

Chapitre 7

Les Entrées/Sorties

Programmation Java 2003/2004 - D.Pastre

Chap.7.1

Les Entrées/Sorties

paquetage java.io

classes pour traiter les flux de données

+ une classe pour représenter les fichiers et les répertoires

octets ou *caractères* (Stream ou Reader/Writer)

extraction ou *fonctionnalités* (types, ligne, ...)

tampons (Buffered)

conversion de données (Data)

conversion octets-caractères (InputStream-OutputStream)

Programmation Java 2003/2004 - D.Pastre

Chap.7.2

Classes

Flots d'octets

- **InputStream**
 - **FilterInputStream**
 - **DataInputStream**
 - **ObjectInputStream**
 - **FileInputStream**
- **OutputStream**
 - **FilterOutputStream**
 - **BufferedOutputStream**
 - **PrintStream**
 - **DataOutputStream**
 - **ObjectOutputStream**
 - **FileOutputStream**

Flots de caractères

- Reader**
 - **BufferedReader**
 - **LineNumberReader**
 - **FilterReader**
 - **InputStreamReader**
 - **FileReader**
- **Writer**
 - **BufferedWriter**
 - **FilterWriter**
 - **OutputStreamWriter**
 - **FileWriter**
 - **PrintWriter**

Programmation Java 2003/2004 - D.Pastre

Chap.7.3

classe pour représenter les fichiers et les répertoires

- **File**

constantes de la classe System

static InputStream in	The "standard" input stream
static PrintStream out	The "standard" output stream
static PrintStream err	The "standard" error output stream

interfaces

- **DataInput**, implémentée par **DataInputStream**
 - **ObjectInput**, implémentée par **ObjectInputStream**
- **DataOutput**, implémentée par **DataOutputStream**
- **Serializable**, implémentée par **DataInputStream** et **DataOutputStream**

Programmation Java 2003/2004 - D.Pastre

Chap.7.4

Quelques méthodes

- **print** et **println**(type élémentaire, Object ou String)
*méthode de toutes les classes **Print...***
d'où le `System.out.println` habituel
- **read()** *méthode des classes ...**InputStream**, renvoie un octet*
*et des classes ...**Reader**, renvoie un caractère*
- **write(byte)** *méthode de ...**OutputStream***
- **write(char)** *méthode de ...**Writer***
- **write(String)** *"*
- **write(int)** *méthode des deux types de classes*

- **readLine()** *méthode de **BufferedReader**,*
lit une ligne, la renvoie sous forme de chaîne de caractères
- **close()** *ferme un flot de données (indispensable en écriture)*

lecture caractères lignes clavier

Lecture d'un caractère au clavier

`char c = (char) System.in.read(); // c = le caractère lu`

ou

`int code = System.in.read(); // le code ASCII du caractère lu`

On peut lire une suite de caractères jusqu'à, par exemple, la fin de ligne ou le premier blanc (voir `bonjour.BonjourNom`)

Lecture d'une ligne (chaîne de caractères) au clavier

`BufferedReader entree =`

`new BufferedReader(// ligne de texte, extraction`

`new InputStreamReader(// conversion caractères, extraction`
`System.in)); // instance de InputStream (octets, extraction)`

`String ligne = entree.readLine();`

lecture

lignes
lexèmes

clavier

Avec la classe **StringTokenizer**

*Lecture d'une ligne (chaîne de caractères) au clavier
et extraction de lexèmes*

Les séparateurs sont par défaut sont " ", "\n", "\t", "\r" et "\f"

BufferedReader entree =

new BufferedReader(new InputStreamReader(System.in));

String ligne = entree.readLine();

StringTokenizer st;

while (!(ligne.equals(""))) { // deux "entree" pour s'arrêter

st = new StringTokenizer(ligne);

while (st.**hasMoreTokens**())

System.out.println("lexeme="+st.**nextToken**());

ligne=entree.readLine();

}

Programmation Java 2003/2004 - D.Pastre

Chap.7.7

La classe **StringTokenizer**

sous-classe de **Object** , implements **Enumeration**

Constructeurs

StringTokenizer(String str)

Constructs a string tokenizer for the specified string.

