

Introduction à la programmation en Java

Cours 6

Stéphane Airiau

Université Paris-Dauphine

Entrée et sortie

Entrée/sortie : échange de données entre le programme et une source :

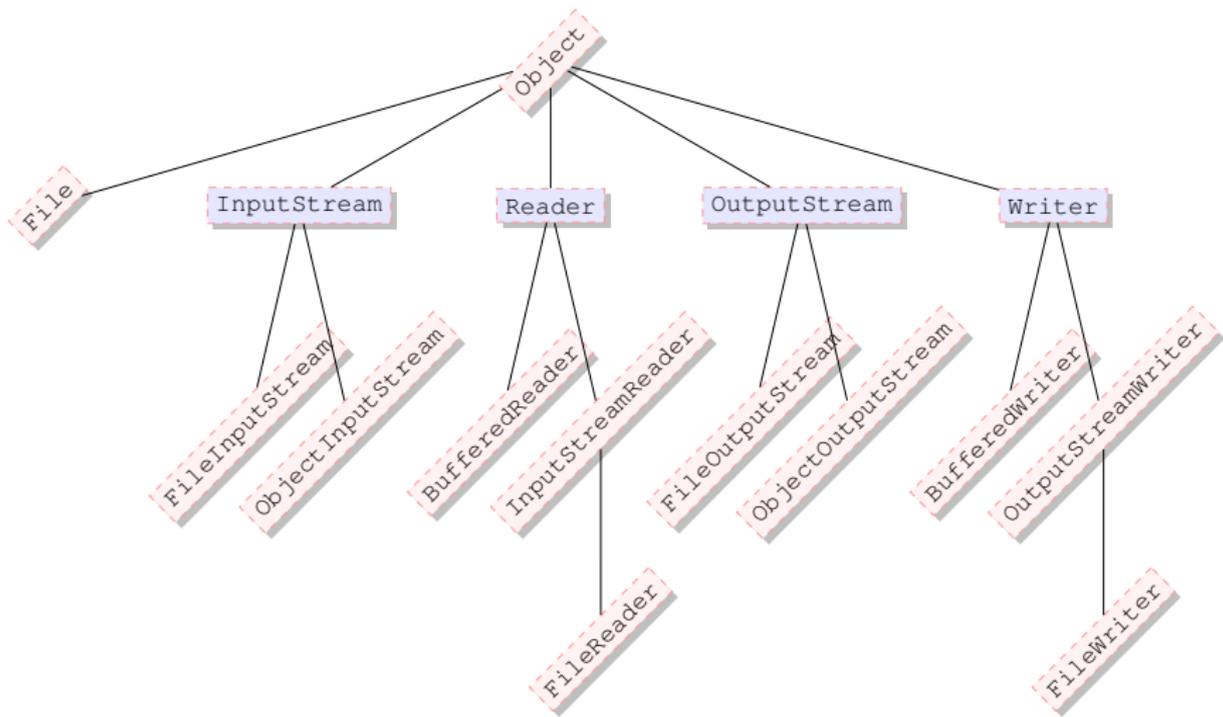
- entrée : au clavier, lecture d'un fichier, communication réseau
- sortie : sur la console, écriture d'un fichier, envoi sur le réseau

⇒ Java utilise des flux (stream en anglais) pour abstraire toutes ses opérations.

de manière générale, on observera trois phases :

