Les Flots (Java)

Ahcène Bounceur

Utilisation des flots

- Entrées/Sorties
  - Fichiers
  - Clavier/Souris
  - Écran
  - Port série
- Communication Réseaux
  - Filaire
  - Bluetooth
  - Wifi
  - 3G

Généralités sur les flots

- Données lues et écrites à la suite
- Pas d'accès aléatoire (cf. tableaux)
- Lecture dans l'ordre d'écriture
- Ecritures indépendantes des lectures
- Possibilité d'utilisation de tampons

Rappel : octet et caractère

- 1 octet = 8 bits

  00000000

  00110110

  10101100

- 1 caractère = 8 bits

  00000000

  10000010

  10000110

Rappel : octet et caractère

- 1 octet = 8 bits
- Déclaration (java) : **byte b ;**

byte b ;

00000000 →

b = (byte) 0; ou : b = 0x00; ou : b = 0b00000000 ;

00110011 →

b = (byte) 27; ou : b = 0x33; ou : b = 0b00110011 ;

01010110 →

b = (byte) 88; ou : b = 0x56; ou : b = 0b01010110 ;

Les types en Java

Type	Signification	Taille (en octets)	Plage de valeurs acceptées
char	Caractère Unicode	2	'\u0000' ? '\uffff' (0 à 65535)
byte	Entier très court	1	-128 ? +127
short	Entier court	2	-32 768 ? +32 767
int	Entier	4	-2^{31}?$-2{,}147\times10^{9}$? $+2^{31}-1$?$2{,}147\times10^{9}$
long	Entier long	8	-2^{63}?$-9{,}223\times10^{18}$? $+2^{63}-1$?$9{,}223\times10^{18}$
float	Nombre réel simple	4	$\pm2^{-149}$?1.4×10^{-45} ? $\pm2^{128}-2^{104}$?3.4×10^{38}
double	Nombre réel double	8	$\pm2^{-1074}$?$4{,}9\times10^{-324}$? $\pm2^{1024}-2^{971}$? $1{,}8\times10^{308}$
boolean	Valeur logique (booléen)	1	true (vrai), ou false (faux)

Paquetage Java : **java.io**

- Permet de :
  - Manipuler des octets
  - Manipuler des caractères

Paquetage Java : **java.io**

- Exemples :
  - Fichiers
  - Tubes (pipes)

- **Attention** :
  - utilisation systématique des **exceptions** (la reprise sur erreur peut être complexe à mettre en œuvre)

Flots d'octets

- Class **InputStream**
- Class **OutputStream**

- Toutes les classes qui manipulent des flots d'octets héritent de l'une de ces deux classes.

Flots d'octets : en lecture

La classe **InputStream**

Flots d'octets : en lecture

- Lire un octet :
  - La méthode : **int read()**
  - Elle retourne l'octet suivant dans le flot ou -1 si la fin du flot est atteinte.

Flots d'octets : en lecture

- Lire un ou plusieurs octets :
  - La méthode : **int read(byte [] b)**
  - Lit dans le flot au plus **b.length** octets et les place dans le tableau b.
  - Les octets sont stockés dans l'ordre de lecture.
  - Le nombre d'octets effectivement lus est retourné, ou -1 si la la fin du flot est atteinte.

Flots d'octets : en lecture

- Lire un ou plusieurs octets :
  - La méthode : **int read(byte [] b, int offset, int len)**
  - Comme le read précédent mais les octets sont stockés à partir de la case "*offset*" et au plus "*len*" octets sont lus.

Propriétés des méthodes **read**

- **Bloquante** si rien à lire.
  - Tant qu'il n'y a rien à lire dans un flot et que la fin du flot n'est pas atteinte, les méthodes "*read*" sont bloquantes.

- La méthode **available()** peut être utilisée pour éviter les blocages.

