Les entrées/sorties

Flux

Flux d'entrée/sortie ¶ §

- En java, toutes les entrées/sorties sont gérés par des **flux** (= *streams*).
  - Lecture du clavier
  - Affichage sur la console
  - Lecture/écriture dans un fichier
  - Echange de données réseau avec des sockets
- Un flux est une série d'informations envoyée sur un canal de communication entre deux **entités**.

Circulation d'informations dans un flux

- Il y a un **émetteur** (qui envoie/écrit) et un **récepteur** (qui reçoit/lit).
- Le flux véhicule des **octets** entre eux.
- Emetteur et récepteur doivent se mettre d'accord sur le format des données envoyées (protocole)
- Emetteur et récepteur se chargent de la transformation des données.

Flux standards

Dans un système d'exploitation, il existe trois flux standards:

- entre l'application et l'écran pour transmettre une information (**System.out**).
- entre l'application et l'écran pour indiquer une erreur (**System.err**).
- entre le clavier et l'application (**System.in**).

Exemple de saisie avec *System.in*

Voici un programme qui lit l'entrée du clavier et qui envoie le caractère saisi 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: "+inChar);
  }
}
```

Autres entités

Outre les flux standards, d'autres flux sont possibles entre des entités pouvant émettre ou recevoir des informations:

- Le **fichier**, point d'accès au disque dur (*java.io.File*).
- Le **socket réseau**, point d'accès pour une connexion TCP/IP entre deux machines (*java.net.Socket*)

Le package *java.io*

Description

Le package *java.io* fournit un mécanisme d'entrée/sortie au moyen de flux de données, de caractères, d'objets. Le fichier est un émetteur ou récepteur privilégié de ces flux.

Consultez le tutoriel IO sur les entrées/sorties ainsi que l'API standard.

Création d'un flux

1. instanciation des entités

```
File f1 = new File("/tmp/toto");
File f2 = new File("/tmp/titi");
```

2. instanciation des récepteurs (flux d'entrée) ou des émetteurs (flux de sortie)

```
FileInputStream fin = new FileInputStream(f1);
FileOutputStream fout = new FileOutputStream(f2);
```

3. réception (lecture) ou émission (écriture)

```
val = fin.read();
fout.write(val);
```

Flux d'entrée

Les flux d'entrée héritent tous de la classe abstraite *InputStream* dont voici les méthodes principales:

- `abstract int read()` Lit le prochain octet. La valeur de l'octet est retournée comme un

entier entre 0 et 255, -1 si la fin de fichier est atteinte.
- `int read(byte[] b)` Lit un certain nombre d'octets et les copie dans le buffer b. Le nombre d'octets lu est retourné, -1 si la fin de fichier est atteinte.
- `void close()` Ferme proprement le flux.

Flux de sortie

Les flux de sortie héritent tous de la classe abstraite *OutputStream* dont voici les méthodes principales:

- `void write(int b)` Ecrit l'octet b (seulement les 8 bits de poids faibles sont pris en compte).
- `void write(byte[] t)` Ecrit les octets du tableau d'octets t.
- `void close()` Ferme proprement le flux.

Ex.1. Copie de fichiers (30 min)

- Ecrivez une classe exécutable *Copy* qui réalise la copie d'un fichier dans un autre octet par octet, en respectant la convention suivante:
  - `java Copy source cible` pour copier `source` vers `cible`.
  - `java Copy source` pour écrire sur la sortie standard.
  - `java Copy` pour lire puis écrire sur la sortie standard.
- Le coeur de la copie sera effectuée dans la méthode suivante:

```
private static void copy(InputStream is, OutputStream os)
  throws IOException { ... }
```

Décoration de flux

Pour traiter une série d'octets avant de les transmettre, il est possible de **décorer** les flux. Dans ce contexte, un décorateur est un objet léger qui est construit à partir d'un flux et qui se comporte comme un flux. Il réalise des traitements supplémentaires avant ou après la transmission qui est déléguée au flux à partir duquel il est construit.

Il existe plusieurs familles de décorations de flux:

- données : conversion des octets en types primitifs (long, double, etc.)
- caractères : conversion en texte
- objet : conversion en objets

Exemple de flux de données

Comment faire pour lire/écrire des variables de type double?

```
FileInputStream fis = new FileInputStream("source");
DataInputStream dis = new DataInputStream(fis);
double d = dis.readDouble();
```

```
FileOutputStream fos = new FileOutputStream("cible");
DataOutputStream dos = new DataOutputStream(fos);
dos.writeDouble(123.456);
```

Exemple de flux de caractères

Comment faire pour lire/écrire les caractères UNICODE (sur 2 octets) ?

```
FileInputStream fis = new FileInputStream("source.txt");
InputStreamReader r = new InputStreamReader(fis);
//equivalent: FileReader r = new FileReader("source.txt");
char c = (char) r.read();
```

```
FileOutputStream fos = new FileOutputStream("cible.txt");
OutputStreamWriter w = new OutputStreamWriter(fos);
//equivalent: FileWriter w = new FileWriter("cible.txt");
w.write( (int) 'a' );
```

Exemple de flux de lignes

Comment faire pour lire/écrire des lignes dont le caractère de fin de ligne dépend du systèmes d'exploitation ?