StringTokenizer(String str, String delim)

Constructs a string tokenizer for the specified string. The characters in the delim argument are the delimiters for separating tokens.

Méthodes

boolean **hasMoreTokens**() ou **hasMoreElements**()

Tests if there are more tokens available from this tokenizer's string.

String **nextToken**() ou Object **nextElement**()

Returns the next token from this string tokenizer.

Programmation Java 2003/2004 - D.Pastre

Chap.7.8

lecture	lexèmes	clavier
---------	---------	---------

Avec la classe **StreamTokenizer**

Lecture de lexèmes au clavier

Les séparateurs sont par défaut sont " ", "\n", "\t", "\r" et "\f"

StreamTokenizer entree =

```

    new StreamTokenizer(new InputStreamReader(System.in));
int type = entree.nextToken(); // type du lexème suivant
while (type != StreamTokenizer.TT_EOF) { // fin de fichier
    if (type == StreamTokenizer.TT_NUMBER) // nombre
        System.out.println(entree.nval);
    if (type == StreamTokenizer.TT_WORD) // chaine # nombre
        System.out.println("entree.sval");
    type = entree.nextToken();
}
```

La classe **StreamTokenizer**

Attributs

double **nval**

String **sval**

static int **TT_EOF**

static int **TT_NUMBER**

static int **TT_WORD**

int **ttype**

*sous-classe de **Object***

Constructeur

StreamTokenizer(Reader r)

Create a tokenizer that parses the given character stream.

Méthode

int **nextToken**()

Parses the next token from the input stream of this tokenizer.

lecture écriture	caractères	fichiers textes
---------------------	------------	-----------------

Un caractère

```

FileReader fr = new FileReader ("entree");
char c = (char) fr.read(); // ou int c = fr.read();
System.out.println(c);
fr.close();
FileWriter fw = new FileWriter("sortie");
fw.write(c); fw.close();

```

Un fichier de caractères

```

FileReader fr = new FileReader("entree");  FileWriter fw = new
FileWriter("sortie");
int c; while ((c=fr.read())!=-1) fw.write(c); // ou tout autre travail
fr.close();                                // sur c

```

lecture	caractères	fichiers textes
---------	------------	-----------------

Une ligne de caractères

```

FileReader fr = new FileReader("entree.txt");
int c; while ((c=fr.read())!=10) System.out.print((char)c);
// ou
// char c; while ((c=(char)fr.read())!='\n') System.out.print(c);

```

On peut aussi lire une suite de caractères jusqu'au premier séparateur (blanc, entrée, ...)

lecture	lignes	fichiers textes
---------	--------	-----------------

Lecture de lignes dans un fichier texte

```
BufferedReader br =
    new BufferedReader( // facultatif, pour l'efficacité
        new FileReader("entree")); // caractères, fonctionnalité
String ligne;
while ((ligne = br.readLine()) != null) System.out.println(ligne);
```

lecture	lignes lexèmes	fichiers textes
---------	-------------------	-----------------

Avec la classe **StringTokenizer**

Lecture de lignes dans un fichier texte et extraction de lexèmes

```
BufferedReader br = new BufferedReader(new FileReader("entree"));
StringTokenizer st; String ligne;
while ((ligne = br.readLine()) != null) {
    System.out.println("ligne="+ligne);
    st = new StringTokenizer(ligne);
    while (st.hasMoreTokens())
        System.out.println("lexeme="+st.nextToken());
}
```

Avec la classe **StreamTokenizer**

Lecture de lexèmes dans un fichier texte

```
StreamTokenizer entree =
    new StreamTokenizer(new FileReader("entree"));
int type = entree.nextToken();
while (type != StreamTokenizer.TT_EOF ) {
    System.out.print("type="+type);
    if (type == StreamTokenizer.TT_NUMBER)
        System.out.println(entree.nval);
    if (type == StreamTokenizer.TT_WORD)
        System.out.println(entree.sval);
    type = entree.nextToken();
}
```

