Les entrées/sorties en Java

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-10-31

Table des matières

1 Utilisation des Flots (streams) 1
2 Sources ou destinations de flots 2
3 Le paquetage java.io 2
4 Les grands types de flots 2
5 Les types de classes 2
6 Les classes de base du paquetage java.io 2
7 Les sources et destinations concrètes 3
8 Les décorateurs 3
9 Ecrire des primitifs dans un fichier 3
10 Lire des primitifs dans un fichier 4
11 Les exceptions liées aux entrées-sorties 4
12 Reader (Classe abstraite de base) 4
13 Séparateur de lignes 5
14 Codage 5
15 La Classe ~~File~~ Path 5
16 Les fonctionnalités de la classe File 6
17 Classe URL 6

1 Utilisation des Flots (streams)

- En Java, la communication entre le programme et les échanges de données entre un programme et l'extérieur (autre programme, fichier ou application réseau, ...) sont réalisées à travers un « flot « de données
- Un flot permet de transporter séquentiellement des données. Les données sont transportées une par une (ou bloc) et dans l'ordre
- Le cycle d'utilisation d'un flot de données est le suivant :
  1. Ouverture du flot
  2. Tant qu'il y a des données à lire (ou à écrire), on lit (ou on écrit) la donnée suivante dans le flot

3. Fermeture du flot

2 Sources ou destinations de flots

- Fichier
- Socket pour échanger des données sur un réseau
- Données de grandes tailles dans une base de données (blob) (images, par exemple)
- Pipe entre 2 processus
- Tableau d'octets (en mémoire)
- Chaîne de caractères
- URL (adresse Web)
- ...

3 Le paquetage java.io

- La plupart des classes liées au E/S sont définies dans le paquetage java.io
- Il prend en compte un grand nombre de flots :
  - On distingue deux grands types de flots :
    - Les octets
    - Les caractères
- différentes sources et destinations
- Il est possible de décorer les flots : le modifier au fur et à mesure
- Le nombre de classes de ce paquetage est important, il faut regarder la javadoc.

4 Les grands types de flots

- Les flots d'octets servent à lire ou écrire des octets « bruts «, qui représentent des données codées, manipulées par un programme
- Les flots de caractères servent à lire ou écrire des données qui représentent des caractères lisibles par un homme (codés en Unicode)

5 Les types de classes

- Pour les deux grands type de flots (octets et caractères), on distingue :
- Les classes associées à une source ou une destination « concrète «
  - FileReader pour lire un fichier
- Les classes qui permettent de « décorer « un autre flot
  - BufferedReader qui ajoute un tampon (buffer) pour lire un flot de caractères

6 Les classes de base du paquetage java.io

- InputStream (Lecture d'octets)
- OutputStream (Ecriture d'octets)
- Reader (Lecture de caractères Unicode)

- Writer (Ecriture de caractères Unicode)
- File et Path (Manipulation de noms de fichiers et de répertoires)
- StreamTokenizer (Segmentation lexicale d'un flot d'entrée)

7 Les sources et destinations concrètes

- Fichiers :
  - File{In|Out}putStream
  - File{Reader|Writer}
- Tableaux
  - ByteArray{In|Out}putStream
  - CharArray{Reader|Writer}
- Chaînes de caractères
  - String{Reader|Writer}

8 Les décorateurs

- Pour ajouter un tampon sur une entrée/sortie :
  - Buffered{In|Out}putStream
  - Buffered{Reader|Writer}
- Pour lire et écrire des types primitifs sous une forme binaire :
  - Data{In|Out}putStream
  - Pour compter les lignes lues :
    - LineNumberReader
- Pour écrire sous forme de chaînes de caractères :
  - PrintStream, PrintWriter
- Pour permettre de remettre un caractère lu dans le flot :
  - PushbackInputStream, PushbackReader

9 Ecrire des primitifs dans un fichier

```
try (DataOutputStream dos=
  new DataOutputStream(
    new BufferedOutputStream(
      new FileOutputStream("monFichier.bin")))) {
 dos.writeDouble(27.7);
 dos.writeUTF("Pierre");
 dos.writeInt(56);
}
```

10 Lire des primitifs dans un fichier

```
try (FileInputStream fis = new FileInputStream("monFichier.bin");
     BufferedInputStream bis = new BufferedInputStream(fis);
     DataInputStream dis = new DataInputStream(bis)) {

    double d = dis.readDouble();
    String s = dis.readUTF();
    int i = dis.readInt();
    System.out.println("%f %s %d".formatted(d,s,i));
}
```

```
try (DataInputStream disBis = new DataInputStream(
        new BufferedInputStream(new FileInputStream("monFichier.bin")))) {
        int i = disBis.readInt();
        System.out.println("%d".formatted(i));
}
```

