Java - 1ère année

Entrées-sorties, fichiers

Cours 10

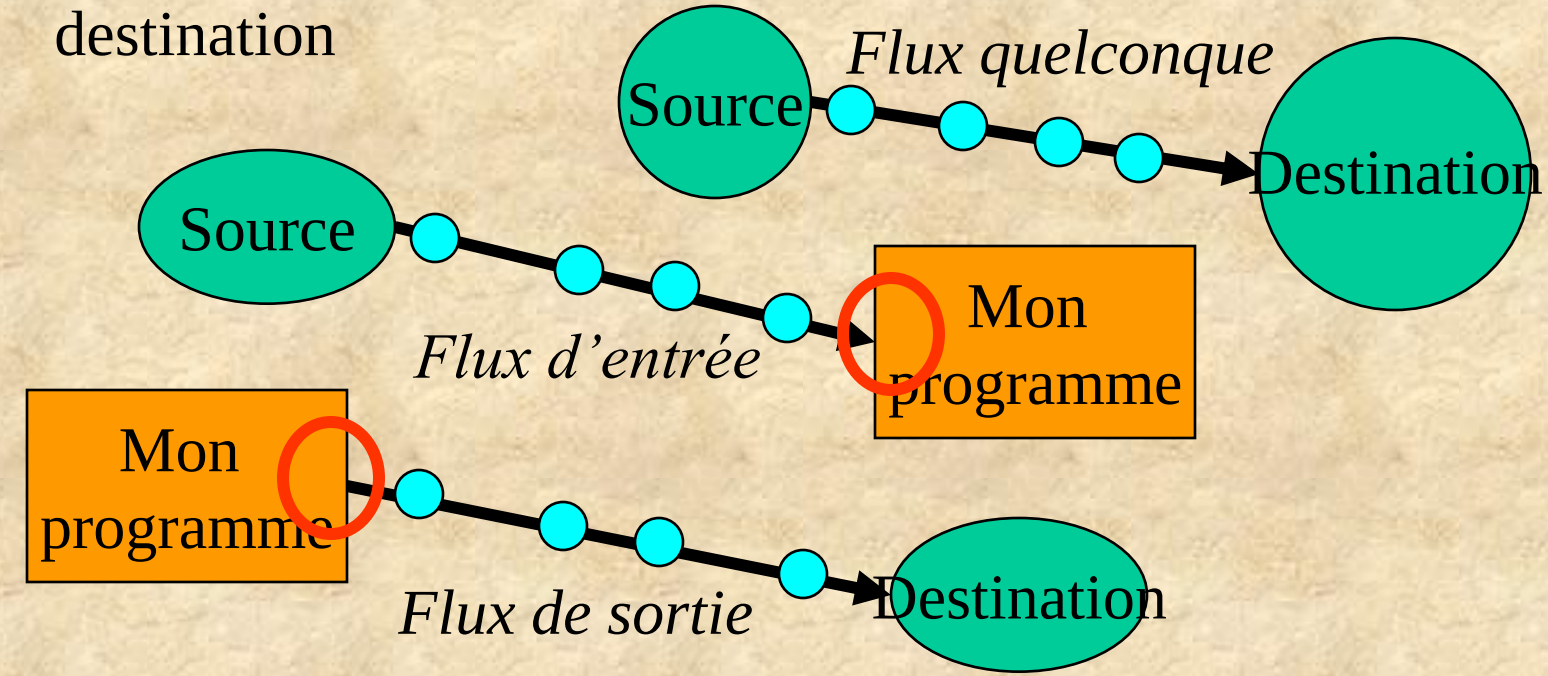
E/S

En Java, les échanges de données entre un programme et l'extérieur (autre programme, fichier, réseau) sont effectués par un « flot ou flux (streams en anglais) » de données

- Un flot permet de transporter (le plus souvent) séquentiellement des données. Les données sont transportées une par une (ou groupe par groupe), de la première à la dernière donnée

Flux d’entrées/sorties

- Un flux d’entrées/sorties (IO Stream) représente un canal par lequel circule des données entre une source et une destination



Sources ou destinations de flots

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

  etc...

Flux d’entrées/sorties

- Les flux **séquentiels** : Ces flux peuvent être associés à des fichiers, des tableaux, ou des connexions réseau. Deux «styles» de flux séquentiels :
  - les flux d’octets en lecture et en écriture (*InputStream*, *OutputStream*).
  - les flux de caractères en lecture et en écriture (*Reader*, *Writer*).

