LES FLUX DE DONNEES ET LA SERIALISATION DES OBJETS

Ludovic Liétard

INTRODUCTION (1)

- Un flux de données représente une suite d'octets :
  - Issue d'un programme pour une destination
  - En provenance d'une source pour un programme
- Exemples de destinations ou de sources :
  - Chaîne de caractères
  - Fichier
  - Socket…

INTRODUCTION (2)

- Pour manipuler des flux, Java propose quatre classes abstraites qui héritent directement de la classe `Object`
- Pour les flux d'octets :
  - Classe `InputStream` (lire des octets)
  - Classe `OutputStream` (écrire des octets)
- Pour les flux de caractères :
  - Classe `Reader` (lire des caractères)
  - Classe `Writer` (écrire des caractères)

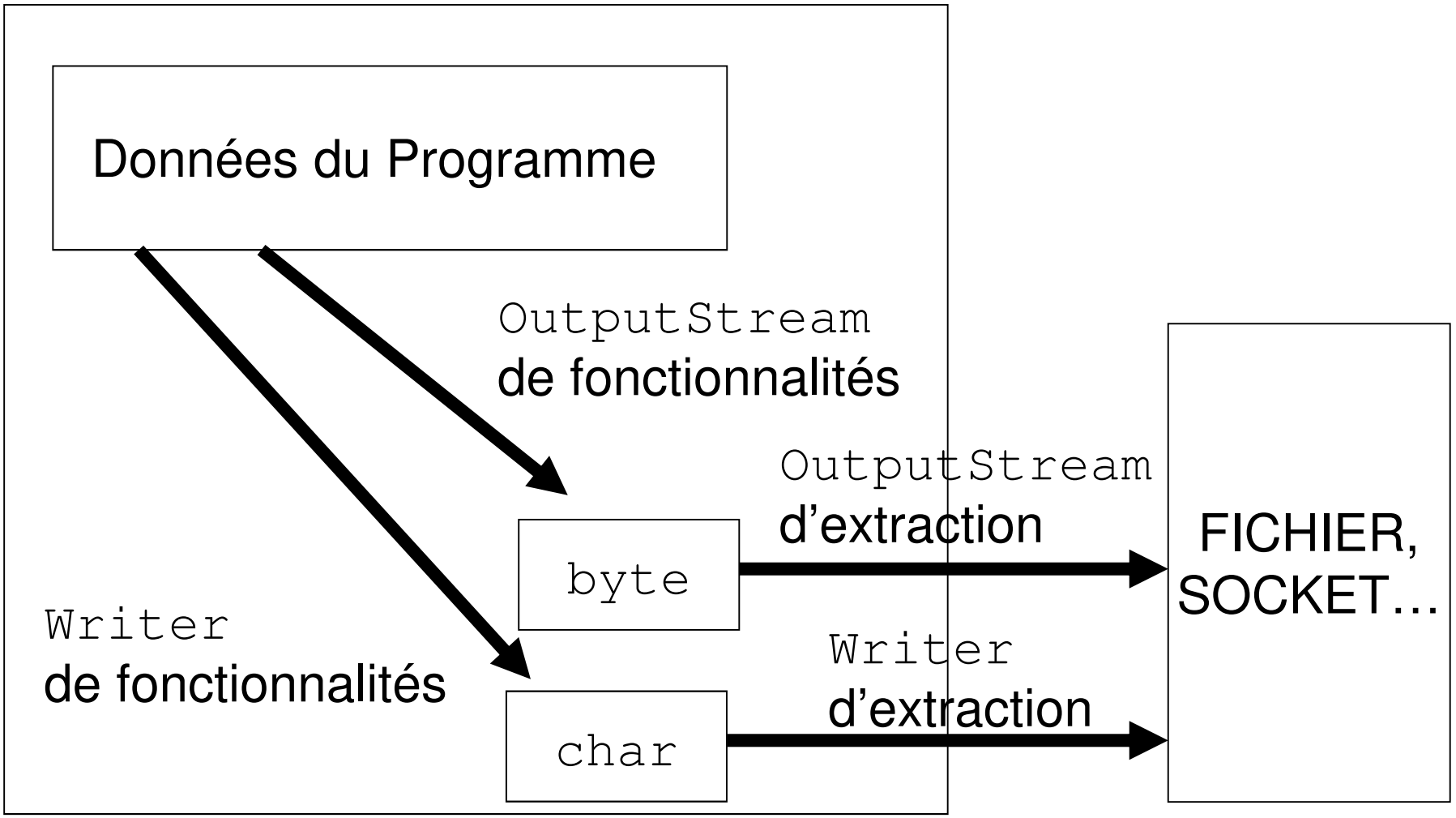
INTRODUCTION (3)