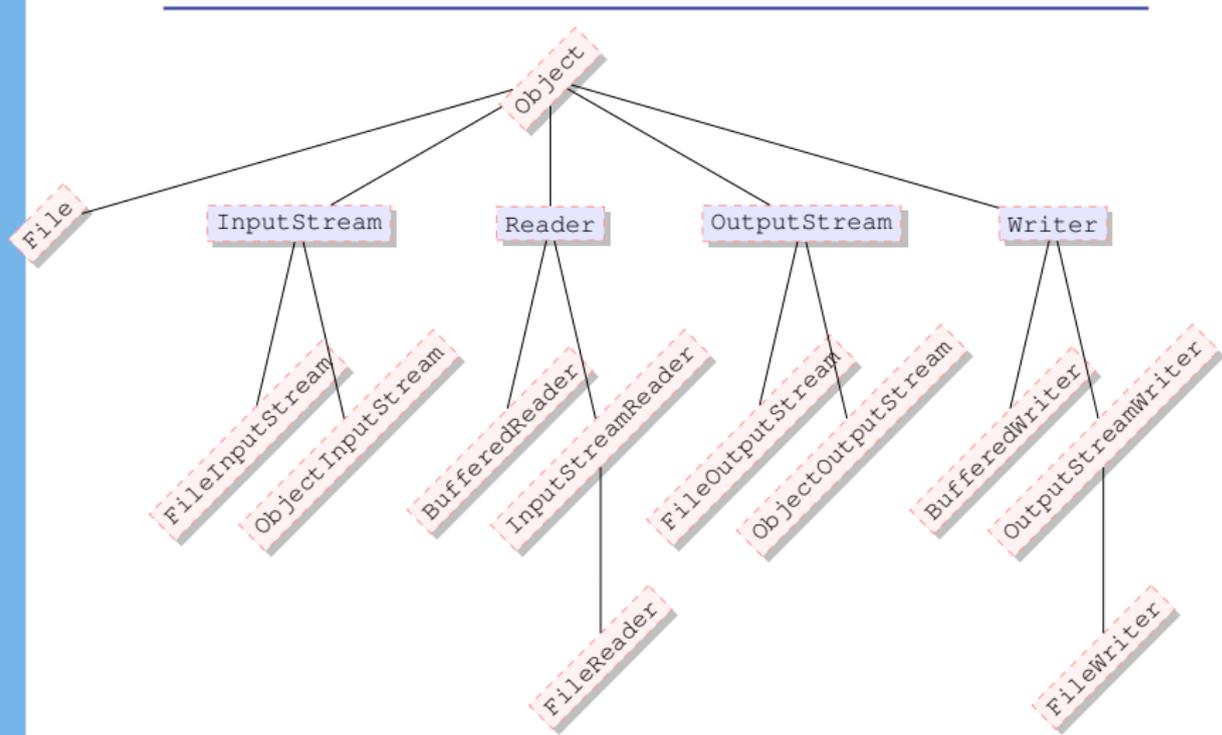
- 1- ouverture du flux
- 2- lecture/écriture du flux
- 3- fermeture du flux

le package java.io (fragment)



La classe `Files` permet de manipuler des fichiers et répertoire

- nom, chemin absolu, répertoire parent
- s'il existe un fichier d'un nom donné en paramètre
- droit : l'utilisateur a-t-il le droit de lire ou d'écrire dans le fichier
- la nature de l'objet (fichier, répertoire)
- la taille du fichier
- obtenir la liste des fichiers
- effacer un fichier
- créer un répertoire
- accéder au fichier pour le lire ou l'écrire



Flux

Les flux transportent des `bytes` ou des `char`.

Direction du Flux :

- objets qui gèrent des flux d'entrée : **in**
 - `InputStream`, `FileInputStream`, `FileInputStream`
- objets qui gèrent des flux de sortie : **out**
 - `OutputStream`, `FileOutputStream`, `FileOutputStream`

Source du flux :

- **fichiers** : on pourra avoir des flux vers ou à partir de fichiers
 - `FileInputStream` et `FileOutputStream`
- **objets** : on pourra envoyer/recevoir un objet via un flux
 - `ObjectInputStream` et `ObjectOutputStream`

Obtenir un flux : le plus facile, avec une méthode **static**.

- flux de ou vers un **fichier**

```
InputStream in = Files.newInputStream(path);  
OutputStream out = Files.newOutputStream(path);
```

path est un chemin sur votre ordinateur.

```
Path p = Paths.get("/", "home", "stephane");
```

La méthode `get` de la classe `Paths` va former un chemin avec les chaînes passées en argument (et va séparer avec le séparateur pour les répertoire) et va donner un objet de type `Path`.

Il y a des choses pour regarder les chemins relatifs ou absolus.