- Les flux à **accès direct** *RandomAccessFile* : ces flux sont forcément associés à des fichiers.

Flux d’entrées/sorties

- Un flux d’entrées/sorties (IO Stream) peut véhiculer toutes sortes de données, brutes (binaire octet), structurées, formatées etc. On distingue :
  - Les flux octet (base de construction des autres flux)
  - Les flux de données
  - Les flux de texte
  - Les flux d’objets

Flux d’octet ou flux binaires

- Un flux binaire sert de base aux autres flux.
- Il transfère des données octet par octet. Le sens de ces données lui est indifférent.
- Un flux est,
  - physiquement, un mécanisme technique de l’OS mis au service d’un programme qui l’utilise ;
  - vu du programme, comme un objet d’une certaine classe. Vous voyez le flux comme une instance.

Flux d’octet ou flux binaires

- Utiliser un flux :
  - On construit une instance d’un flux.
  - On utilise le flux en y lisant (entrée) ou en y écrivant (sortie).
  - On referme un flux.
- Peut-on utiliser plusieurs flux dans un même programme ?
  - Oui, en fabriquant plusieurs instances distinctes d’un flux.

Flux d’entrée clavier

- Pour lire un caractère au clavier :
- System.in.read();
- qui renvoie une valeur de type byte que l’on convertira en caractère par un opérateur cast.
  - char c = (char)System.in.read();

Flux d’octet ou flux binaires

- Les classes :
  - Flux en entrée : InputStream => read(...)
  - Flux en sortie : OutputStream => write(...)
- Lecture/écriture par octet(s) => byte ou byte[]
- Que peut-il arriver ?
  - Lis-je toujours dans un flux valide (=> exception)
  - Si je lis, ai-je quelque chose à lire ? (=> blocage)
  - Ecris-je toujours dans un flux valide (=> exception)

Flux de caractères

- Le problème de l'encodage
  - Un seul caractère plusieurs codes possibles
  - Encodage sur 8 bits : 256 caractères possibles
  - Jeu ASCII : 0 à 127 : caractères américains et typographiques de base
  - 128 à 255 : extensions différentes selon les pays (Eastern Europe : Tchéquie, Slovaquie Pologne, Western Europe : France, Espagne, Langues scandinaves)
- L'Unicode
  - Présente une possibilité multi-octets
  - L'Unicode 16 : 16 bits d'encodage => 65636 caractères

Flux de caractères

- Le flux de caractère Java :
  - s’appuie sur le flux d’octets
  - adapte le jeu de caractère local du système d’exploitation
  - gère le problème de la fin de ligne (problème principal du texte)

Flux de caractères

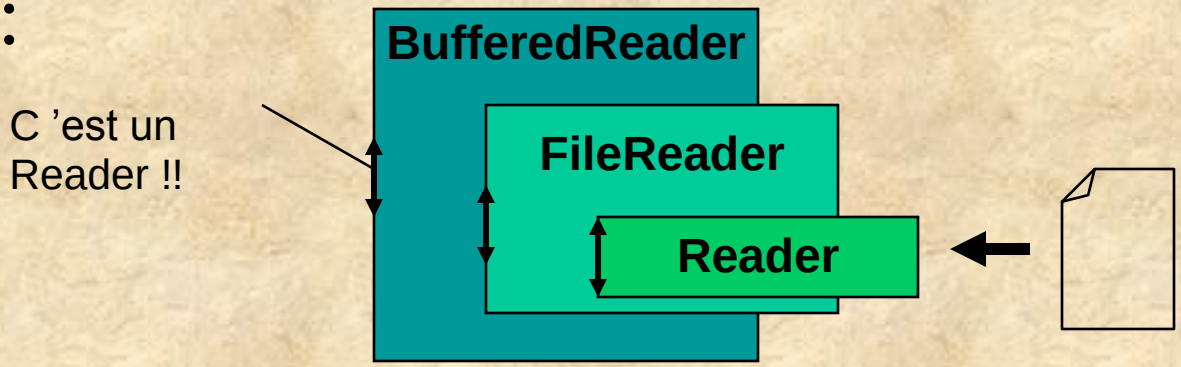
- Deux classes de base
  - Reader
  - Writer
  - Toutes les applications qui en découlent s'y appuient :
    - ex : « écrire dans un fichier » : FileWriter, « lire un texte filtré » : FilterReader, etc.

