



## Chapitre 5 : IO



# Le système d'E/S Java



Java

## Sommaire



- **Présentation de Java**
- **La Grammaire**
  - Types
  - Opérateurs et Structures de contrôles
  - Tableaux
  - Exceptions
- **Les Objets dans Java**
  - Classes, attributs et méthodes
  - Le polymorphisme et Java
  - Paquetages
- **Les Classes de bases**
  - Chaînes de caractères
  - Outils mathématiques
  - Dates
  - Vecteur et tables de hachages
- **Conventions de codage Java**
- **AWT : Abstract Window Toolkit**
  - Applet et application
  - Frame, Panel et Layout
  - Composants GUI
  - Gestion des événements
- **Le système d'E/S Java**
- **Les Threads**
  - La Classe Thread et l'interface Runnable
  - Contrôler des threads : synchronisation et ordonnancement
- **La programmation réseau**
  - Les URL
  - Les sockets
  - Java RMI



## I/O : classes de lecture et d'écriture de données sur des supports physiques de données



Java classes	Commentaire
<code>Reader</code>	Classe générique de lecture
<code>Writer</code>	Classe générique d'écriture
<code>FileReader</code>	Pour la lecture fichier
<code>FileOutputStream</code>	<code>FileWriter</code> Pour l'écriture fichier
<code>StringReader</code>	Pour la lecture et écriture de
<code>StringWriter</code>	chaînes de caractères sur fichier
<code>CharArrayReader</code>	Permet a un buffer memoire d'etre
<code>CharArrayWriter</code>	utilisé en input ou output Stream
<code>PipedReader</code>	Utilisé comme source ou sortie
<code>PipedWriter</code>	de donnée en multithreading

© Rémy Courdier

1998-2007 - Java I/O- Version du cours 1.5

3

## I/O : classes logiques de lecture et d'écriture de données



Java classes	Commentaire
<code>FilterReader</code>	classe abstraite pour les autres
<code>FilterWriter</code>	classes de lecture/ecriture
<code>BufferedReader</code>	permet <code>readLine( )</code>
<code>BufferedWriter</code>	evite une ecriture physique à chaque ecriture
<code>DataInputStream</code>	lect. de types primitifs (char,int)
<code>DataOutputStream</code>	ecr. de types primitifs (char,int)
<code>PrintWriter</code>	Pour produire une sortie formatée
	Classe très couramment utilisée
<code>LineNumberReader</code>	garde la trace du numero de
	ligne du fichier ( <code>getLineNumber()</code> )
<code>StreamTokenizer</code>	n'hérite pas de <code>InputStream</code> ou
	<code>OutputStream</code> mais travail sur
<code>PushBackReader</code>	gestion d'un buffer contenat le
	dernier carcatère lu
<code>SequenceInputStream</code>	
<code>RandomAccessFile</code>	permet de ce déplacer d'un record à
	l'autre ( <code>seek()</code> , <code>length()</code> , ...)

© Rémy Courdier

1998-2007 - Java I/O- Version du cours 1.5

4



## I/O : Lecture de fichier simple & lecture de flots standards



- \* **System.in**  
 approche UNIX avec :  
   "std input": System.in  
 System.in est de type  
 DataInputStream  
 Un BufferedReader a besoin  
 d'un argument de type Reader  
 c'est pourquoi on passe par un  
 InputStreamReader
- \* **BufferedReader**  
 Pour augmenter les  
 performances de lecture on  
 passe le fichier au  
 constructeur d'une classe qui  
 utilise un buffer de lecture  
 physique

```
import java.io.*;
public class stdin {
    public static void main(String args[]) {
        try {
            // 1a. lecture de lignes en entree:
            BufferedReader in =
                new BufferedReader(
                    new FileReader(args[0]));
            String s, cchar = new String();
            while((s = in.readLine()) != null)
                cchar += s + "\n";
            in.close();
            // 1b. lecture du flot d'entree standard:
            BufferedReader stdin =
                new BufferedReader(
                    new InputStreamReader(System.in));
            System.out.print("Entrer une ligne:");
            System.out.println(stdin.readLine());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

© Rémy Courdier

1998-2007 - Java I/O- Version du cours 1.5

5

## I/O : écriture de fichier simple & lecture de flots standards



- \* **System.out & System.err**  
 approche UNIX avec :  
   "std output ":  
   System.out  
   "error output":  
   System.err
- \* Redirection par l'appel à des  
 méthodes de classes de la  
 classe System  
   setIn(InputStream)  
   setOut(PrintStream)  
   setErr(PrintStream)

```
import java.io.*;
class Redirection {
    public static void main(String args[]) {
        try {
            BufferedInputStream in =
                new BufferedInputStream(
                    new FileInputStream(
                        "Redirection.java"));
            PrintStream err =
                new PrintStream(
                    new BufferedOutputStream(
                        new FileOutputStream("err.out")));
            System.setIn(in);
            System.setErr(err);
            BufferedReader br =
                new BufferedReader(
                    new InputStreamReader(System.in));
            while((String s = br.readLine()) != null) {
                System.out.println(s);
                System.err.println(s);
            }
            err.close(); // en securite
        } catch (IOException e) {e.printStackTrace();}
    }
}
```

© Rémy Courdier

1998-2007 - Java I/O- Version du cours 1.5

6



## I/O : Manipulation de fichiers et de répertoires

### La classe File



#### Attention !!!

La classe File ne représente pas un pointeur sur le fichier mais seulement le nom d'un fichier ou d'un ensemble de fichiers dans un répertoire

