Flux de données (Rappels de cours et exercices corrigés)

Le langage Java fournit une API de manipulation de flux de données (Stream en anglais), possède 2 classes prinicpales

- `InputStream` : cette classe abstraite définit les fonctions de lecture (entrée ou *input* en anglais),
- `OutputStream` : cette classe abstraite définit les fonctions d'écriture (sortie ou *output* en anglais).

Un flux d'entrées/sorties (IO Stream) peut véhiculer toutes sortes de données, brutes (binaire octet), structurées, formatées etc. On distingue :

- Les flux octet (base de construction des autres flux)
- Les flux de données
- Les flux de texte
- Les flux d'objets

1. Flux de données :

public final class System **extend**s Object {
Il existe trois flux prédéfinis.
static InputStream in ; //flux associé à l'entrée standard d'une application, s
static PrintStream out **;** // flux associé à sa sortie standard
static PrintStream err **;** // flux associé à sa sortie standard d'erreurs.
}

➔ Utilisation du bloc de traitement d'exception : **try .. catch** et affichage de la pile des appels de méthodes

```
import java.io.*;
class Lecture {
  public static void main(String arg[]){
    char c;

try {
    char c;
    System.out.print("Saisie :");
    while((c = (char) System.in.read()) != -1) {
        System.out.println("c="+c);
    }
} catch(IOException e) {
    e.printStackTrace();
}
try {
        Reader in = new InputStreamReader(System.in);
        c=(char)in.read();
        System.out.println(" c= "+c);
    }
    catch (IOException e) {
        System.out.println(e.toString());
    }
  }//fin main
}// fin classe
```

```
/*** exemple d'execution ***
Saisie :abcdef
 c= a
 c= b
```

```
import java.io.*;
class Lecture2 {
 public static void main(String arg[]){
   char buf[]=new char[10];
   try {
        System.out.print("Saisie :");
        Reader in = new InputStreamReader(System.in);
        in.read(buf,0,5);
        String s = new String(buf);
        System.out.println("chaine lue :"+s);
   }
   catch (IOException e) {
        System.out.println(e.toString());
   }
 }
}
```

```
/*** exemple d'execution ***
Saisie : abcdefghijk
chaine lue :abcde
```

➔ Un objet **InputStreamReader** et **OutputStreamWriter** sont des ponts entre les flux d'entrée d'octets et ceux de caractères.

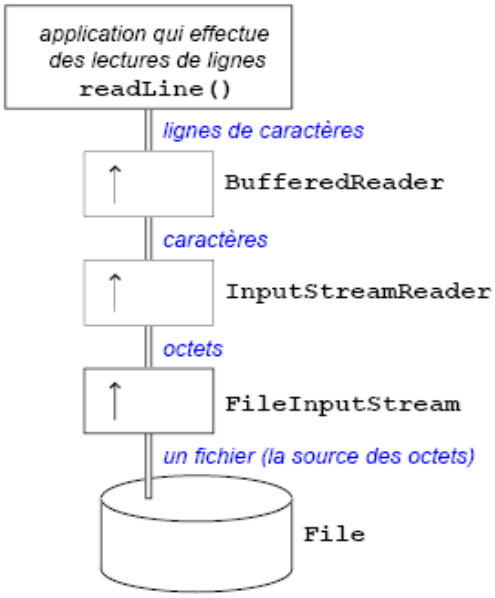
Il est préférable de ne pas directement utiliser la classe **InputStream,** mais plutôt la classe **BufferedReader.**

➔ On génère un objet de type **BufferedReader** à partir de l'objet de type **InputStream(**ex. System.in**)** en faisant au préalable une conversion en passant par un objet de type **InputStreamReader**.

```
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader keyboard = new BufferedReader(isr);

System.out.print("Entrez une ligne de texte : ");
String line = keyboard.readLine();
System.out.println("Vous avez saisi : " + line);
```

Cet exemple permet facilement de récupérer une chaîne de caractère saisie sur la console par l'utilisateur.



Ce qui, dans le code, donne :

```
File f = new File("mydata.txt");
FileInputStream fis =
            new FileInputStream(f);
InputStreamReader isr =
            new InputStreamReader(fis);
BufferedReader br =
            new BufferedReader(isr);
```