Flux de caractères

- Des applications :
  - Par « support » : FileReader, StringReader
  - Par fonctionnalité : PipedReader, FilterReader, PushbackReader, etc.
  - BufferedReader/Writer permet d'obtenir une « unité de lecture » à la ligne de texte et non au caractère.

Flux de caractères

- Ces classes appliquées se combinent comme des Légo :

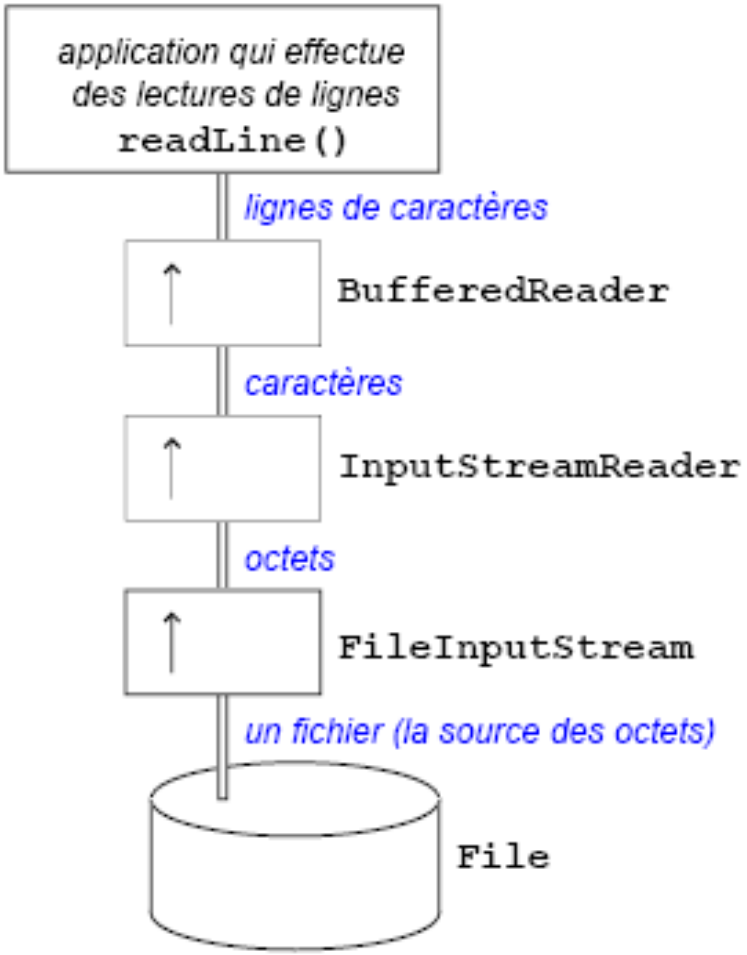


Exemple :

```
BufferedReader monFlux =
    new BuffererReader(new FileReader("monFicher.txt"));
```

Le résultat est un « BufferedFileReader » implicite.

Flux de données



Ce qui, dans le code, donne :

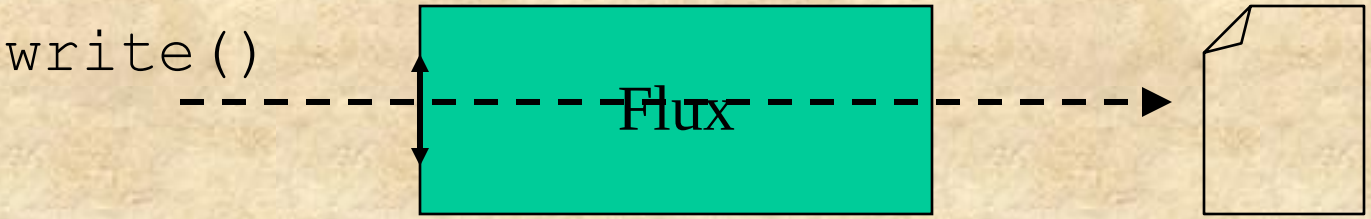
```
File f = new File("mydata.txt");
FileInputStream fis =
            new FileInputStream(f);
InputStreamReader isr =
            new InputStreamReader(fis);
BufferedReader br =
            new BufferedReader(isr);
```

ou, en version compacte :

```
BufferedReader br =
    new BufferedReader(
        new InputStreamReader(
            new FileInputStream(
                new File(
                    "mydata.txt"))));
```