11 Les exceptions liées aux entrées-sorties

- Exceptions
- IOException (Exception durant une entrée-sortie)
- EOFException (Lecture d'une fin de fichier)
- FileNotFoundException (Fichier n'existe pas)
- ObjectStreamException (Problème lié à la sérialisation)

```
try (DataInputStream dis = new DataInputStream(
        new BufferedInputStream(new FileInputStream("monFichier.bin")))) {
    while (true) {
        double d=dis.readDouble();
    }
} catch ( FileNotFoundException e ) { System.out.println("File not found."); }
  catch ( EOFException e ) { System.out.println("End of file.");}
  catch ( IOException e ) { System.out.println("IO Exception");}
  finally { System.out.println("Done.");}
```

12 Reader (Classe abstraite de base)

- InputStreamReader (Lecture de caractères Unicode à partir d'un flot d'octets)
- FileReader (Lecture de caractères Unicode à partir des octets d'un fichier)
- FilterReader (Décorateur)
- BufferedReader (Entrées bufférisées)

13 Séparateur de lignes

- La façon de séparer les lignes dépend du système d'exploitation
- Pour être portable utiliser
  - println de PrintWriter (le plus simple)
  - writeLine ou newLine de BufferedWriter
  - ou la propriété système line.separator (System.getProperty(« line.separator »))
- Ne pas utiliser le caractère " n" qui ne convient pas, par exemple, pour Windows

14 Codage

- En Java les caractères sont codés en Unicode
- Des classes spéciales permettent de faire les traductions entre le codage Unicode et un autre codage
- Un codage par défaut est automatiquement installé par le JDK, conformément à la locale

```
try (BufferedReader br = new BufferedReader(new FileReader("monFichier.txt"))) {
  String ligne;
  while ((ligne=br.readLine()) =≠ null ) {
      System.out.println(ligne);
    }
} catch ( FileNotFoundException e ) { System.out.println("file not found:
%s".formatted(e.getMessage()));}
```

```
try ( PrintWriter pw = new PrintWriter(
        new BufferedWriter ( new FileWriter("autreFichier.txt")),true) ) {
    String lignes[]={"e:echo","f:foxtrot"};
    Arrays.stream(lignes).forEach(pw::println);
}
```

15 La Classe ~~File~~ Path

- **Utiliser Path à la place**
- Cette classe représente la notion de fichier, indépendamment du système d'exploitation
- Un fichier est repéré par un chemin abstrait composé d'un préfixe optionnel (nom d'un disque par exemple) et de noms (noms des répertoires parents et du fichier lui-même)
- Attention, File fichier = new File(« /bidule/truc »); ne lève aucune exception si / bidule/truc n'existe pas dans le système de fichier
- Constructeurs

- Les chemins passés en premier paramètre peuvent être des noms relatifs ou absolus
  - File (String chemin)
  - File (String cheminParent, String chemin)
  - File (File parent, String chemin)
- La classe File offre des facilités pour la portabilité des noms de fichiers

16 Les fonctionnalités de la classe File

- Elle permet d'effectuer des manipulations sur les fichiers et répertoires considérés comme un tout (mais pas de lire ou d'écrire le contenu) :
  - lister un répertoire,
  - supprimer, renommer un fichier
  - créer un répertoire
  - créer un fichier temporaire connaître les droits que l'on a sur un fichier (lecture, écriture)
  - etc.

```
Path path = Paths.get("autreFichier.txt");
BufferedReader reader = Files.newBufferedReader(path);
```

```
Files.lines(Paths.get("autreFichier.txt"))
    .map(line →line.split(":"))
    .map(t → "La lettre %s se dit %s.".formatted(t[0],t[1]))
    .forEach(System.out::println);
```

StringTokenizer et StreamTokenizer

paquetage java.nio Reprend toute l'architecture des classes pour les entrées/sorties notion de canal (channel) et de buffer Utilise les possibilités avancées du système d'exploitation hôte pour optimiser les entrées-sorties et offrir plus de fonctionnalités

17 Classe URL

- protocole://machine[:port]/cheminPage
- Cette classe du paquetage java.net fournit de nombreux constructeurs
- Elle permet d'extraire des éléments à partir d'un URL (getPort, getHost,...)
- Les données associées à l'URL peuvent être obtenues par les méthodes
  - openConnection et openStream ou par la méthode getContent

---

```
URL url = new URL("https://www.univ-tln.fr/");
BufferedReader br=new BufferedReader ( new InputStreamReader(url.openStream()));
br.lines().filter(l->l.contains("<title>")).forEach(System.out::println);
```

```
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

Document doc = Jsoup.connect("https://en.wikipedia.org/").get();
System.out.println(doc.title().toString().toUpperCase());
Elements newsHeadlines = doc.select("#mp-itn b a");
for (Element headline : newsHeadlines) {
  System.out.println("%s\t\t\t\t%s".formatted(
    headline.attr("title"), headline.absUrl("href")));
}
```