JAV - TD 7
Les entrées / sorties JAVA

Les Flux

Le package java.io

La gestion de fichiers en java

La sérialisation d'objets

Les Flots/Flux/Streams

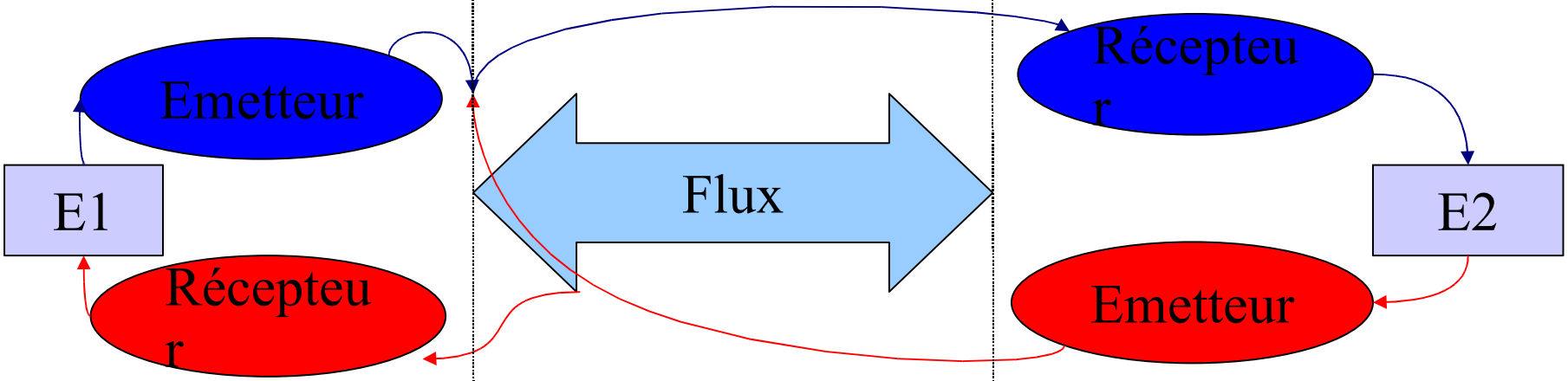
- Toutes les entrées/sorties en JAVA sont gérées par des flux (streams)
  - Lecture du clavier
  - Affichage sur la console
  - Lecture/écriture dans un fichier
  - Echange de données réseau avec des sockets

- Stream: objet JAVA possédant la caractéristique de pouvoir envoyer ou recevoir des données

Flux / Flots / Stream

- Un flux est une serie d'informations envoyée sur un canal de communication
- C'est le paradigme utilisé dans le monde objet pour représenter une communication entre 2 entités
- Un flux se comporte comme une rivière :
  - Il faut une source/émetteur
  - Et un recepteur/consommateur
    - Un flux est bidirectionnel, par contre les points d'accès (émetteur/consommateur) ne peuvent travailler que dans un seul sens

Circulation d'information sur un flux



- Les flux informatiques ne véhiculent que des octets
- Les émetteurs et les récepteurs se chargent de transformer les octets sous des formes plus intéressantes pour les entités (data, buffer, crypto...)
- Emetteur et récepteur doivent se mettre d'accord sur le format des structures envoyées (notion de protocole d'accord)
- La bibliothèque java.io définit l'ensemble des transformations que l'on peut désirer sur un ensemble d'octets arrivant dans le flux
- Il existe également des émetteurs et des récepteurs standards

Emetteurs et récepteurs standards du système d'exploitation

- Il existe 3 flux standards pour un système d'exploitation
  - Les octets circulant entre une application (A) et l'écran (E) pour indiquer une information standard **System.out**
  - Les octets circulant entre une application (A) et l'écran (E) pour indiquer une information d'erreur **System.err**
  - Les octets circulant entre le clavier (C) et une application (A) **System.in**
- **Exercice:** Représentez sur un schéma les flux et les points d'accès liés aux entrées/sorties standards

Exemple de saisie avec System.in

- Voici le code qui lit l'entrée du clavier et qui envoie le caractère sur la sortie standard (affichage du code UNICODE du caractère)

```
import java.io.IOException;

public class MainClass {

    public static void main(String[] args)  throws IOException{
        System.out.println("Entrez un caractère");
        int inChar = System.in.read();
        System.out.println("Vous avez saisi le caractère de code
            : "+inChar);
    }

}
```