Valeur retournée par read

```
try {
   FileOutputStream fos = new FileOutputStream("test.str");
   fos.write(0x01);     // 00000001 = 1
   fos.write(0x0F);     // 00001111 = 15
   fos.write(0x00);     // 00000000 = 0
   fos.write(0x10);     // 00010000 = 16
   fos.close();
}
catch (IOException e) {}
```

Valeur retournée par read

```
int x ;
try {
   FileInputStream fis = new FileInputStream("test.str");
   x = fis.read();
   System.out.println(x) ;
   x = fis.read();
   System.out.println(x) ;
   x = fis.read();
   System.out.println(x) ;
   x = fis.read();
   System.out.println(x) ;
   fis.close();
} catch (IOException e) {}
```

```
1
15
0
16
```

Valeur retournée par read

```
int x ;
byte [] b = new byte[1];
try {
   FileInputStream fis = new FileInputStream("test.str");
   x = fis.read(b);
   System.out.println(x) ;
   x = fis.read(b);
   System.out.println(x) ;
   x = fis.read(b);
   System.out.println(x) ;
   x = fis.read(b);
   System.out.println(x) ;
   fis.close();
} catch (IOException e) {}
```

```
1
1
1
1
```

Valeur retournée par read

```
int x ;
byte [] b = new byte[10];
try {
   FileInputStream fis = new FileInputStream("test.str");
   x = fis.read(b);
   System.out.println(x) ;
   x = fis.read(b);
   System.out.println(x) ;
   x = fis.read(b);
   System.out.println(x) ;
   x = fis.read(b);
   System.out.println(x) ;
   fis.close();
} catch (IOException e) {}
```

```
4
-1
-1
-1
```

Valeur retournée par read

```
int x ;
byte [] b = new byte[10];
try {
   FileInputStream fis = new FileInputStream("test.str");
   x = fis.read(b);
   System.out.println(b[0]) ;
   System.out.println(b[1]) ;
   System.out.println(b[2]) ;
   System.out.println(b[3]) ;
   fis.close();
} catch (IOException e) {}
```

```
1
15
0
16
```

Valeur retournée par read

```
int x ;
byte [] b = new byte[2];
try {
   FileInputStream fis = new FileInputStream("test.str");
   x = fis.read(b);
   System.out.println(b[0]) ;
   System.out.println(b[1]) ;
   x = fis.read(b);
   System.out.println(b[0]) ;
   System.out.println(b[1]) ;
   fis.close();
} catch (IOException e) {}
```

```
1
15
0
16
```

Valeur retournée par read

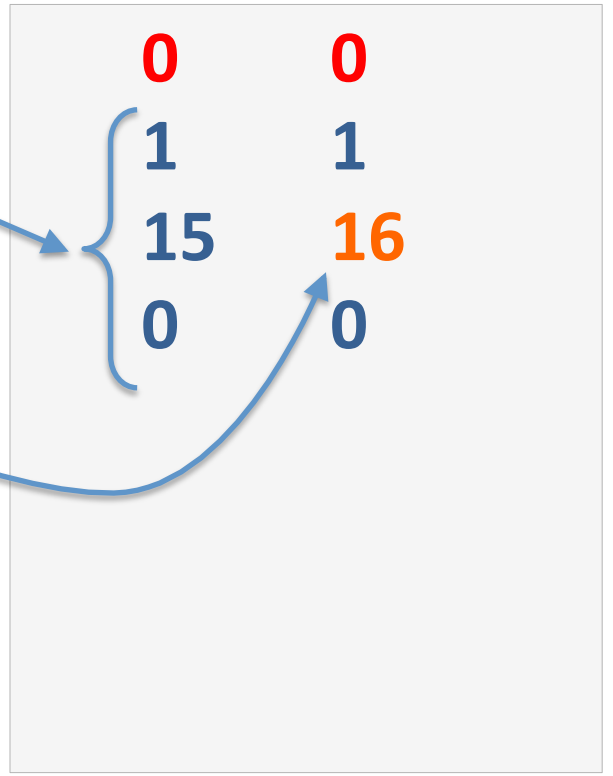

```
int x ;
byte [] b = new byte[10];
try {
   FileInputStream fis = new FileInputStream("test.str");
   x = fis.read(b);
   System.out.println(x) ;
   fis.close();
} catch (IOException e) {}
```

Que signifie la réponse 0 ?