- flux de ou vers **internet**

```
URL url = new URL("http://www.lamsade.dauphine.fr");  
InputStream in = url.openStream();
```

- venant d'un tableau de **byte**

```
byte[] bytes = ...  
InputStream in = new ByteArrayInputStream(bytes);
```

Lire un flux de bytes

- `read()` lit un seul byte! (retourne `-1` si la fin de l'output n'a pas été atteinte)
- `readAllBytes()` lit toutes les bytes possibles et les place dans un tableau

```
byte[] tab = in.readAllBytes();
```

Pour un fichier, on a directement

```
byte[] tab = Files.readAllBytes(path);
```

- lire quelques bytes `read(byte[], int, int)` ou `readNBytes(byte[], int, int)` `readNbytes` va attendre d'avoir lu n bytes, alors que `read` va retourner moins de bytes s'il échoue

Ecrire un flux de bytes

- écrire un seul byte

```
int b = 17;  
out.write(b);
```

- écrire tous les éléments d'un tableau de bytes

```
byte[] tab = ...  
out.write(tab);  
out.write(tab, start, length);
```

- il existe pas mal de méthodes utilitaires : par exemple, transférer depuis un fichier

```
Files.copy(path, out);
```

Un mot sur l'encodage

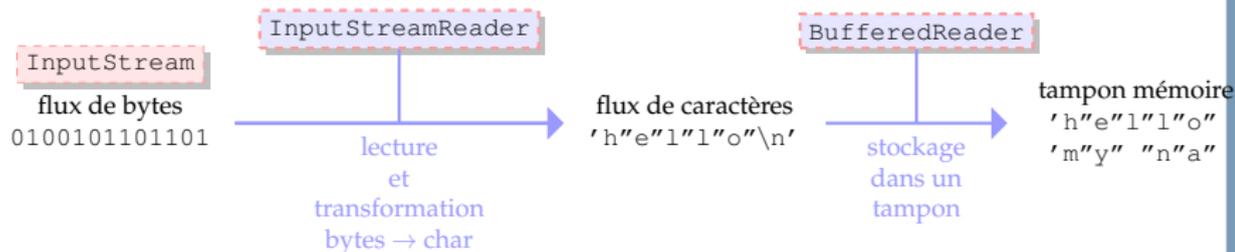
comment les caractères sont encodés en bytes ?

- java utilise le standard Unicode UTF-8, UTF-16 encode les "points code" en bit.
Java utilise UTF-16 qui utilise un ou deux mots de 16bit chacun.
- quelques interprétations différentes du standard. Quelques programmes (dont MS NotePad) ajoute un byte en début d'un fichier pour aider à décoder, mais Java ne le prend pas en compte...
- il n'y a pas de moyen fiable de reconnaître un encodage
⇒ essayer de bien spécifier l'encodage (ex : Content-Type dans le header d'une page web)
- classe Charset

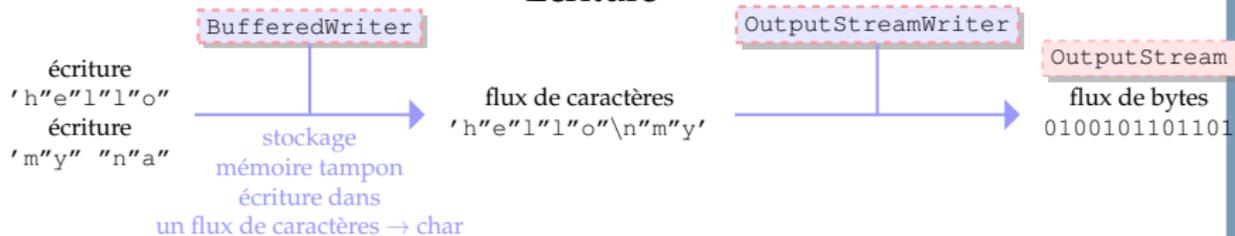
```
String s = new String(bytes, StandardCharsets.UTF_8);
```

Processus de lecture et d'écriture

Lecture



Écriture



Selon le type de la source ou de la destination (fichier, objet), on utilisera

- `FileReader` à la place de `InputStreamReader`
- `FileOutputStream` ou `ObjectOutputStream` comme implémentation de la classe abstraite `OutputStream`

```
Reader reader = new InputStreamReader(inStream, charset);
```

- lire les symboles un par un

```
int s = in.read();
```

pas très utilisé en pratique

- lire tout un texte

```
String contenu = new String(Files.readAllBytes(path), charset);
```

- recupérer une séquence de ligne

```
List<String> lines = Files.readAllLines(path, charset);
```

N.B. On expliquera ce `List<String>` bientôt, c'est juste une liste contenant des chaînes de caractères.