Intanciation des émetteurs / récepteurs

- Il existe des entités classiques pouvant être la source ou la destination d'octets
  - Le fichier : il permet de stocker des octets
    - Classe: java.io.File (nommage, droits d'accès et propriétés d'un fichier ou d'un répertoire)
    - File f=new File("/tmp/toto");
  - La socket réseau : elle permet d'envoyer des octets à une autre machine
    - Classe: java.net.Socket (point d'accès pour une connexion TCP/IP entre deux machines)
    - Socket s=new Socket("www.insa-lyon.fr", 80)

Instanciation des flux de communication

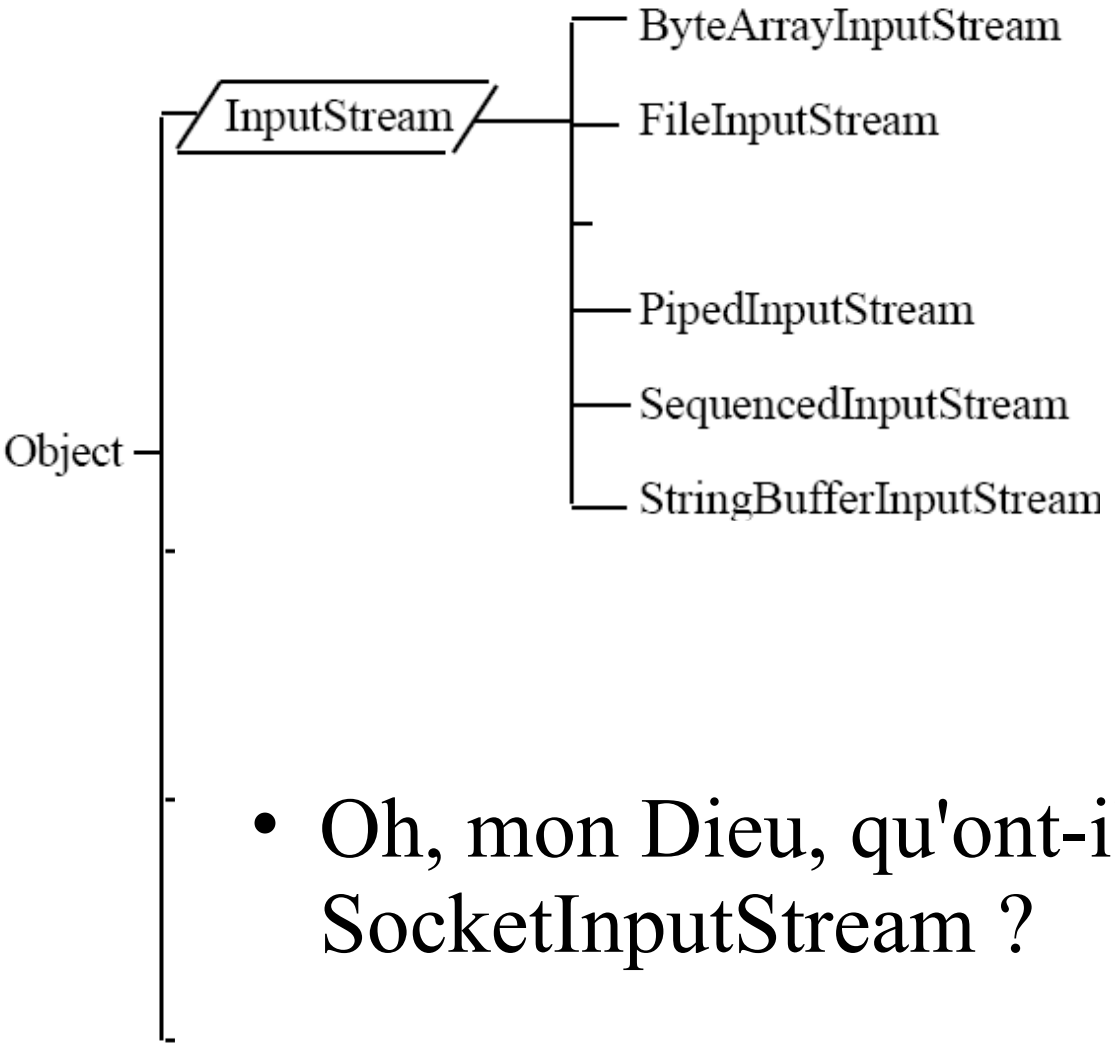
- Après la création de la source ou de la destination, il est nécessaire de construire des objets d'accès au flux pour pouvoir échanger les données
- Les accès sur les points d'entrée ne sont pas toujours homogènes :

```
File f=new File("/tmp/toto");
FileInputStream fin=new FileInputStream(f);
val=fin.read();
Socket s=new Socket("www.insa-lyon.fr", 80);
InputStream sin=s.getInputStream();
val2=sin.read();

private void lecture (InputStream in) throws IOException{
  int i=in.read();
  while(i!=-1) {System.out.println(i); i=in.read();}
}
```

- Que peut-on dire des classes FileInputStream et InputStream ? Quelles sont les fonctions que l'on peut appliquer sur un flux d'entrée ? Et sur un flux de sortie ?

java.io: arborescence de classes



- Oh, mon Dieu, qu'ont-ils fait de la SocketInputStream ?