0

Valeur retournée par read

```
int x ;
byte [] b = new byte[4];
try {
   FileInputStream fis = new FileInputStream("test.str");
   x = fis.read(b,1,3);
   System.out.println(b[0]) ;
   System.out.println(b[1]) ;
   System.out.println(b[2]) ;
   System.out.println(b[3]) ;
   x = fis.read(b,2,1);
   System.out.println(b[0]) ;
   System.out.println(b[1]) ;
   System.out.println(b[2]) ;
   System.out.println(b[3]) ;
   fis.close();
} catch (IOException e) {}
```



Nombre d'octets lus

- Les méthodes read sont **Imprévisibles**
  - Les méthodes "read" n'assurent pas la lecture d'un nombre exact d'octets.
  - Elles retournent dès qu'elles peuvent lire au moins un octet.

Exercice de TD

- Ecrivez la méthode permettant de lire exactement n octets.

readFully !

- La méthode **readFully** est implantée dans la classe **DataInput** (même rôle que dans l'exemple précédent).
- Remplissage complet du tableau (buffer).

Modification des comportements des flots

- **long skip(long n)**
- Consomme les n premiers octets d'un flot. Retourne le nombre d'octets consommés.

Marquage des flots

- **mark(int taillemax)**
- **reset()**
- On peut marquer la position courante et revenir ensuite à la position marquée.
- Question: à quoi sert "taillemax" ?

Marquage des flots

- Repositionne la position courante dans le flux de données sur la dernière marque positionnée par la méthode **mark()**.
- Si aucune marque n'a été positionnée ou si la marque est invalide, une exception IOException est déclenchée.

Marquage des flots

- public boolean **markSupported** ()
  - Renvoie true si le flux de données autorise l'utilisation des méthodes **mark()** et **reset()**.

Lecture sur l'entrée standard

- L'entrée standard : **clavier**
- Le flux : **InputStream is = System.in** ;
- Lecture :

  is.read()

  is.read(b)

  is.read(b, i, len)
- Remarque : utilisez le flux de lecture selon le type de données traité.

La méthode flush()

- Les données écrites dans un flot peuvent être stockées dans un tampon (buffer). La méthode flush() permet de vider ce tampon d'écriture vers le médium de sortie.

La méthode close()

- Fermeture du flot
- La méthode close() ferme le flot. Les ressources systèmes sont libérées.
- Attention: fermer les flots autant que possible pour éviter la saturation du système.

Flots d'octets : en écriture

La classe **OutputStream**

Flots d'octets : écriture standard

- System.out
- System.err

Flots d'octets : en écriture

Même fonctionnement que pour read

- void write (int b)
- void write (byte [] b)
- void write (byte [] b, int offs , int len)

Flots de caractères

- Class **Reader**
- Class **Writer**

- Toutes les classes qui manipulent des flots de caractères héritent de l'une de ces classes.

Flots de caractères : en lecture

- int read()
- int read(char [] cbuf)
- int read(char [] cbuf, int off, int len)

- Pour int read() le *caractère lu est stocké dans les deux octets de poids faible* de l'entier.

Flots de caractères : en écriture

- void write(int c)
- void write(char [] cbuf)
- void write(char [] cbuf, int off, int len)
- void write(String str)
- void write(String str, int off, int len)

- Les méthode write acceptent des objets de type String.

Exercice de TD

- Problème d'encodage
  - convertir un flot codé en Cp850 en un flot codé en 8859_1

Liste des encodages:

http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html

Codage des caractères

Canonical Name for java.nio API	**Canonical Name for java.io and java.lang API**	**Description**
US-ASCII	ASCII	American Standard Code for Information Interchange

Codage des caractères

Canonical Name for java.nio API	Canonical Name for java.io and java.lang API	Description
windows-1250	Cp1250	Windows Eastern European
windows-1251	Cp1251	Windows Cyrillic
windows-1252	**Cp1252**	**Windows Latin-1**
windows-1253	Cp1253	Windows Greek
windows-1254	Cp1254	Windows Turkish
windows-1257	Cp1257	Windows Baltic

Fréquent pour les fichiers HTML créés sous Windows

Codage des caractères

Canonical Name for java.nio API	**Canonical Name for java.io and java.lang API**	**Description**
ISO-8859-1	ISO8859_1	ISO 8859-1, Latin Alphabet No. 1