ou, en version compacte :

```
BufferedReader br =
    new BufferedReader(
        new InputStreamReader(
            new FileInputStream(
                new File(
                    "mydata.txt"))));
```

```
import java.io.* ;
/*
*import java.io.* ; importe du code source préprogrammé et précompilé,
* contenu dans un paquetage (ou bibliothèque). Plus précisément,
* ici on importe toutes les classes (.*) du paquetage java.io
*/

public class LirePrenom {
  public static String lireChaine () throws IOException {
    // définir une table d'octets pour stocker au max. 15 caractères
    byte[] lu = new byte[15];
    // attendre la frappe des caractères au clavier
    // et les stocker dans la table
    System.in.read(lu,0,15);
    // utilisation d'un constructeur de String
    // pour stocker dans la chaine texte les
    // caractères lues au clavier
    String texte = new String (lu,0,lu.length);
    // renvoyer la chaine sans contrôle
    return texte;
 }
 public static void main (String args[]) throws IOException{
   System.out.print("Quel est ton prenom ? ");
   String prenom = lireChaine();
   System.out.println("Bonjour, "+ prenom);
 }
}

/*** exemple d'execution ***
Quel est ton prenom ? toto
Bonjour, toto
```

```
/*************   Classe Scanner   ********/
//Ceci importe la classe Scanner du package java.util version 1.5
import java.util.Scanner;
//Ceci importe toutes les classes du package java.util
import java.util.*;

public  class essai {
     public static void main (String[] args){
              Scanner  sc = new Scanner(System.in) ;
              System.out.println(« Entrer un mot ») ;
              String str =sc.nextLine() ;
              System.out.println(« Vous avez saisi le mot: » + str) ;
              char caractere = str.charAt(0);
           /*
             System.out.println(sc.next());
             int i = sc.nextInt();
             double d = sc.nextDouble();
             long l = sc.nextLong();
             byte b = sc.nextByte();
             sc.useDelimiter("\\s*fish\\s*");
            */
             sc.close();

          /*
             Scanner fichier = new Scanner (new File(« teste.txt »)) ;
          */
    }
}


/*************   classe Console    java 1.6  ***************/
```

La class console dans le paquetage java.io.Console

La classe System possède une méthode console() qui permet d'obtenir une instance de la classe Console.

La méthode printf() permet de formater et d'afficher des données.

La méthode readLine() permet la saisie d'une ligne de données dont les caractères sont affichés sur la console.

La méthode readPassword() est identique à la méthode readLine() mais les caractères saisis ne sont pas affichés

```
public class TestConsole {

        public static void main(String args[]) {
            String string = "La façade nécessaire";
            System.out.println(string);
            System.console().printf("%s%n", string);
    }

}
```

```
La faþade nŰcessaire
```

La façade nécessaire

```
import java.io.Console;
import java.util.Arrays;
 public class test3 {
   public static void main(String[] args) {
    // Get a console object //
    Console console = System.console();
    // Read username from the console //
    String username = console.readLine("Username: ");
    // Read password, the password will not be echoed to the console screen
    // and returned as an array of characters. //
    char[] password = console.readPassword("Password: ");
    if (username.equals("admin") &&
                          String.valueOf(password).equals("secret")) {
       console.printf("Welcome to Java Application %1$s.\n", username);
       // // Clear the password after validation successful //
       Arrays.fill(password, ' ');
    }
    else {
       console.printf("Invalid username or password.\n");
    }
  }
}

 /****ou bien */
 Console con = System.console();
 if (con != null) {  Scanner sc = new Scanner(con.reader()); }
```

2. Classes de flux de données

Le tableau suivant vous montre quelques classes proposées dans le package **java.io**

	flux texte (lecture) Caractère 16 bits	flux texte (ecriture) Caractère 16 bits	flux binaire (lecture) Octet 8 bits	flux binaire (ecriture Octet 8 bits
classe Abstraite de base	**Reader**	**Writer**	**InputStream**	**OutputStream**
implementation	InputStreamReader	OutputStreamWriter	DataInputStream	DataOutputStream
fichier	FileReader	FileWriter	FileInputStream	FileOutputStream
E/S bufferisée	BufferedReader	BufferedWriter	BufferedInputStream	BufferedOutputStream

Les classes **Reader** ou de **Writer** permettent de réaliser des transformations de code d'un système **ASCII** dérivé vers **Unicode** et réciproquement.

3. Sérialisation

- Des objets de java.io prennent en charge la sauvegarde et la restauration des objets dans des fichiers (binaires)
- C'est un service important car les objets sont souvent complexes : un objet ayant pour membre d'autre objets devient vite un graphe cyclique
- Il s'agit de transformer un graphe complexe en une série d'octets «(à plat), d'où le nom de sérialisation (écriture) et dé-sérialisation (lecture)

4. Manipulation de fichiers indépendamment des données qu'ils contiennent :

➔ La classe **java.io.FILE** vous permet de manipuler le système de fichiers. Vous pouvez ainsi créer, supprimer, déplacer ou obtenir des informations aussi bien sur des fichiers que sur des dossiers

Le tableau suivant présente certaines méthodes couramment utilisées

String getName();	Retourne le nom du fichier.
String getPath();	Retourne la localisation du fichier en relatif.
String getAbsolutePath();	Idem mais en absolu.
String getParent();	Retourne le nom du répertoire parent.
boolean renameTo(File newFile);	Permet de renommer un fichier.
boolean exists() ;	Est-ce que le fichier existe ?
boolean canRead();	Le fichier est t-il lisible ?
boolean canWrite();	Le fichier est t-il modifiable ?
boolean isDirectory();	Permet de savoir si c'est un répertoire.
boolean isFile();	Permet de savoir si c'est un fichier.
long length();	Quelle est sa longueur (en octets) ?
boolean delete();	Permet d'effacer le fichier.
boolean mkdir();	Permet de créer un répertoire.
String[] list();	On demande la liste des fichiers localisés dans le répertoire.