API : InputStream

`int available()`

- Returns the number of bytes that can be read (or skipped over) from this input stream without blocking by the next caller of a method for this input stream.

`int read()`

- Reads the next byte of data from the input stream.

`int read(byte[] b)`

- Reads some number of bytes from the input stream and stores them into the buffer array b.

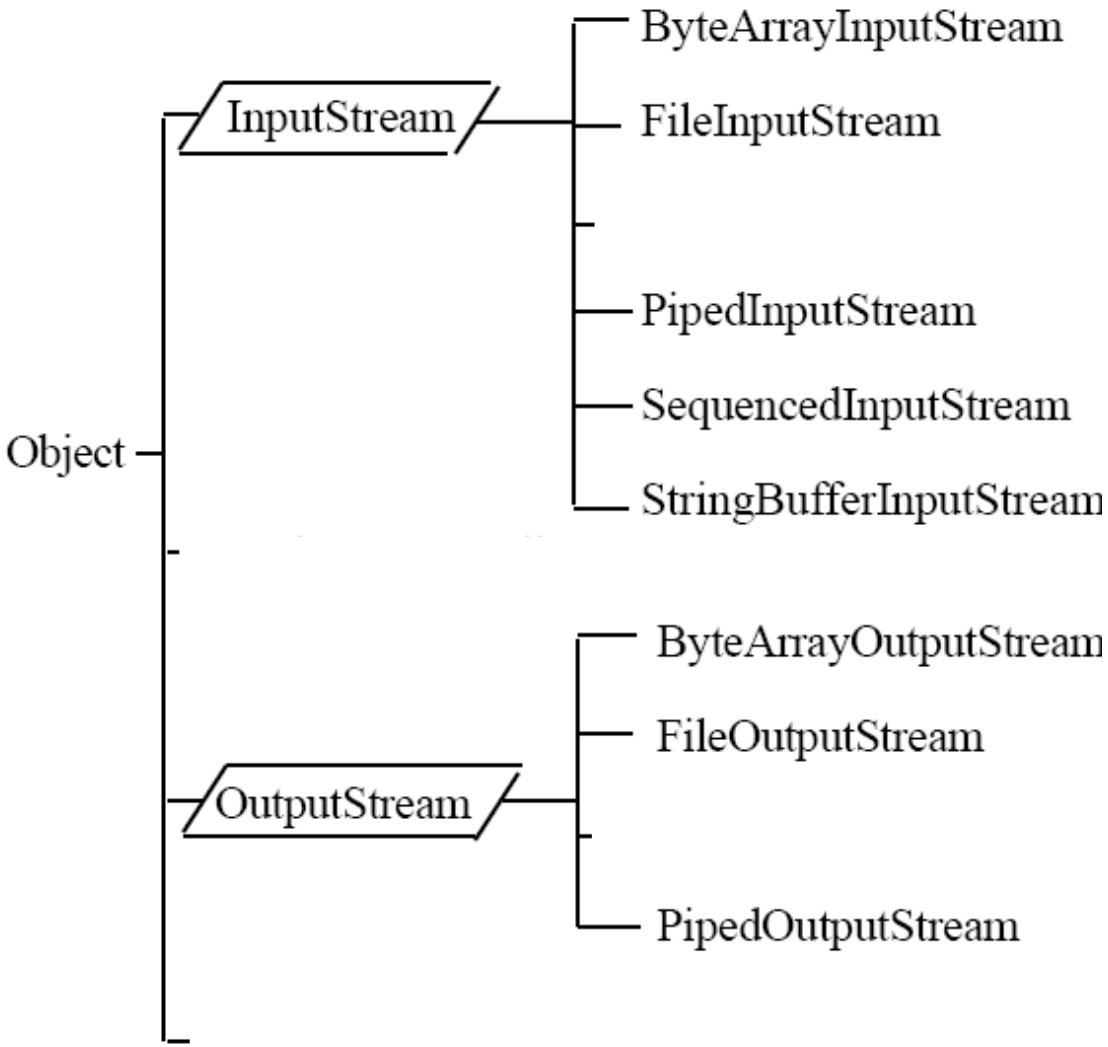
`int read(byte[] b, int off, int len)`

- Reads up to len bytes of data from the input stream into an array of bytes.

