class LectureFichier {

  public static void main(String[] args) {
    try {
      File fichier = new File("monFichier.txt");
      FileInputStream flux = new FileInputStream(fichier);
      int c;
      while ((c = flux.read()) > -1) {
        System.out.write(c);
      }
      flux.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Il arrive souvent d'enregistrer des données dans des fichiers textes. Il peut alors être utile d'utiliser un `BufferedReader` ou un `Scanner` pour effectuer la lecture. Dans les exemples suivants, on considère une matrice 10×10 enregistrée dans un fichier texte `matrice.txt` ligne par ligne, avec les colonnes séparées par des espaces :

```java
import java.io.*;

public class LectureMatrice {

  public static void main(String[] args) {
    try {
      FileReader fileReader = new FileReader("matrice.txt");
      BufferedReader reader = new BufferedReader(fileReader);
      while (reader.ready()) {
        String[] line = reader.readLine().split(" ");
        for (String s  : line) {
          System.out.print(s);
        }
        System.out.println();
      }
      reader.close();
      fileReader.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

On peut effectuer un traitement similaire avec un `Scanner` :

```
import java.io.*;
import java.util.Scanner;

public class LectureMatriceScanner {
  public static void main(String[] args) {
    try {
      Scanner fileScanner = new Scanner(new File("matrice.txt"));
      while (fileScanner.hasNextLine()) {
        Scanner lineScanner = new Scanner(fileScanner.nextLine());
        while (lineScanner.hasNext())
           System.out.print(lineScanner.next());
        System.out.println();
      }
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    }
  }
}
```

3.1.3 Lecture d'objets enregistrés

Il est parfois utile d'enregistrer l'état d'objet (de n'importe quelle classe implémentant les interfaces `java.io.Serializable` ou `java.io.Externalizable`) pour des exécutions futures. Le flux d'entrée est encore créé à l'aide de la classe `FileInputStream` et est ensuite encapsulé dans un autre flux spécifiant le format des données à lire. L'exemple suivant illustre la lecture d'un objet de la classe `Date` dans un fichier nommé `monFichier.dat` :

```
import java.io.*;
import java.util.Date;

public class LectureDate {
  public static void main(String[] args) {
    try {
      File fichier = new File("monFichier.dat");
      ObjectInputStream flux = new ObjectInputStream(
      new FileInputStream(fichier));
      Date laDate = (Date) flux.readObject();
      System.out.println(laDate);
      flux.close();
    } catch (IOException ioe) {
      System.err.println(ioe);
    } catch (ClassNotFoundException cnfe) {
      System.err.println(cnfe);
    }
  }
}
```

L'objet qui est lu dans le fichier doit être une instance de la classe `java.util.Date`.

3.2 Flux de sortie

Un flux de sortie est une instance d'une sous-classe de `OutputStream`. Comme pour les flux d'entrée, chaque classe de flux de sortie a son propre mode d'écriture de données. Les classes les plus couramment utilisées sont :

- **ByteArrayOutputStream** permet d'écrire des octets vers le flux de sortie ;
- **DataOutputStream** permet d'écrire des types de données primitifs de Java vers le flux de sortie.
- **FileOutputStream** est utilisé pour écrire dans un fichier. Les objets de cette classe sont souvent encapsulés dans un autre objet de classe `OutputStream` qui définit le format des données à écrire.
- **ObjectOutputStream** permet d'écrire des objets (c-à-d des instances de classes Java) vers le flux de sortie, si ces objets implémentent les interfaces `Serializable` ou `Externalizable`.
- **Writer** n'est pas une sous-classe de `OutputStream` mais représente un flux de sortie pour chaînes de caractères. Plusieurs sous-classes de `Writer` permettent la création de flux pour chaînes de caractères.

L'écriture de données vers un flux de sortie suit le même déroulement que la lecture d'un flux d'entrée :

1. **Ouverture du flux :** Elle se produit lors de la création d'un objet de la classe `OutputStream`.

2. **Ecriture de données :** Des données sont écrites vers le flux au moyen de la méthode `write()` ou d'une méthode équivalente. La méthode précise à employer dépend du type de flux ouvert.

3. **Fermeture du flux :** Quand le flux n'est plus nécessaire, il doit être fermé par la méthode `close()`.

3.2.1 Ecriture sur la sortie standard "écran"

Comme pour les entrées du clavier, l'écriture vers l'écran fait appel à la variable statique `out` de la classe `System`. On appelle généralement la méthode `System.out.print` ou `System.out.println` comme cela a été fait dans de nombreux exemples de ce livret.

3.2.2 Ecriture dans un fichier

L'écriture dans un fichier se fait par un flux de la classe `FileOutputStream` qui prend en entrée un fichier (instance de la classe `File`). Ce flux de sortie permet d'écrire des caractères dans le fichier grâce à la méthode `write()`. L'exemple suivant présente une méthode pour écrire un texte dans un fichier :

```
import java.io.*;

public class EcritureFichier {

  public static void main(String[] args) {
    try {
      File fichier = new File("monFichier.txt");
      FileOutputStream flux = new FileOutputStream(fichier);
      String texte = "Hello World!";
      for (int i = 0; i < texte.length(); i++) {
        flux.write(texte.charAt(i));
      }
      flux.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

On peut également utiliser un `Writer` comme un `FileWriter` pour écrire des chaînes de caractères dans un fichier assez simplement. Dans l'exemple suivant, on écrit une série de 10 lignes de 10 entiers aléatoires séparés par des espaces dans un fichier pouvant être lu par la classe `LectureMatrice` :

```
import java.io.*;
import java.util.Random;

public class EcritureMatrice {

  public static void main(String[] args) {
    try {
      FileWriter writer = new FileWriter("random.txt");
      Random generator = new Random(System.currentTimeMillis());
      for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++)
           writer.write(generator.nextInt() + " ");
        writer.write("\n");
      }
      writer.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

3.2.3 Ecriture d'objets

Le flux de sortie peut également être encapsulé dans un flux de type `ObjectOutputStream`, comme le montre l'exemple suivant pour écrire la date courante dans un fichier :

```
import java.io.*;
import java.util.Date;

public class EcritureDate {

  public static void main(String[] args) {
    try {
      File fichier = new File("monFichier.dat");
      ObjectOutputStream flux = new ObjectOutputStream(
      new FileOutputStream(fichier));
      flux.writeObject(new Date());
      flux.close();
    } catch (IOException ioe) {
      System.err.println(ioe);
    }
  }
}
```