

# Fichiers en Java

## Les entrées/sorties Java

Athanasia Katsouraki


02/11/2017

# Introduction

Lors de la conception d'un logiciel, il n'est pas rare de vouloir lire ou écrire dans un fichier (par ex., pour sauvegarder des données)

A chaque fois, il nous faudra utiliser des flots d'E/S (I/O).

# Les flots

- Tous les langages de programmation possèdent un mécanisme permettant l'échange de donnée entre la mémoire vive (ram) et la mémoire secondaire (disque).
- Java  propose un mécanisme basé sur les flots.
- Les flots permettent les échanges entre la mémoire et des entités très différentes comme la mémoire secondaire et la mémoire d'un ordinateur distant.

# Les flots

- Java fait la distinction entre les flots d'entrée et les flots de sortie.
- Les flots d'entrée permettent la lecture de données depuis une entité externe au programme (les entités sources).
- Les flots de sortie permettent l'écriture de données vers une entité externe (les entités destination).
- Les flots sont caractérisés par la nature des informations échangées. On distingue :
  - Les flots d'octets : l'information est codée sur 8 bits, c'est-à-dire avec un type **byte**.
  - Les flots de caractères : l'information est codée avec une type **char** du jeu de caractères UNICODE.

# Construire les flots Java

Les flots construits dérivent tous de quatre classes :

	<i>Nom du flot d'entrée</i>	<i>Nom du flot de sortie</i>
<i>Flots d'octets</i>	<i>InputStream</i>	<i>OutputStream</i>
<i>Flots de caractères</i>	<i>Reader</i>	<i>Writer</i>

# Construire les flots Java

Les noms des classes qui représentent les entités sources ou destination sont formés en faisant précéder le type de flot par le type de l'entité.

	<i>Entrée</i>	<i>Sortie</i>
<i>Flots d'octets</i>	<i>FileInputStream</i>	<i>FileOutputStream</i>
	<i>ByteArrayInputStream</i>	<i>ByteArrayOutputStream</i>
	<i>PipedInputStream</i>	<i>PipedOutputStream</i>
<i>Flots de caractères</i>	<i>FileReader</i>	<i>FileOutputStream</i>
	<i>CharArrayReader</i>	<i>ByteArrayOutputStream</i>
	<i>StringReader</i>	<i>StringWriter</i>
	<i>PipedReader</i>	<i>PipedWriter</i>

# Construire les flots Java

Les classe peuvent être composées avec d'autre pour apporter de nouvelles fonctionnalités au flot initial. Les règles de formation des noms restent la même.

Le mot *Buffered* permettent des E/S en accumulant les données dans un tampon (buffer) avant de les envoyer ou de les retourner augmentant l'efficacité des opérations.

<i>Entrée</i>	<i>Sortie</i>
<i>BufferedReader</i>	<i>BufferedWriter</i>
<i>LineNumberReader</i>	
<i>PushBackReader</i>	

# Exemple

L'exemple suivant lit une ligne d'un fichier texte sur un site distant.

```
public class URLExample {  
  
    public static void main (String[] args){  
  
        try {  
            URL webURL= new URL ("http://www.sample-videos.com/text/Sample-text-file-10kb.txt");  
            BufferedReader line = new BufferedReader (new InputStreamReader(webURL.openStream()));  
            // Flot de traitement pour les caractères ( buffer ).  
            // Ce flot est chaîné au FileReader  
            String ligne =line.readLine() ; // contiendra chaque ligne  
            System.out.println(ligne);  
        } catch (MalformedURLException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        } catch (IOException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        }  
    }  
}
```



# Les flots predefinis

Java propose des flots prédéfinis dans le package java.io : **in** et **out**

- le flot standard **in** relie le clavier à la mémoire vive (programme). Exemple :  
`char c = System.in.read();`
- le flot standard **out** relie la mémoire vive à l'écran. Exemple  
`System.out.println("Exemple");`

# Les fichiers

Les fichiers utilisent les flots et sont caractérisés par la nature des informations échangées. On distingue :

- Les fichiers d'octets (mot de 8 bits), c'est-à-dire de **bytes**,
  - Les fichiers de caractères UNICODE,
  - Les fichiers composé de lignes de caractères
- 
- Les fichiers d'objets.