- Ces quatre classes sont abstraites
- Des classes qui en héritent on distingue :

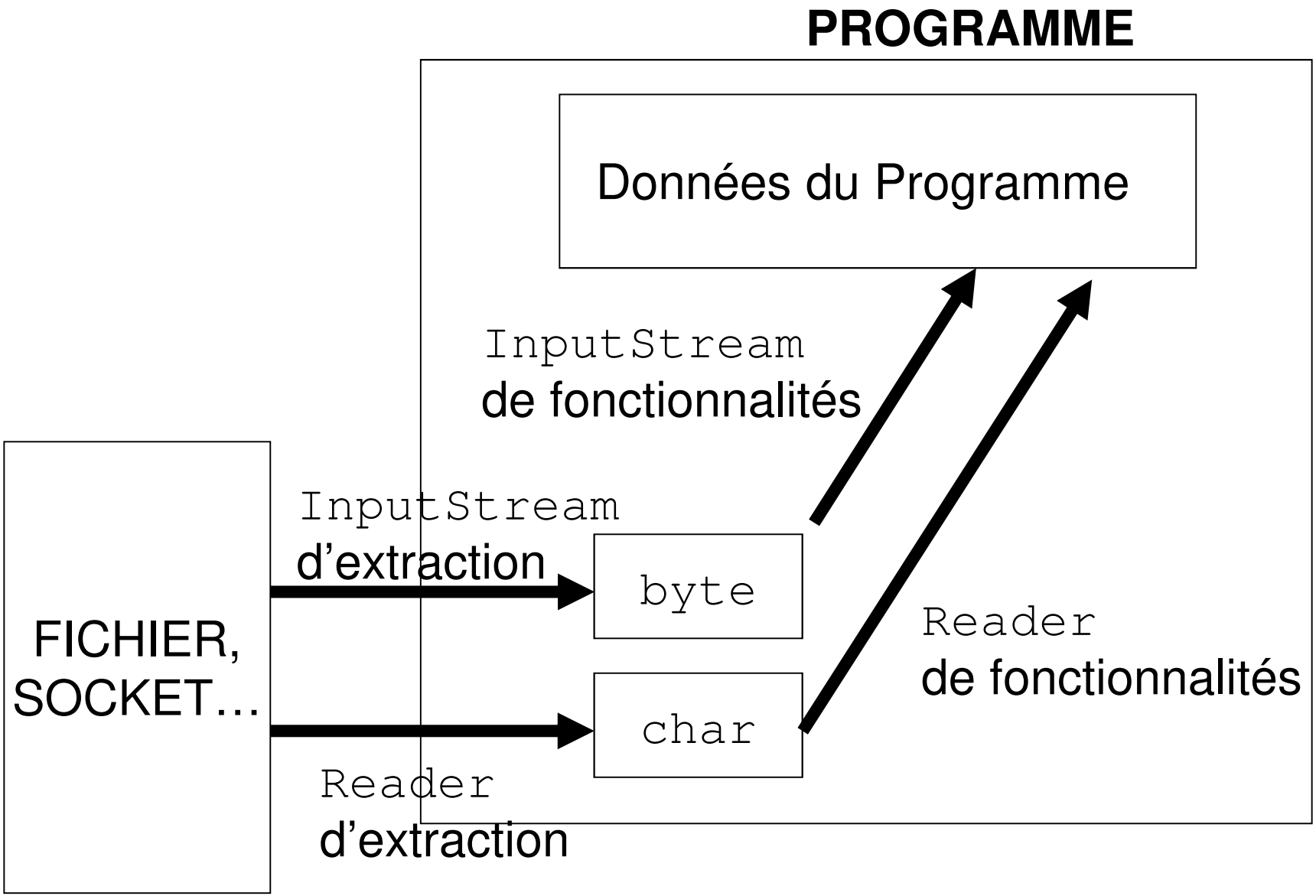
  - Les classes d'extraction (de bas niveau)

  - Les classes de fonctionnalités (de plus haut niveau qui utilisent les précédentes)

INTRODUCTION (4)



INTRODUCTION (5)



LES SOUS-CLASSES D'INPUTSTREAM

- Hiérarchie des classes d'extraction :
  - `StringBufferInputStream`, `ByteArrayInputStream`, `PipedInputStream`,`FileInputStream` héritent de `InputStream`

- Hiérarchie des classes de fonctionnalités :
  - `BufferedInputStream` et `DataInputStream` héritent de `FilterInputStream`
  - `FilterInputStream`, `ObjectInputStream`, `SequenceInputStream` héritent de `InputStream`