Codage des caractères

- ISO-8859-1 : également connue sous le nom de Latin-1, ce n'est que le premier de 15 codages (de ISO-8859-1 à ISO-8859-15), chacun englobant tout (ou, parfois, partie) des caractères utilisés dans une région ou une forme de langue : Europe de l'Ouest, du Nord, du Sud, et Centrale, Arabe, Grec, Hébreu, Celte, Thaï.
- Notoirement créé rapidement, ISO-8859-1 est très diffusé mais pas idéal - il manque par exemple certaines lettres françaises. Il faut cependant compter avec lui car il s'agit du codage standard d'UNIX et de nombreux logiciels (dont certains navigateurs et logiciels de courrier), et Unicode en est une extension.
- Les normes ISO-8859-xx ne sont pour autant pas complètes (notamment en ce qui concerne l'Extrême-Orient), et il est préférable de nos jours de se référer à Unicode ou UTF-8.

Codage des caractères

Canonical Name for java.nio API	**Canonical Name for java.io and java.lang API**	**Description**
UTF-8	UTF8	Eight-bit UCS Transformation Format

Codage des caractères

- **UTF-8** : un codage tiré de Unicode (UTF signifie Unicode Transformation Format), où au lieu d'avoir un caractère encodé en 2 octets, il l'est de manière variable en UTF-8 (1, 2 voire 3 octets pour les caractères complexes), et permet une sérialisation plus efficace du texte. C'est le codage de base de XML, et donc le plus courant aujourd'hui.

Codage des caractères

Canonical Name	**Description**
Cp037	USA, Canada (Bilingual, French), Netherlands, Portugal, Brazil, Australia

Extended Encoding Set
(contained in lib/charsets.jar)
Supported by java.io and java.lang APIs

Encodage

- Classes :

InputStreamReader

OutputStreamWriter

- Exemples :

```
InputStreamReader isr = new InputStreamReader(is,"encodage")

OutputStreamWriter os< = new OutputStreamWriter(os,"encodage")
```

```
is (InputStream)
os (OutputStream)
```

Caractères ← Octets

- Classes :

`InputStreamReader`

`OutputStreamWriter`

- Exemple

```
BufferedReader br = new InputStreamReader(inputStream)
```

Flots concrets

- Tableaux
- Chaînes de caractères
- Pipes (tubes)
- Fichiers

Flots sur les tableaux

- Class **ByteArrayInputStream**
- Class **ByteArrayOutputStream**
- Class **CharArrayReader**
- Class **CharArrayWriter**

→ Construction de tableaux dont la taille n'est pas connue à l'avance

Exercice de TD

- Convertir un flot d'octets en un tableau d'octets.

Flots sur les chaînes de caractères

- Class **StringReader**
- Class **StringWriter**

Exercice de TD

- Utiliser les flots pour accéder aux caractères d'une chaîne.

Les pipes (tubes)

- Class **PipedInputStream**
- Class **PipedOutputStream**
- Class **PipedReader**
- Class **PipedWriter**

→ On connecte les flots en lecture aux flots en écriture en les passant en paramètre.

Exercice de TD

- Utilisation d'un pipe :
  - Créer un pipe de caractères.

Flots sur les fichiers

- Class **FileInputStream**
- Class **FileOutputStream**
- Class **FileReader**
- Class **FileWriter**
- Class **RandomAccessFile**

FileInputStream et FileOutputStream

- Dérivent de InputStream et OutputStream
- Classes permettant de créer des flots d'octets associés à des fichiers.

Les fichiers

```
File f  = new File("monfichier");
FileInputStream fis = new FileInputStream(f);
```

Ou :

```
FileInputStream fis = new FileInputStream("monfichier");
```

Exercices de TD

- Utilisation de FileInputStream :
  - Afficher le contenu d'un fichier.

Accès aléatoire

- Class **RandomAccessFile**

  getFilePointer()

  seek(long pos)

- Lecture et écriture dans le même flot.
- Lecture ou écriture à la position courante.
- Déplacement aléatoire possible.

Manipulation des fichiers

- Class **File**

- Représentation des fichiers **indépendante de la plate-forme**.
- Liste des fichiers d'un répertoire, suppression d'un fichier, etc.