# Instancier des objets fichiers

Les classes qui permettent de manipuler les fichiers possèdent des constructeurs auxquels on spécifie :  
le nom du fichier à ouvrir,

- donné au constructeur sous la forme d'un objet *String*, *File* ou *FileDescriptor*.

le sens d'échange de donnée (lecture ou écriture),

- Une ouverture en mode lecture qui positionne la lecture en début de fichier. Celui-ci doit exister et être accessible, toute erreur déclenche une exception *FileNotFoundException* ou *SecurityException*.
- Une ouverture en mode **écriture** sur un fichier existant. Ce mode vide ce dernier sauf si on précise que l'ouverture se fait en mode ajout. Si le fichier n'existe pas, il est créé, si les droits l'autorisent. Dans le cas contraire une exception *SecurityException* est émise.

le mode d'ouverture (création, ajout, ...)

# Chemin des fichiers

Avec UNIX le chemin d'un fichier est spécifié par des *String* séparés par '/'

Avec Windows le chemin est spécifié par des *String* séparés par '\\'. Comme ce caractère est un méta caractère, il faut le doubler.

```
// UNIX : ouverture d'un fichier d'octets
FileInputStream f = new FileInputStream("java/exemple.class");

// Windows : ouverture d'un fichier de caractères
FileReader fr = new FileReader("c:\\java\\exemple.java");

// Windows : ouverture d'un fichier de caractères chemin UNC
FileReader fr2 = new FileReader("\\\\machine\\java\\exemple.java");
```

# Fichiers d'octets

Les fichiers d'octets sont des instances des classes *FileInputStream* et *FileOutputStream*.

Les opérations élémentaires sont la lecture et l'écriture séquentielle

Entrée	Sortie
<code>int read()</code>	<code>void write()</code>
<code>int read(byte[] b)</code>	<code>void write(byte[] b)</code>

Lorsque la fin du fichier est atteint, *read* retourne -1. Exemple :

```
FileInputStream is = new FileInputStream("src\\Flots.java");
FileOutputStream os = new FileOutputStream("Flots.txt");
byte b;
while ((b = (byte) is.read()) != -1)
    os.write(b);
is.close();
os.close();
```

# Fichiers de characters

Les fichiers de caractères sont des instances des classes *FileReader* et *FileWriter*.  
Les opérations élémentaires sont la lecture et l'écriture séquentielle héritée des classes *FileInputStream* et *FileOutputStream*

```
java.lang.Object
└ java.io.Reader
    └ java.io.InputStreamReader
        └ java.io.FileReader

public static void main(String[] args) throws IOException
{
    FileReader is = new FileReader(".\\src\\Flots3.java");
    FileWriter os = new FileWriter("Flots3.txt");
    int b;
    while ((b = (int) is.read()) != -1)
        os.write(b);
    is.close();
    os.close();
}
```



# Lecture de fichier de characters ligne/ligne

Les fichiers de chaines de caractères sont des instances des classes *BufferedReader* et *BufferedWriter* construit à partir d'instances *FileReader* et *FileWriter*.

Les opérations élémentaires sont la lecture et l'écriture séquentielle héritée des classes *FileInputStream* et *FileOutputStream*

```
java.lang.Object  
└ java.io.Reader  
    └ java.io.InputStreamReader  
        └ java.io.FileReader
```

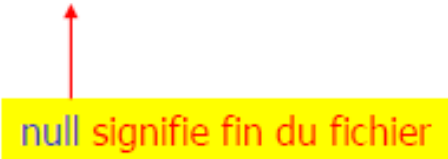
# Exemple

La lecture du fichier ligne par ligne se fait avec la fonction *readLine* proposée dans l'objet *BufferedReader*

Les opérations E/S sont susceptibles de provoquer de erreurs. Java impose d'intercepter les exceptions : *readLine* comporte une clause *throws*

```
public static void main(String[] args) throws IOException
{
    BufferedReader line_in = new BufferedReader(
        new FileReader(
            new File(".\\src\\Flots5.java")));
    BufferedWriter line_out= new BufferedWriter(
        new FileWriter(
            new File("Flots5.java")));

    String ligne;
    while ((ligne = line_in.readLine()) != null)
        line_out.write(ligne + "\n");
    line_out.close();
    line_in.close();
}
```





