Programmation Java

flux (Streams) et sérialisation

P. Bakowski

bako@ieee.org

Lew flux (`Stream`)

Les flux de données constituent l'interface à travers laquelle s'effectue la communication avec l'utilisateur (entrée/sortie standard), avec le système de fichiers et avec le réseau.

Dans un flux, les données sont écrites et lues de façon séquentielle.

Le flux peut être accédé par des unités (blocs) de taille arbitraire: un octet, un mot, une ligne, etc.

Pour améliorer les performances de communication, un flux peut utiliser un tampon (`Buffer`); on parle de flux à tampon.

Les classes consacrées aux flux sont regroupées dans le package `java.io.`

Lew flux (Streams)

La classe abstraite `Reader` contient les méthodes `read()` permettant la lecture des caractères et arrays de caractères:

```
int read()

int read(char cbuf[])

int read(char cbuf[], int offset, int length)
```

La classe abstraite `Writer` contient les méthodes `write()` permettant l'écriture des caractères et arrays de caractères:

```
int write(int c)

int write(char cbuf[])

int write(char cbuf[], int offset, int length)
```

Les flux : InputStreamReader

```
import java.io.* ;
public class MonReadStream throws IOException{
  public static void main(String[] args){
  int nClus;
  Reader stdIn = new InputStreamReader(System.in);
  char[] userInput = new char[256];
  while((nClus=stdIn.read(userInput))!=-1)
        System.out.println(userInput);
  }
}
```

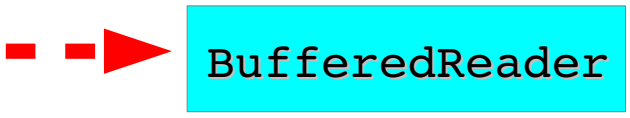
Lecture des caractères sans tampon

userInput[256]

Les flux : BufferedReader

```
import java.io.* ;
public class MonReadBuffered throws IOException{
  public static void main(String[] args){
  BufferedReader stdIn = new BufferedReader(
                         new InputStreamReader(System.in));
  String userInput;
  while((userInput=stdIn.readLine())!=null)
        System.out.println(userInput);
  }
}
```

Lecture des caractères avec tampon



Les flux : System.in et System.out

```
System.in. …  // méthode
int nol;
while((nol=System.in.read())!='.')..
```

```
System.out. …  // méthode

String userInput ;
System.out.println(userInput);
```

Flux d'octets

La classe abstraite **InputStream** contient:

```
int read();
int read(char cbuf[]);
int read(char cbuf[], int offset, int length);
```

La classe abstraite **OutputStream** contient :

```
int write();
int write(char cbuf[]);
int write(char cbuf[], int offset, int length);
```

Dispositifs pour chaînes d'octets

mémoire:

`CharArrayReader`	
`CharArrayWriter`	`ByteArrayInputStream,`
	`ByteArrayOutputStream`
`StringReader`	`StringBufferInputStream`
`StringWriter`	

fichiers:

`FileReader`	`FileInputStream`
`FileWriter`	`FileOutputStream`

Mémoire : lecture/écriture

```
String str = "XYZ"; // une chaine à écrire sur l'écran
StringReader reader = new StringReader(str);
int ncl;
while(( ncl=reader.read()) != -1)
// lecture caractère par caractère
{
System.out.println((char)ncl);
// écriture caractère par caractère
}
```

```
ByteArrayOutputStream out = new ByteArrayOutputStream();
int nol;
while((nol = System.in.read()) !=-1) { out.write(nol);}
// la taille du tampon out augmente automatiquement
byte[] btab = out.toByteArray();  // conversion
out.reset(); // effacement du tampon
```

Fichiers : lecture/écriture

Les classes de flux sur des fichiers sont: `FileInputStream`, `FileOutputStream`, `FileReader`, `FileWriter` et `RandomAccessFile`.

Les classes `FileInputStream` et `FileOutputStream` permettent de créer des flux d'octets.

Les classes `FileReader` et `FileWriter` sont dérivées des classes `InputStreamReader` et `OutputStreamWriter`.

La classe `RandomAccessFile` permet de déplacer le pointeur du fichier sur la position demandée.

Les constructeurs de flux sur des fichiers prennent en paramètre le nom du fichier ou le nom d'un objet de la classe `File`.

```
FileReader in = new FileReader("fichier.txt");

FileReader = new FileReader(new File("fichier.txt"));
```

Fichiers texte : CopyTextFile

```
import java.io.*;
public class CopyTextFile {
public static void main(String[] args) throws IOException
{
File inputFile  = new File(args[0]);
File outputFile = new File(args[1]);
FileReader in  = new FileReader(inputFile);
FileWriter out = new FileWriter(outputFile);
int c;
while ((c = in.read()) != -1) out.write(c);
in.close();
out.close(); }
```