LES SOUS-CLASSES D'OUTPUTSTREAM

- Hiérarchie des classes d'extraction :
  - `ByteArrayOutputStream`, `PipedOutputStream`, `FileOutputStream` héritent de `OutputStream`
- Hiérarchie des classes de fonctionnalités :
  - `BufferedOutputStream`, `PrintStream` et `DataOutputStream` héritent de `FilterOutputStream`
  - `FilterOutputStream`, `ObjectOutputStream`, héritent de `OutputStream`

LES SOUS-CLASSES DE READER

- Hiérarchie des classes d'extraction :
  - `CharArrayReader`, `StringReader`,`PipedReader`, `FileReader` héritent de `Reader`
- Hiérarchie des classes de fonctionnalités :
  - `BufferedReader`,`FilterReader` et `InputStreamReader` héritent de `Reader`

LES SOUS-CLASSES DE WRITER

- Hiérarchie des classes d'extraction :
  - `CharArrayWriter`, `StringWriter`, `FileWriter`, `PipedWriter` héritent de `Writer`
- Hiérarchie des classes de fonctionnalités :
  - `BufferedWriter`, `FilterWriter`, `OutputStreamWriter` et `PrintWriter` héritent de `Writer`

LES SOUS CLASSES D'INPUTSTREAM (1)

- Les classes d'extraction
  - `StringBufferInputStream` : `InputStream` où les octets lus sont fournis par une instance de `String`
  - `ByteArrayInputStream : InputStream` où les octets lus sont fournis par un tableau d'octets
  - `PipedInputStream : InputStream` qui doit être relié à un `PipedOutputStream` (instance d'`OutputStream`) pour former un tube d'octets
  - `FileInputStream` : `InputStream` qui obtient des octets depuis un fichier.

LES SOUS CLASSES D'INPUTSTREAM (2)

- Les classes de fonctionnalités
  - `BufferedInputStream` : permet de gérer un buffer (tableau d'octets) associé à `l'InputStream`
  - `DataInputStream` : permet de lire des types primitifs depuis un `InputStream`
  - `FilterInputStream` : pour proposer des méthodes de lecture (uniformisation des `InputStream`)
  - `ObjectInputStream` : désérialise des données primitives ou des objets (écrits en utilisant un `ObjectOutputStream`)

LES SOUS CLASSES D'INPUTSTREAM (3)

- `SequenceInputStream` :
  concaténation logique d'`InputStream`
  qui sont alors lus les uns après les autres

LES SOUS CLASSES D'OUTPUTSTREAM (1)

- Les classes d'extraction
  - `ByteArrayOutputStream : OutputStream` où les octets sont écrits dans un tableau d'octets
  - `PipedOutputStream : OutputStream` qui doit être relié à un `PipedInputStream` (instance d'`InputStream`) pour former un tube d'octets
  - `FileOutputStream : OutputStream` qui écrits des octets dans un fichier.

LES SOUS CLASSES D'OUTPUTSTREAM (2)

- Les classes de fonctionnalités
  - `BufferedOutputStream` : permet de gérer un buffer (tableau d'octets) associé à `l'OutputStream`
  - `PrintStream` et `DataInputStream` : permettent d'écrire des types primitifs dans un `OutputStream`
  - `FilterOutputStream` : pour proposer des méthodes de lecture (uniformisation des `OutputStream`)
  - `ObjectOutputStream` : pour écrire des types primitifs et des objets (sérialisables) dans un `OutputStream`

LES SOUS CLASSES DE READER (1)

- Les classes d'extraction
  - `CharArrayReader : Reader` qui lit depuis un tableau de char
  - `StringReader : Reader` qui lit depuis une instance de `String`
  - `PipedReader : Reader` qui doit être relié à un `PipedWriter` (instance de `Writer`) pour former un tube de char
  - `FileReader : Reader` pour lire dans un fichier texte

LES SOUS CLASSES DE READER (2)

- Les classes de fonctionnalités
  - `BufferedReader` : associe un buffer (de char) à un `Reader`
  - `FilterReader` : pour lire dans un `Reader`
  - `InputStreamReader` : pour lire dans un `InputStream` comme s'il était un `Reader` (conversion)

LES SOUS CLASSES DE WRITER (1)

- Les classes d'extraction
  - `CharArrayWriter : Writer` qui écrit dans un tableau de char
  - `StringWriter : Writer` qui écrit dans une instance de `String`
  - `PipedWriter : Writer` qui doit être relié à un `PipedReader` (instance de `Reader`) pour former un tube de char
  - `FileWriter : Writer` pour écrire dans un fichier texte

LES SOUS CLASSES DE WRITER (2)

- Les classes de fonctionnalités
  - `BufferedWriter` : associe un buffer (de char) à un `Writer`
  - `FilterReader` : pour écrire dans un `Writer`
  - `OutputStreamWriter` : pour écrire dans un `OutputStream` comme s'il était un `Writer` (conversion)
  - `PrintWriter` : pour faire des écritures formatées dans un `Writer`

EXEMPLE (1)

- La classe de fonctionnalités `PrintWriter` admet un constructeur avec un paramètre instance de `Writer`.