# Les principales methodes disponibles

int	<a href="#"><u>read</u></a> (byte[] b) Reads some number of bytes from the contained input stream and stores them into the buffer array b.
int	<a href="#"><u>read</u></a> (byte[] b, int off, int len) Reads up to len bytes of data from the contained input stream into an array of bytes.
boolean	<a href="#"><u>readBoolean</u></a> () See the general contract of the readBoolean method of DataInput.
byte	<a href="#"><u>readByte</u></a> () See the general contract of the readByte method of DataInput.
char	<a href="#"><u>readChar</u></a> () See the general contract of the readChar method of DataInput.
double	<a href="#"><u>readDouble</u></a> () See the general contract of the readDouble method of DataInput.
int	<a href="#"><u>readInt</u></a> () See the general contract of the readInt method of DataInput.
long	<a href="#"><u>readLong</u></a> () See the general contract of the readLong method of DataInput.
short	<a href="#"><u>readShort</u></a> () See the general contract of the readShort method of DataInput.

# Exemple 1

```
// Ecrit un tableau de 4 entiers
int[] t = { 1, 2, 3, 4 };
// Ecrit le tableau
DataOutputStream os = new DataOutputStream(
    new FileOutputStream("data.bin"));
for (int j=0; j < t.length; j++)
    os.writeInt(t[j]);
os.close();
```

# Exemple 2

```
// La fin du fichier est détectée par l'exception EOFException.
DataInputStream is = new DataInputStream(
    new FileInputStream("data.bin"));

int b;
try
{
    for (;;) // La fin du fichier déclenche l'exception EOFException
    {
        b = is.readInt();
        System.out.println(b);
    }
}
catch (EOFException e)
{
    System.out.println("Fin de fichier");
    is.close();
}
catch (Exception e)
{
    System.out.println("Autre exception");
}
```

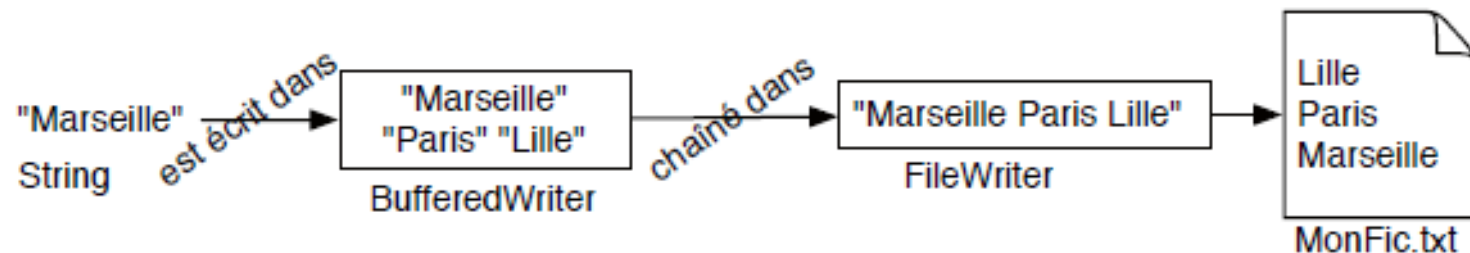
# Exemple 3 – Lire du text dans un fichier

```
import java.io.*; // A ne pas oublier

class LireFichier {
    public static void main ( String[] args) {
        try {
            File f = new File("MonFic.txt");
            FileReader fr = new FileReader(f);
            // Flot de communication pour les caractères,
            // qui se connecte à un fichier
            BufferedReader br = new BufferedReader(fr);
            // Flot de traitement pour les caractères (buffer).
            // Ce flot est chaîné au FileReader
            String ligne = null; // contiendra chaque ligne
            while ((ligne = br.readLine()) != null) {
                System.out.println (ligne);
            }
            br.close();
        } catch (IOException ex) {ex.printStackTrace();}
    }
}
```

# Les buffers

Les buffers fournissent un contenant temporaire pour grouper les données jusqu'à ce qu'il soit plein. De cette façon, vous pouvez limiter le nombre d'accès au disque dur (qui prennent beaucoup de temps).



Exemple :

```
FileWriter fw = new FileWriter("monFic.txt");  
BufferedWriter bw = new BufferedWriter(fw);
```

Dans cet exemple les données seront écrites sur le disque lorsque le buffer sera plein. En effet, ce dernier transfèrera alors toutes ses données au FileWriter qui écrira toutes les données sur le disque en une seule fois.



Si vous voulez envoyer les données avant que le buffer soit plein, il suffit de le vider par un appel de `bw.flush()`.