Paramètres du programme `args[0]` et `args[1]`: `toto.txt tata.txt`
Attention: si le programme est lancé dans l'Eclipse le fichier `toto.txt` doit être disponible dans le répertoire du projet (project directory)

Fichier binaire: CopyByteFile

```
import java.io.*;
public class CopyByteFile {
public static void main(String[] args) throws IOException
{
FileInputStream  in  = new FileInputStream(args[0]);
FileOutputStream out = new FileOutputStream(args[1]);
byte[] tampon  = new byte[1024] ;
int bl;
while ((bl=in.read(tampon))!=-1) out.write(tampon,0,bl);
in.close();
out.close(); }
```

args[0] args[1] - keyboard1.png keyboard2.png

Attention: si le programme est lancé dans l'Eclipse le fichier keyboard1.png doit être disponible dans le répertoire du projet (project directory)

Tampons et Filtres

Quand un programme a besoin de ces données, il commence par regarder dans le tampon du flux avant de revenir à la source originale du flux.
Le tampon est implémenté par les classes `BufferedInputStream` et `BufferedOutputStream`.

Le flux d'entrée peut exploiter deux constructeurs:
`BufferedInputStream(InputStream)` et
`BufferedInputStream(InputStream,int)`.
La méthode la plus simple `read()` est sans arguments; elle retourne un octet - une valeur entre 0 et 255. A la fin du flux la méthode renvoie la valeur `-1`.

Le flux de sortie peut exploiter également deux constructeurs:
`BufferedOutputStream(InputStream)` et
`BufferedOutputStream(InputStream,int)`.
Le `int` spécifie la taille du tampon.
La méthode `write(int)` (0 - 255) permet d'envoyer un octet dans le flux à tampon de sortie.

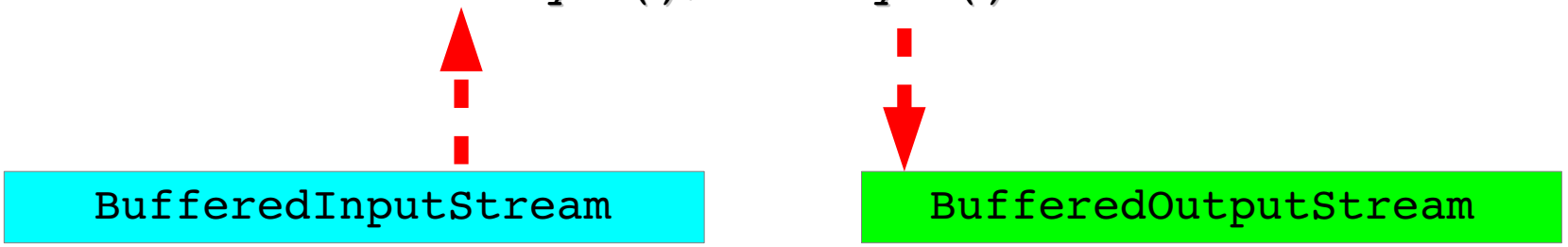
Tampons et Filtres

```
import java.io.*;
public class BufCopyFile {
public static void main(String[] args) throws IOException
{
FileInputStream  fin  = new FileInputStream(args[0]);
BufferedInputStream tamponIn =
                          new BufferedInputStream(fin);
FileOutputStream fout = new FileOutputStream(args[1]);
BufferedOutputStream tamponOut =
                          new BufferedOutputStream(fout);
int bl;
while ((bl=tamponIn.read())!=-1)
                          tamponOut.write(bl);
tamponIn.close();
tamponOut.close(); }
```

Tampons et Filtres

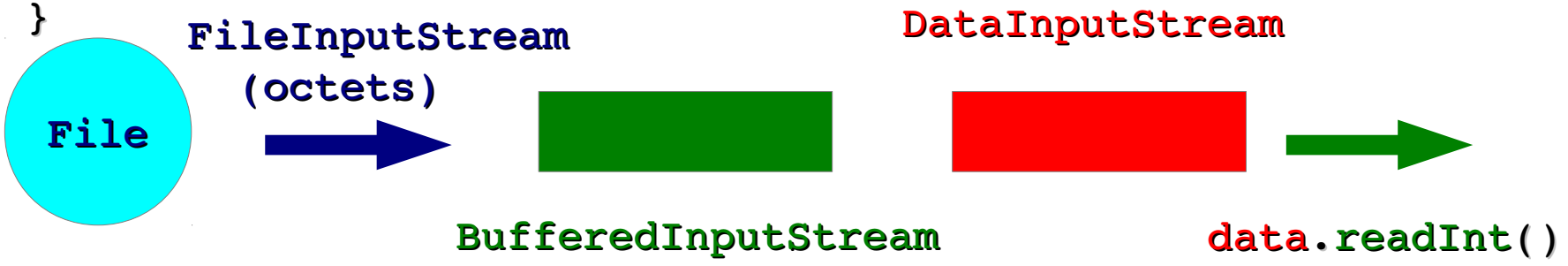
La lecture/écriture dans un flux de données peut être réalisée par les méthodes suivantes (filtres):

```
readInt()/ writeInt()
readFloat()/writeFloat()
readLong()/writeLong()
readBoolean()/writeBoolean()
readDouble()/writeDouble()
readByte()/writeByte()
```