Lire du Texte

Pour lire des nombres, utilisez la classe `Scanner`

```
Scanner scan = new Scanner(path, "UTF-8");  
double value = scan.nextDouble();
```

Pour lire du texte petit à petit, on peut utiliser un `BufferedReader`.
En particulier, la classe possède une méthode `readLine()` pour lire ligne par ligne.

On utilise un `Writer`

```
OutputStream out = ...  
Writer write = new OutputStreamWriter(out, charset);  
out.write("We all live in a yellow submarine");
```

- pour écrire dans un fichier

```
Writer write = Files.newBufferedWriter(path, charset);
```

- Il est plus pratique d'utiliser `PrintWriter` car il contient des méthodes familières

`print`, `println`, `printf`

- pour écrire dans un fichier

```
PrintWriter writer =  
    new PrintWriter(Files.newBufferedWriter(path, charset));
```

- pour écrire dans un autre flux

```
PrintWriter writer =  
    new PrintWriter(new OutputStreamWriter(out, charset));
```

On peut utiliser des expressions régulières pour lire ou chercher du texte.

ici, on donne un seul exemple pour découper du texte avec `split`, une méthode de la classe `String`.

```
String line = ...  
String[] parts = line.split(";")
```

cette line va mettre dans un tableau de chaîne de caractères les morceaux délimités par `;`.

But : envoyer toute l'information d'un objet

↳ mécanisme de « sérialisation »

- la classe doit implémenter l'interface `Serializable`
- l'interface `Serializable` n'a pas de méthodes : c'est juste un marqueur.
- Java transforme l'objet automatiquement en un code pas lisible pour les humains

NB : Si un attribut de la classe est un objet d'une classe `MaClasse`

- `MaClasse` est « sérialisable » : ✓
- `MaClasse` n'est pas « sérialisable » : on peut utiliser le mot-clé `transient` pour indiquer de ne pas enregistrer cet attribut

Exemple

```
1 IrreducibleGaulois panoramix =
2     new IrreducibleGaulois("Panoramix", 1.75);
3
4 ObjectOutputStream oos =
5     new ObjectOutputStream(
6         new FileOutputStream(
7             new File("panoramix.txt")));
8
9 oos.writeObject(panoramix);
10 oos.close();
11
12 ObjectInputStream ois =
13     new ObjectInputStream(
14         new FileInputStream(
15             new File("panoramix.txt")));
16
17 IrreducibleGaulois copyPanoramix =
18     (IrreducibleGaulois) ois.readObject();
19 System.out.println(copyPanoramix.nom);
20 ois.close();
```

N.B. Le code n'est pas correct (gestion des exceptions)

Lire depuis la console, afficher sur la console

- `System.in` :
 - entrée « standard »
 - objet de type `InputStream`
- `System.out` :
 - sortie « standard »
 - objet de type `PrintStream` qui hérite de `OutputStream`

La classe `Scanner` permet de récupérer ce que vous tapez

```
1 Scanner scan = new Scanner(System.in);  
2 int n = scan.nextInt();  
3 double x = scan.nextDouble();  
4 String s = scan.nextLine();
```

Exceptions et entrée/sortie

```
1  try {
2      FileInputStream fis = new FileInputStream(new File("test.txt"));
3      byte[] buf = new byte[8];
4      int nbRead = fis.read(buf);
5      System.out.println("nb bytes read: " + nbRead);
6      for (int i=0;i<8;i++)
7          System.out.println(Byte.toString(buf[i]));
8      fis.close();
9
10     BufferedReader reader =
11         new BufferedReader(new FileReader(new File("test.txt")));
12     String line = reader.readLine();
13     while (line!= null) {
14         System.out.println(line);
15         line = reader.readLine();
16     }
17     reader.close();
18 } catch (FileNotFoundException e) {
19     e.printStackTrace();
20 }
21 catch (IOException e) {
22     e.printStackTrace();
23 }
```