*Ecriture dans un fichier texte de n'importe quel type élémentaire
ou object*

```
PrintWriter pw =
    new PrintWriter( // caractères - extraction
        new BufferedWriter( // facultatif, pour l'efficacité
            new FileWriter("sortie"))); // caractères- fonctionnalité
pw.println("aaa");pw.println(3);pw.println(new Double(4));
Vector v = new Vector();v.add("1");v.add("2");pw.println(v);
Rationnel r = new Rationnel(7,5); pw.println(r);
pw.close();
```


écriture
lecture

types élémentaires
chaînes

fichiers binaires

Ecrire et lire dans un fichier binaire

`DataOutputStream dos =`

`new DataOutputStream(// octets, extraction`

`new BufferedOutputStream(// facultatif, pour l'efficacité`

`new FileOutputStream("binaire"))); // octets, fonctionnalité`

`dos.writeInt(1234);dos.writeDouble(576.64);dos.writeUTF("abcde");`

`dos.close();`

`DataInputStream dis = new DataInputStream(`

`new BufferedInputStream(new FileInputStream("binaire")));`

`int i = dis.readInt();`

`double d = dis.readDouble();`

`String s = dis.readUTF();`

`System.out.println("i="+i+" d="+d+" s="+s);`

Programmation Java 2003/2004 - D.Pastre

Chap.7.17

Méthodes

- **readInt()**
- **readDouble()**
- **readChar()**
- ...

*méthodes abstraites de **DataInput**
méthodes de **DataInputStream**
et **ObjectInputStream***

- **readUTF()**, renvoie une String (codage de longueur variable plus compact que le codage habituel sur deux octets)

- **writeInt(int)**
- **writeDouble(double)**
- **writeChar(char)**
- ...
- **writeUTF(String)**

*méthodes abstraites de **DataOutput**
méthodes de **DataOutputStream**
et **ObjectOutputStream***

Programmation Java 2003/2004 - D.Pastre

Chap.7.18

(Sérialisation)

```
ObjectOutputStream oos =
    new ObjectOutputStream(    // octets, extraction
        new FileOutputStream("seria"));    // octets, fonctionnalité
int n = 3;
Double d = new Double(6.6);
Date da = new Date();
Vector v = new Vector();v.add("7");v.add("8");
Rationnel r = new Rationnel(11,7);
int[] tab = {1,2,3,4,5,6};

oos.writeInt(n); oos.writeObject(d);
oos.writeObject(da); oos.writeObject(v);
oos.writeObject(r); oos.writeObject(tab);
```

```
ObjectInputStream ois =
    new ObjectInputStream(new FileInputStream("seria");
int n1 = ois.readInt();
Double d1 = (Double) ois.readObject();
Date da1=(Date) ois.readObject();
Vector v1=(Vector) ois.readObject();
Rationnel r1=(Rationnel) ois.readObject();
int[] tab1= (int[]) ois.readObject();
```

```
System.out.println("\n"+n1+"\n"+d1+"\n"+da1+"\n"+v1);
for (int i=0;i<6;i++) { System.out.print(tab1[i]);}
```

- **writeObject** (Object) *méthode abstraite de **ObjectOutputStream***
*méthode de **ObjectOutputStream***
- **readObject**() *méthode abstraite de **ObjectInputStream***
*méthode de **ObjectInputStream***

Pour pouvoir être "sérialisé", un objet doit être instance d'une classe **implémentant** l'interface **Serializable**, qui ne contient aucun membre et ne sert qu'à identifier les objets "sérialisables".

C'est le cas des classes prédéfinies Double, Date, Vector.
Ce devra le cas de la classe Rationnel dont on modifiera l'en-tête :

```
public class Rationnel implements Serializable
```

La classe **File**

```
String fichier = Clavier.readString();
File f = new File(fichier);
if (f.exists()) {
    String nom = f.getName(); String chemin = f.getPath();
    String cheminAbsolu = f.getAbsolutePath(); // String
    File cheminAbsoluFile = f.getAbsoluteFile(); // File
    boolean bf = f.isFile(); boolean bd = f.isDirectory();
    long longueur = f.length();
    if (f.isDirectory()) {
        String parent = f.getParent();
        File parentFile = f.getParentFile();
        String[] liste = f.list() ; // fichiers et répertoires
        File[] listeF = f.listFiles() ; // fichiers et répertoires (File)
    }
}
```