Exercices de TD

- Utilisation de la classe File :
  - Programmer la fonction Unix : **ls -l**

BufferedInputStream et BufferedOutputStream

- Bufferisation des divers flots

```
BufferedOutputStream out = new
BufferedOutputStream(System.out, 255);
```

- Nouveau flot en écriture avec tampon de 255 caractères.

LineNumberReader

- Lecture d'un flot ligne par ligne grâce à la méthode **readLine()**
- Accès au numéro de ligne courante.

Les filtres

- Flot particulier qui enveloppe un autre flot : les données lues viennent du flot enveloppé
- Traitement possible sur les données : utilisation d'un tampon, codage, etc.

BufferedReader

- BufferedReader

`readLine()`

```
InputStreamReader isr = new InputStreamReader(System.out)
BufferedReader br = new BufferedReader(isr)
```

BufferedReader

- BufferedReader

`readLine()`

```
FileInputStream f = new FileInputStream("nom")
InputStreamReader isr = new InputStreamReader(f)
BufferedReader br = new BufferedReader(isr)

Ou

FileReader f = new FileReader("nom")
BufferedReader br = new BufferedReader(f)
```

BufferedWriter

- BufferedWriter

newLine()

```
OutputStreamWriter osr = new
OutputStreamWriter(System.in);

BufferedWriter bw = new BufferedWriter(osr);
```

BufferedWriter

- BufferedWriter

`newLine()`

```
FileOutputStream f = new FileOutputStream("nom");
OutputStreamWriter osr = new OutputStreamWriter(f);
BufferedWriter bw = new BufferedWriter(osr);
```

Ou

```
FileWriter f = new FileWriter("nom");
BufferedWriter br = new BufferedWriter(f);
```

Exercice de TD

- Utilisation de LineNumberReader :
  - Afficher le contenu d'un fichier ligne par ligne en faisant précéder chaque ligne de son numéro.

DataInputStream et DataOutputStream

- Lecture et écriture des types primitifs
- Lecture et écriture des chaînes
- Interfaces DataInput et DataOuput
  - writeChar
  - writeFloat
  - writeInt
  - writeDouble
  - writeBoolean
  - writeUTF
- Lecture et écriture binaire

Exercices de TD

- Utilisation de DataOutput :
  - Convertir des entiers et des réels en caractères.

PushbackInputStream et PushbackReader

- Possibilité de re-placer les données dans le flot avec **unread()**

PrintStream

- Ecriture dans un flot de données sous la forme de chaînes de caractères.
  - Utilisation de String.valueOf
  - PrintStream System.out
  - PrintStream : utilisation de print ou println

```
FileOutputStream fos = new FileOutputStream("nom.txt");
PrintStream ps = new PrintStream(fos);
ps.println("Bonjour tout le monde");
```

PrintStream

- Ecriture dans un flot de données sous la forme de chaînes de caractères.
  - Utilisation de String.valueOf
  - PrintStream System.out
  - PrintStream : utilisation de print ou println

```
FileOutputStream fos = new FileOutputStream("nom.txt");
PrintStream ps = new PrintStream(fos, true);
ps.println("Bonjour tout le monde");
```

Un flush est lancé après chaque écriture

PrintWriter

- Ecriture dans un flot de données sous la forme de chaînes de caractères.
  - Utilisation de String.valueOf
  - PrintStream System.out
  - PrintStream : utilisation de print ou println

```
FileOutputStream fos = new FileOutputStream("nom.txt");
PrintWriter pw = new PrintWriter (fos);
pw.println("Bonjour tout le monde");
```

Flots de lexèmes

- Class **StreamTokenizer**
- Comportement semblable à un filtre
- Décomposition d'un flot en une suite de lexèmes.
- Même principe que le StringTokenizer :

  Chaine → Stream

Exercice de TD

- Lire un fichiers de numéros (double) et afficher :
  - Le nombre de numéros lus
  - Leur somme
  - Leur moyenne

Passer d’octets aux caractères

De Stream à Reader

- Souvent on récupère un flux de caractères par l’intermédiaire d’un flux d’octets
  - Cas de System.in
  - Cas des Sockets
  - …

Questions ?