```
FileInputStream fis = new FileInputStream("source.txt");
BufferedReader in = new BufferedReader( new InputStreamReader(fis) );
String line = in.readLine();
```

```
FileOutputStream fos = new FileOutputStream("cible.txt");
PrintWriter out = new PrintWriter( new OutputStreamWriter(fos) );
out.println("blabla");
```

Flux d'objets: Sérialisation

La **sérialisation** (= *serialization*) est le processus de transformation d'un objet en flux (série) d'octets. La déserialisation est le processus inverse.

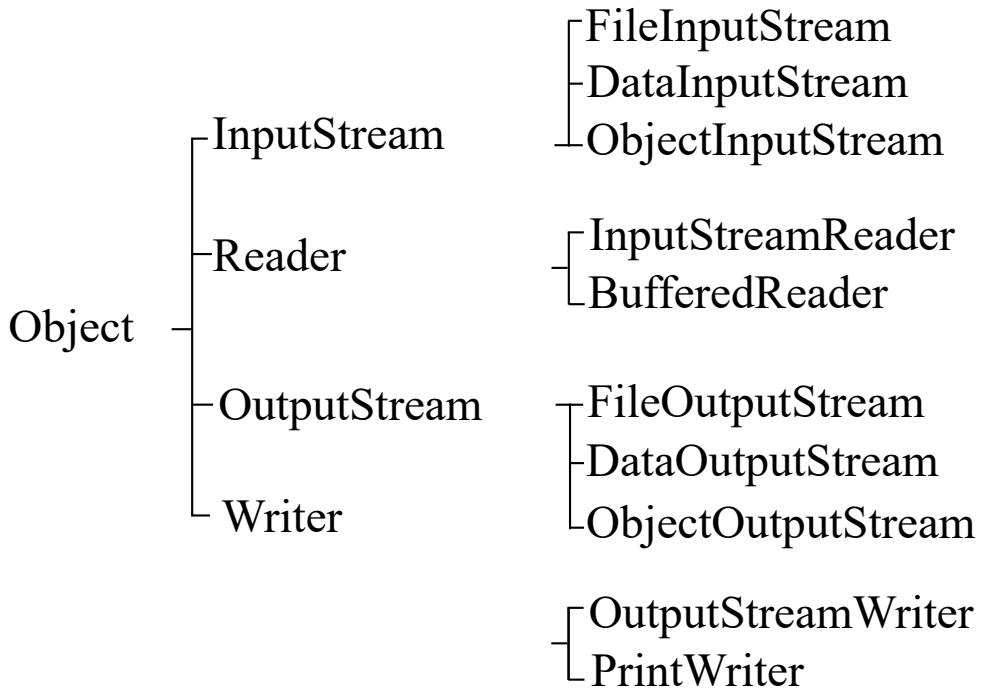
Un objet peut ainsi être facilement sauvegardé dans un fichier ou transféré sur le réseau.

La classe de l'objet qu'on veut sérialiser doit implémenter l'interface *java.io.Serializable*. Si tous les attributs sont de type primitif ou sont des objets sérializables, il n'y a rien d'autre à faire.

Ex.2 Sérialisation (15 min)

- Téléchargez les classes **`Message`** et **`EcrireMessage`**.
- Compilez et tapez la commande **java EcrireMessage msg.ser**. Que se passe-t-il ?
- Ecrire une classe exécutable appelée *LireMessage* qui, en utilisant la méthode *readObject* de *ObjectInputStream*, lit le message stocké dans le fichier *msg.ser*, puis l'affiche sur la sortie standard.

Principales classes à connaitre



Le package *java.net*

Description

Le package *java.net* fournit un ensemble de classes pour l'implémentation d'applications réseau comme les adresses, modélisant des adresses IP, ou les *sockets*, modélisant les extrémités d'un canal de communication bidirectionnelle entre deux processus via le réseau.

Consultez le tutoriel réseau sur les fonctionnalités réseau ainsi que l'API standard.

SocketServer et *Socket*

La classe *SocketServer* permet au processus *serveur* d'attendre et d'accepter la connection d'un processus *client*. La méthode *accept()* retourne un objet de type *Socket* lorsque la connection est établie.

```
ServerSocket connection = new ServerSocket(numeroPort);
```

```
Socket socket = connection.accept();
```

La classe *Socket* modélise une extrémité du canal de communication entre les deux processus. Côté client, un objet de type *Socket* peut être créé à partir d'une adresse IP et d'un numéro de port.

```
Socket socket = new Socket("localhost", numeroPort);
```

Flux d'entrée et de sortie

La méthode *getInputStream()* renvoie un flux d'entrée...

```
InputStream is = socket.getInputStream();
```

...tandis que la méthode *getOutputStream()* retourne un flux de sortie.

```
OutputStream os = socket.getOutputStream();
```

Ces flux peuvent être décorés.

Ex.3. Serveur (15 min)

- Ecrivez une classe *Serveur* écrivant sur la sortie standard les lignes de textes envoyées par le client. Comme vous devez procéder ligne par ligne, vous allez envelopper le flux d'entrée dans un *BufferedReader* comme dans *l'exemple précédent*.
- Testez votre serveur avec telnet: `telnet localhost 8080`

Ex.4. Client (15 min)

- Ecrivez une classe *Client* qui envoie des lignes de texte lues sur l'entrée standard au serveur. Comme vous devez procéder ligne par ligne, vous allez envelopper le flux de sortie dans un *PrintWriter* comme dans *l'exemple précédent*. Cependant, activez le *flush* automatique avec l'argument supplémentaire *true*:

```
PrintWriter out = new PrintWriter(
  new OutputStreamWriter( socket.getOutputStream() ), true );
```

- Pour aller plus loin, dans la classe *Serveur*, redirigez le texte envoyé par le client vers ce dernier comme un echo.