```
import java.io.*;
/**
 * affiche la liste des fichier
 * d'un repertoire
 */
public class FileListe {
    public static void main(String args[] ) {
        try {
            File path = new File(".");
            String[] list = path.list();
            for(int i = 0; i < list.length; i++)
                System.out.println(list[i]);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

## I/O : Manipulation de fichiers

### Création, renommage et suppression de fichiers



```
import java.io.*;
public class FileExemple {
    private static void infoFic (File f){
        System.out.println(
            "path: " + f.getAbsolutePath() +
            "\n read: " + f.canRead() +
            "\n write: " + f.canWrite() +
            "\n Name: " + f.getName() +
            "\n Parent: " + f.getParent() +
            "\n Path: " + f.getPath() +
            "\n length: " + f.length() +
            "\n Modif.: " + f.lastModified());
        if(f.isFile())
            System.out.println("fichier");
        else if(f.isDirectory())
            System.out.println("repertoire");
    }
    ...
}
```

```
...
public static void main(String args[]){
    if(args[0].equals("-r")) {
        if(args.length!=3) System.exit(1);
        File
            ancien = new File(args[1]),
            nouveau = new File(args[2]);
        ancien.renameTo(nouveau);
        infoFic(ancien );
        infoFic(nouveau);
    } else {
        if(args.length!=2) System.exit(1);
        File f = new File(args[1]);
        if(args[0].equals("-d"))
            if(f.exists()) f.delete();
        if(args[0].equals("-c")) {
            if(f.exists()) System.exit(1);
            f.mkdirs();
        }
        infoFic(f);
    }
}
```



## I/O : La compression de fichiers

✚ Java fournit les algorithmes des 2 techniques de compression communément utilisées :

- Zip
- GZIP

```
import java.io.*;
import java.util.zip.*;
public class FichierGzip {
    public static void main(String args[]) {
        try {
            BufferedReader in =
                new BufferedReader(
                    new FileReader(args[0]));
            BufferedOutputStream out =
                new BufferedOutputStream(
                    new GZIPOutputStream(
                        new FileOutputStream("fic.gz")));
            System.out.println("écriture du fichier");
            while((int c = in.read()) != -1)
                out.write(c);
            in.close(); out.close();
            System.out.println("Reading file");
            BufferedReader inBis =
                new BufferedReader(
                    new InputStreamReader(
                        new GZIPInputStream(
                            new FileInputStream("fic.gz"))));
            while((String s = inBis.readLine()) != null)
                System.out.println(s);
        } catch(Exception e) { e.printStackTrace();}
    }
}
```

## I/O: Persistance d'objets les mécanismes de "sérialisation" d'objets

### ✚ la Classe ObjectOutputStream

- classe du package java.io : implémente les méthodes pour "serialiser" des objets, des tableaux, et les types de base dans un output stream
- Principales méthodes :
  - writeObject() // sauve un objet et les objets qu'il référence
  - defaultWriteObject() // utilisé lors des sauvegardes personnalisées

### ✚ la Classe ObjectInputStream

- classe du package java.io : implémente les méthodes pour "déserialiser" des objets, des tableaux, et les types de base dans un output stream
- Principales méthodes :
  - readObject() //lit un objet et les objets qu'il référence
  - defaultReadObject() //utilisé lors des lectures personnalisées

✚ Toutes les méthodes de ces classes peuvent provoquer une IOException



*I/O: Persistence d'objets*  
*Exemple simple de sauvegarde*

```
import java.awt.*;
import java.io.*;
import java.util.*;
public class sauvegarde {
    public static void main(String
    args[]) {
        try {
            Vector v = new Vector();
            v.addElement("sauvegarde");
            v.addElement(new Integer(10));
            v.addElement(new Rect
            (10,10,150,100));
            System.out.println(v)
            FileOutputStream fos =
            new FileOutputStream
            ("sauve.data");
            ObjectOutputStream oos =
            new ObjectOutputStream
```

**Output :**  
 > java sauvegarde  
 > [sauvegarde, 10, java.awt.rectangle  
 x=10,y=10,width=150,height=100]

© Rémy Courdier 1998-2007 - Java I/O- Version du cours 1.5 11

*I/O: Persistence d'objets*  
*Exemple simple de lecture*

```
import java.io.*;
public class lecture {
    public static void main(String args[]) {
        try {
            FileInputStream fis =
            new FileInputStream (arg[0]);
            ObjectInputStream ois =
            new ObjectInputStream (fis);
            Object unObjet = ois.readObject();
            System.out.println (unObjet);
            fis.close(); // securite
        } catch(Exception e) {
            System.out.println ("erreur : + e)
        }
    }
}
```

**Output :**  
 > java lecture sauve.data  
 > [sauvegarde, 10, java.awt.rectangle  
 x=10,y=10,width=150,height=100]

© Rémy Courdier 1998-2007 - Java I/O- Version du cours 1.5 12



## I/O: Persistence d'objets serialisation personnalisées ("custom")



### ✦ le mot clé transient :

- Les variables précédés du mot clés **transient** ne sont pas sauvegardée

```
private transient int compteur;
```

### ✦ Persistence et variables static

- Ces variables partagées entre toutes les instances ne sont pas sauvegardées. il faut les gérer de manière particulière

- Surcharger les méthodes writeObject(...) et readObject(...)

```
private void writeObject(ObjectOutputStream oos) {...  
    oos.defaultWriteObject();  
    if (...) oos.writeBoolean (unBoolStatic);  
    ...} // dans cet exemple les exceptions sont non gérées
```

### ✦ Classe non « sérialisables »

- Certaines classes ne sont pas "sérialisables" et provoque une exception NotSerializableException : ceci indique que l'interface **serializable** n'est pas implémentée dans la classe en question

ex. la classe Thread n'implémente pas cette interface