Série d'exercices corrigés

Exercice 1: copie de fichier en Java

Ecrire un programme en java qui permet de réaliser une copie de fichier. Pour ce faire, votre programme attend que le nom du fichier source et le nom du fichier de destination soient renseignés sur la ligne de commande servant à lancer le programme. A titre indicatif, voici un exemple de commande servant à démarrer le programme en sachant que si le nombre de paramètres n'est pas correct, le programme vous informe par un message console de son bon usage.

> java Copy sourceFile.txt destFile.txt

Corrigé exercice 1:

```
import java.io.*;

public class Copy {

    public static void main (String[] argv)   {
        // Test sur le nombre de paramètres passés
        if (argv.length != 2) {
            System.out.println("Usage> java Copy sourceFile destinationFile");
            System.exit(0);
        }

        try {
            // Préparation du flux d'entrée
            File sourceFile = new File(argv[0]);
            FileInputStream fis = new FileInputStream(sourceFile);
            BufferedInputStream bis = new BufferedInputStream(fis);
            long l = sourceFile.length();

            // Préparation du flux de sortie
            FileOutputStream fos = new FileOutputStream(argv[1]);
             BufferedOutputStream bos = new BufferedOutputStream(fos);

            // Copie des octets du flux d'entrée vers le flux de sortie
            for(long i=0;i<l;i++) {
                bos.write(bis.read());
            }

            // Fermeture des flux de données
            bos.flush();
            bos.close();
            bis.close();

        } catch (Exception e) {
            System.err.println("File access error !");
            e.printStackTrace();
        }

        System.out.println("Copie terminée");
    }
}
```

Exercice 2 : copie d'objet en Java

Ecrire 2 programmes **TestSerialisable.java** et **TestDeserialisable**, le premier pour sauvegarder la classe famille dans un fichier **famille.dat**, et le second pour l'afficher**.**

Fichier Individu.java

```
public class Individu {
    String prenom;
    Famille famille;
    public Individu(String p, Famille f) {
        prenom = p;
        famille = f;
    }
    public String toString() {
        return prenom + " " + famille.nom;
    }
}
```

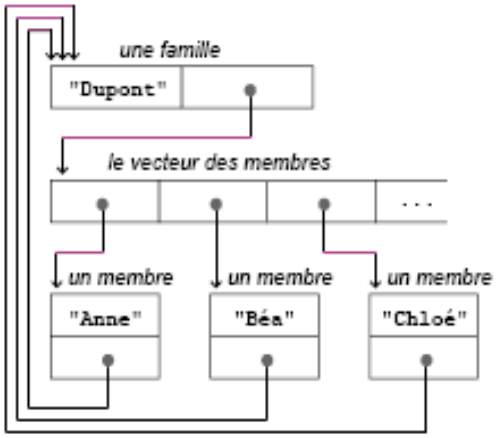
Fichier TestSerialisation.java

```
public class TestFamille {
    public static void main(String[] args) {
        Famille uneFamille =
                        new Famille("Dupont");
        uneFamille.ajouterMembre("Anne");
        uneFamille.ajouterMembre("Béa");
        uneFamille.ajouterMembre("Chloé");

        System.out.println(uneFamille);
    }
}
```

Fichier Famille.java

```
public class Famille {
    String nom;
    List membres = new Vector();
    Famille(String n) {
        nom = n;
    }
    void ajouterMembre(String prenom) {
        membres.add(new Individu(prenom, this));
    }
    public String toString() {
        return membres.toString();
    }
}
```



Corrigé exercice 2 :

Fichier Famille.java

import java.io.Serializable;

public class Famille **implements Serializable**{
private static final long serialVersionUID = 1L ;
}

Fichier Individu.java

import java.io.Serializable;

public class Individu **implements Serializable**{
private static final long serialVersionUID = 2L ;

}

Fichier TestSerialisable.java

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class TestSerialisation {
  public static void main(String[] args) throws IOException, FileNotFoundException{
     Famille uneFamille = new Famille(“Dupont”);
     uneFamille.ajouterMembre(“Anne”) ;
     uneFamille.ajouterMembre(“Béa”) ;
     uneFamille.ajouterMembre(“Chloé”) ;
     File f = new File(“famille.dat”) ;
     FileOutputStream fos = new FileOutputStream(f);
     ObjectOutputStream oos = new ObjectOutputStream(fos);
     oos.writeOject(uneFamille)
     fos.close();
  }
}
```

Fichier TestDeserialisable.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.ObjectInputStream;

public class TestDeserialisation {
  public static void main(String[] args) throws IOException, FileNotFoundException{
     Famille uneFamille;
     File f = new File(“famille.dat”);
     FileInputStream fis = new FileInputStream(f);
     ObjectInputStream ois = new ObjectInputStream(fis);
      uneFamille = (Famille) ois.readObject();
     Sytem.out.println(uneFamille);
     fis.close();
  }
}
```