Tampons et Filtres

```
import java.io.*;
public class ReadIntFile {
public static void main(String[] args) throws IOException {
FileInputStream  fin  = new FileInputStream(args[0]);
BufferedInputStream tampon =
                          new BufferedInputStream(fin);
DataInputStream  data = new DataInputStream(tampon) ;
try {
while (true) {
  int in=data.readInt();
  System.out.print(in + « »);}
  }
  catch(EOFException eof) {tampon.close();}
  }
}
```



Flux et filtres d'objets

Un flux d'objets permet de lire/écrire des objets Java dans un flux.
Cette technique, appelée sérialisation, peut être réalisée par les filtres `ObjectInputStream` et `ObjectOutputStream`.

L'implémentation des interfaces `DataOutput` et `DataInput` est nécessaire pour la sauvegarde des champs de type primitif.

La première fois qu'un objet est sauvegardé dans un flux, tous les objets qui peuvent être atteints depuis cet objet le sont également.
Pour sauvegarder un objet dans un flux, la classe de cet objet doit implémenter l'interface **`Serializable`**.
L'interface **`Serializable`** ne possède aucune méthode.
Lors de la sauvegarde d'un objet sérialisable on sauvegarde un descripteur de sa classe, suivi des valeurs de tous ses champs non statiques. Le descripteur de la classe spécifie son nom, sa version et les types et noms de ses champs.
Pendant la lecture d'un objet sérialisé, ces informations permettent la récupération de sa classe. Les valeurs des champs enregistrées dans le flux permettent d'en reconstruire une instance.

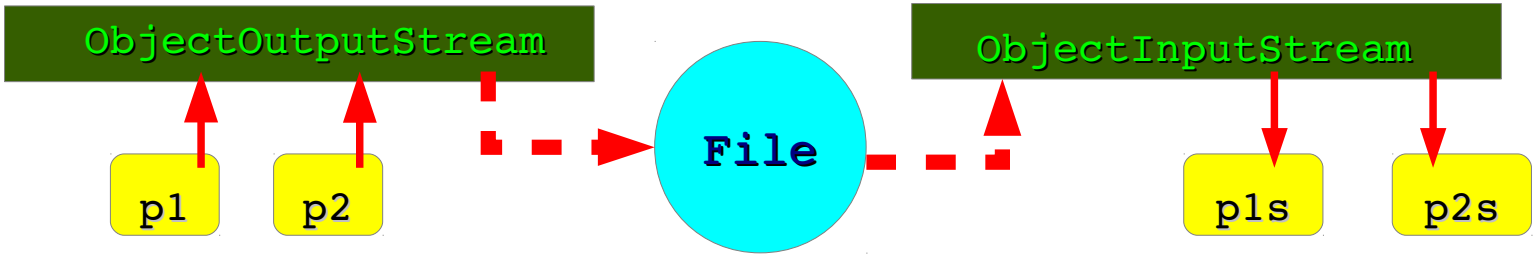
Flux et filtres d'objets

L'objet à sérialiser :

```
import java.io.*;
public class ReadSerPoint implements Serializable {
int x; int y;
public ReadSerPoint(int x, int y) {
this.x=x;
this.y=y;
}
public String toString() {
return "(" + x + "," + y + ")";
}
```

Flux et filtres d'objets (suite)

```
public static void main(String[] args) throws Exception {
ReadSerPoint p1 = new ReadSerPoint(3,4);
ReadSerPoint p2 = new ReadSerPoint(5,6);
File fichier = File.createTempFile("tempFichier","code");
fichier.deleteOnExit(); // detruit a la fin d'execution
ObjectOutputStream ob_out = new ObjectOutputStream(new
FileOutputStream(fichier));
ob_out.writeObject(p1); ob_out.writeObject(p2); ob_out.close();
ObjectInputStream ob_in = new ObjectInputStream(new
FileInputStream(fichier));
ReadSerPoint p1s = (ReadSerPoint) ob_in.readObject();
ReadSerPoint p2s = (ReadSerPoint) ob_in.readObject();
ob_in.close();
System.out.println("p1="+p1);System.out.println("p2="+p2);
System.out.println("p1s="+p1s);System.out.println("p2s="+p2s);
System.out.println("p1 egale a p1s ? "+p1.equals(p1s));
System.out.println("p2 egale a p2s ? "+p2.equals(p2s)); }
}
```



Résumé

Les flux – `Stream`

`InputStreamReader` et `OutputStreamWriter`

Entrées/Sorties standard : `System.in.`, `System.out.`

Dispositifs : mémoire et fichiers

Fichiers texte et fichiers binaires (octets)

Tampons et Filtres

Sérialisation des objets