`void close()`

- Closes this input stream and releases any system resources associated with the stream.

java.io: arborescence de classes



Exercice: copie de fichiers

- Ecrire une classe Copy qui réalise la copie d'un fichier dans un autre.

- Exemple d'utilisation:

java Copy sourceFile.txt destFile.txt

Que se passe-t-il si ?

- Que doit faire une entité lorsqu'elle veut envoyer un long, double… ?
- Que doit faire une entité lorsqu'elle tente de lire un byte, et que celui-ci n'est pas encore disponible ?
- Quelle différence y a-t-il en C entre 'a' et "Bonjour"?
- Quelle différence y a-t-il entre Unix et Windows dans le texte suivant?

  "Bonjour,\nTout le monde"

==> Il faut "décorer" nos flux

Décoration de flux

- La décoration de flux est l'art de traiter une série d'octets avant de les transmettre à l'application demandeuse.

- Il existe 4 familles de décorations de flux
  - Data : permet de convertir les bytes en structure primitive (long, boolean, double...)
  - Character : permet de convertir les bytes en texte
  - Object : permet de convertir les bytes en structure objet
  - Service : buffer, crypto, gzip permet d'appliquer des fonctions spécifiques sur la lecture

- Instanciation de décorateur:

```
DataInputStream dis=new DataInputStream((new FileInputStream("/tmp/toto");
```

Flux de données: Data(Input|Output)Stream

- Ces classes permettent de lire et d'écrire des types primitifs et des lignes sur des flux.

```
FileInputStream fis = new FileInputStream("source.txt");
DataInputStream dis = new DataInputStream(fis);

int i    = dis.readInt();
double d = dis.readDouble();

FileOutputStream fos = new FileOutputStream("cible.txt");
DataOutputStream dos = new DataOutputStream(fos);

dos.writeInt(123);
dos.writeDouble(123.456);
```

Flux de caractères

- Ajout de la notion de flux de caractères (character Streams)
  - Les InputStream et OutputStream se basent sur la lecture et écriture d'octets (bytes)
  - Or les caractères sont codés sur 2 octets

  →Utilisations de deux classes (writer et reader) à la place de InputStream et OutputStream

  →Ecriture facile des caractères UNICODE

  →Pour lire et écrire directement des caractères UNICODE dans un fichier, il est conseillé d'utiliser FileReader et FileWriter à la place de FileInputStream et FileOutputStream

Flux d'objets : Sérialisation

- La sérialisation (Serialization) est le processus de transformation d'un objet en un flux d'octets, et la désérialisation est le processus inverse.
- La sérialisation permet à un objet d'être facilement sauvé sur un fichier ou transféré sur le réseau
- Les classes doivent implanter l'interface Serializable

Flux d'objets : Syntaxe

- La classe de l'objet qu'on veut sérialiser doit implémenter la classe java.io.Serializable

```
public class Elem implements java.io.Serializable{
  private int i;
  private Hello[] h;
  ...
}
```

Sérialisation : exercice 1

Sérialisez des instances de l'objet suivant dans un fichier “/tmp/test” :

```
public class Entry {
  private java.util.Date d;
  private Object src;
  private String message;
}
```

Sérialisation : exercice 1 suite

Désérialisez les instances précédentes.

Exercice: Ecriture de logs

- Ecrire un utilitaire de gestion de logs « WriteLog ».
- Chaque log représente une ligne dans un fichier texte.
- Sur chaque ligne on trouve la date d'écriture, l'identifiant de son écrivain et son commentaire.
- Utilisez une classe Entry ayant pour attributs : la date, l'identifiant de l'écrivain et son commentaire.
- Cette classe doit définir la méthode de sérialisation de ses attributs dans un flux de sortie OutputStream (qui aura été préalablement ouvert).

Exercice: Lecture de logs

- Ecrire un utilitaire « ScanLog » de lecture des logs enregistrés dans l'exercice précédent. Cet utilitaire doit permettre un parcours sélectif en utilisant les options suivantes:
  - *-after d* : pour afficher les enregistrements écrits après la date *d*
  - *-before d* : pour afficher les enregistrements écrits avant la date *d*
  - *-user u* : pour afficher que les enregistrements écrits par l'écrivain *u*

java.io: ce qu'il faut retenir