- Les classes `FileWriter` et `StringWriter` héritent de `Writer`. Ce sont deux classes d'extraction qui servent à écrire dans un fichier pour la première et dans une chaîne de caractères pour la deuxième.

- Ainsi :
  - new PrintWriter(new FileWriter("fichier.txt "));
  - new PrintWriter(new StringWriter());

permettent de disposer de deux objets pour écrire, avec les même méthodes, dans un fichier ou une chaîne de caractères. On peut par exemple, utiliser la méthode println.

EXEMPLE (2)

- Pour lire des informations au clavier :
  - `System.in` est une instance de `InputStream` (classe d'extraction)
- Mais le codage est dépendant de la machine.
- Le codage de caractères utilisé par Java est le codage Unicode. Pour traduire on créer une instance de InputStreamReader à partir de l'InputStream.

EXEMPLE (3)

- On a donc : `new InputStreamReader(System.ini)`
- Mais une telle instance ne peut lire que des caractères les uns après les autres.
- Première façon : On crée une instance de `BufferedReader` qui permet de lire une ligne en entier par la méthode `readline`().

```
String ligne;
BufferedReader entree = new BufferedReader
                        (new InputStreamReader(System.in));
ligne = entree.readline();
```

- Cette ligne est une chaîne de caractères que l'on traite par la création d'une instance de `StringTokenizer` (pour la découper en mots qui sont les informations saisies au clavier).

EXEMPLE (4)

- On a donc : `new InputStreamReader(System.ini)`
- Mais une telle instance ne peut lire que des caractères les uns après les autres.
- Deuxième façon : On crée une instance de `StreamTokenizer` qui fournit un analyseur syntaxique rudimentaire.

```
StreamTokenizer entree = new StreamTokenizer
                         (new
InputStreamReader(System.in));
```

- Cette instance de `StreamTokenizer` permet d'accéder aux informations saisies au clavier.

SERIALISATION (1)

- La classe `ObjectOutputStream` est une classe de fonctionnalités qui permet d ’écrire des objets dans un flux binaire à l ’aide de la méthode `writeObject`

- La classe `ObjectIntputStream` est une classe de fonctionnalités qui permet de lire des objets dans un flux binaire à l ’aide de la méthode `readObject`

- On appelle sérialisation la technique utilisée pour écrire des objets dans un flux binaire.

- La sérialisation peut servir :
  - à archiver des objets dans un fichier binaire
  - à transmettre des objets à travers un réseau
  - pour appeler des méthodes distantes (RMI : Remote Method Invocation)

SERIALISATION (2)

- Pour qu ’un objet puisse être sérialisé, il est nécessaire que sa classe implémente l ’interface `Serializable`. Cette interface ne contient aucune déclaration de méthode; lorsqu ’une classe implémente l ’interface `Serializable`, elle autorise la sérialisation de ses instances.

- Les tableaux d ’objets sérialisables ou de types primitifs sont sérialisables.

- Si on tente de sérialiser un objet d ’une classe qui n ’implémente pas l ’interface `Serializable`, une exception `NotSerializableException` est lancée.

SERIALISATION (3)

- La classe `ObjectOutputStream` (resp. `ObjectInputStream`) possède aussi des méthodes pour écrire (resp. lire) tous les types primitifs :
  - `writeInt, readInt, ...`
- Exemple :

```
class ClasseEssai implements Serializable {
        String chaine;
        int n;

        public ClasseEssai() {
                chaine = "Bonjour";
                n = 1;
        }
}
```

SERIALISATION (4)

```
ObjectOutputStream sortie = new ObjectOutputStream
                            (new FileOutputStream(" essai.dat "));

sortie.writeObject(new ClasseEssai());
sortie.writeObject(new Date());

int[] tableau1 = {10, 11, 12};
sortie.writeObject(tableau1);

Integer[] tableau2 = {new Integer(110), new Integer (111)}
sortie.writeObject(tableau2);

sortie.close();
```

SERIALISATION (5)

```
ObjectInputStream entree = new ObjectInputStream
                              (new FileInputStream(" essai.dat "));

ClasseEssai c = (ClasseEssai) entree.readObject();
System.out.println(c.chaine);
System.out.println(c.n);

Date d = (Date) entree.readObject();
System.out.println(d);

int[] tableau3 = (int []) entree.readObject;
System.out.println(tableau3[1]);

Integer[] tableau4 = (Integer[]) entree.readObject;
System.out.println(tableau4[0]);

entree.close();
```