# Exemple d'utilisation de File

Créer un objet représentant un fichier existant :

`File f = new File("MonFic.txt");` → Cette classe représente un fichier sur le disque,

Créer un répertoire :

```
File rep = new File("MonRep");  
rep.mkdir();
```

Lister le contenu d'un répertoire :

```
if (rep.isDirectory()) {  
    String[] contenuRep = rep.list();  
    for (int i = 0; i < contenuRep.length; i++) {  
        System.out.println (contenuRep[i]);  
    }  
}
```

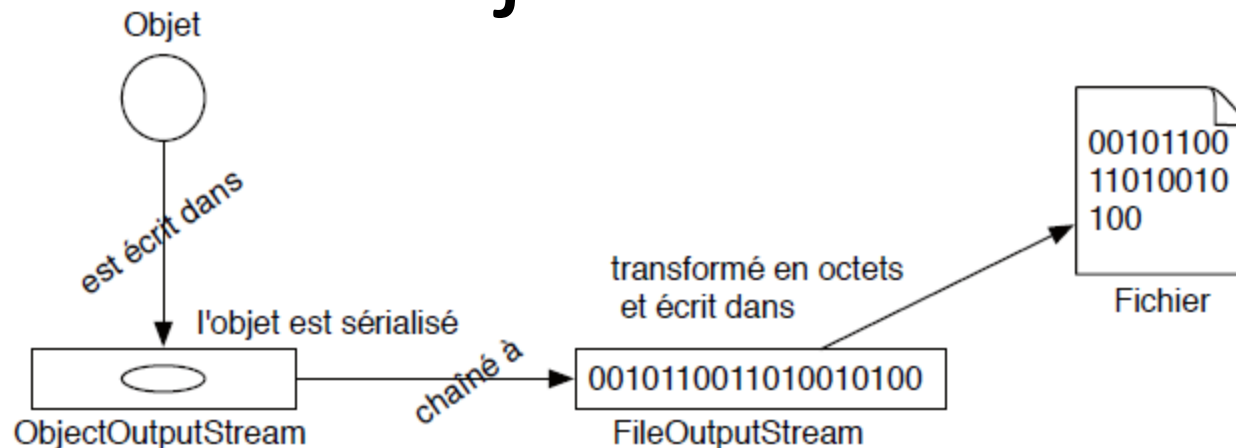
Obtenir le chemin absolu d'un fichier ou d'un répertoire :

```
System.out.println(rep.getAbsolutePath());
```

Supprimer un fichier ou un répertoire :

```
boolean supprime = f.delete();
```

# Ecrire un objet dans un fichier



- 1 Créer un `FileOutputStream`. Cet objet sait comment se connecter à un fichier et même en créer un.

```
FileOutputStream fos = new FileOutputStream("MonZoo.ser");
```

- 2 Créer un `ObjectOutputStream`. Cet objet permet d'écrire des objets, mais il ne peut pas se connecter directement à un fichier. Il a besoin pour cela d'un « intermédiaire », ici un `FileOutputStream`. On va alors `chaîner` les deux flots, le `ObjectOutputStream` au `FileOutputStream`.

```
ObjectOutputStream oos = new ObjectOutputStream(fos);
```

- 3 Écrire les objets à sauvegarder dans « MonZoo.ser ».

```
oos.writeObject(monChien);
oos.writeObject(monCanard);
```

- 4 Fermer `ObjectOutputStream`. La fermeture du premier flot entraîne automatiquement celle des autres.

```
oos.close();
```

# Ecrire du texte dans un fichier

Ecrire du texte (des données de type String) dans un fichier.

Pour réaliser cela, il suffit d'utiliser un `FileWriter` plutôt qu'un `FileOutputStream`. Puisque les chaînes de caractères vont être écrites tel quel, il n'est pas nécessaire d'utiliser un flot de traitement comme `ObjectOutputStream`.

```
import java.io.*; // ce package contient FileWriter

class EcrireTexte {
    public static void main (String[] args) {
        try { // tout le code E/S doit être dans un try/catch
            FileWriter fw = new FileWriter("MonFic.txt");
            // si ce fichier n'existe pas, il sera créé
            fw.write("bonjour");
            fw.close(); // Ferme le flot
        }
        catch (IOException ex) {ex.printStackTrace();}
    }
}
```