Flux bufferisés et non bufferisés

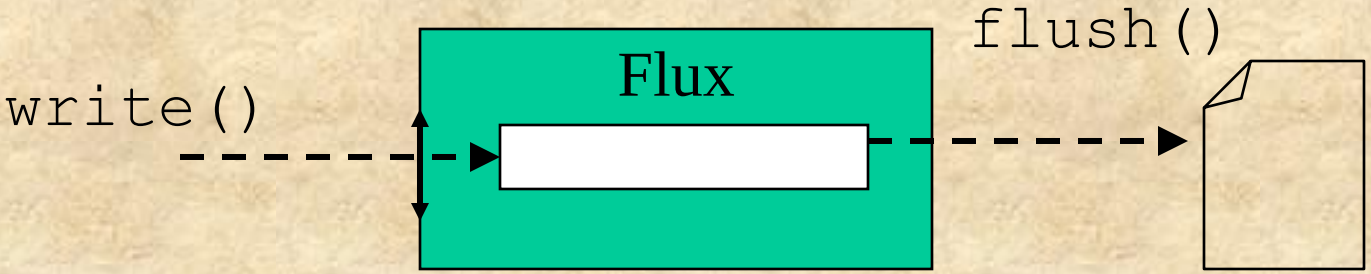
- Flux non bufferisé :



  - chaque interaction avec le stockage est directe
  - performances limitées
  - peu de risques

Flux bufferisés et non bufferisés

- Flux bufferisé :



  - une interaction n'est pas répercutée immédiatement sur le stockage
  - performances améliorées
  - risque de pertes

Flux bufferisés et non bufferisés

- Moments du flush :
  - Write : le tampon est plein
  - Write : le flux est refermé
- Autoflush : certains événements en plus
- Pour la lecture c'est le procédé inverse : prefetch (anticipation de lecture)
  - Read : le tampon est vide et on demande une lecture

Formater et Parser

- Deux activités quotidienne de l'informatique actuelle
  - L'informatique est passée du nombre au texte

« Formater, c'est produire un texte mis en forme à partir d'une certaine modélisation. »

« Parser, c'est reconnaître une syntaxe formelle dans du texte brut. »

Formatage

- Le formatage, c'est la fabrication d'une expression textuelle (lisible ou non) exprimant une certaine valeur.
- Le formatage dit :
  - comment on exprime la valeur
  - quel est l'habillage de sortie autour de la valeur.

Exemple : Pour sortir la température d'un microprocesseur :

La chaîne à produire est : « Processor temp : 98.3° F » basée sur un motif de génération : `"Processor temp : %d° F\n"`

Formatage

- Les superclasses génératrices des formateurs sont `PrintWriter` et `PrintStream`.
  `System.out` est une instance de ce type.
- `print()` et `println()` : impression avec un formatage par défaut basé sur la conversion standard `toString()`.
- `format()` et `printf()` : impression avec un formatage basé sur une chaîne de format (standard POSIX) que l'on trouve dans presque tous les langages (format du C).

Formatage

- Les instructions de formatage prennent en charge :
  - La forme d'expression du nombre
  - La précision du nombre et la longueur d'affichage
  - Le remplissage des chiffres non significatifs
  - L'affichage du signe
  - L'indexation des variables source
  - L'adaptation du retour ligne -> `"%n"` plutôt que `"\n"`

Formatage des nombres

- Le package java.text contient toutes sortes de méthodes de formatages
  - Classe NumberFormat se trouve dans ce package

Fixer le nombre de décimales

```
import java.text.*;
public class Formatage {
    public static void main (String[] args) {
        double d = 5664324563.576927654;
        NumberFormat formateur = new DecimalFormat("0.###");
        String nbr = formateur.format(d);
        System.out.println(nbr);
        }
    }
```

Formater d

Indique le nombre de décimal après la virgule

Résultat : 5664324563.577

Ajouter des virgules

```
import java.text.*;
public class Formatage2 {
    public static void main (String[] args) {
        int d = 566432456;
        NumberFormat formateur = NumberFormat.getNumberInstance();
        System.out.println(formateur.format(d));

        }
    }
```

Sortie: 5,664,324,56

Classe NumberFormat fournit une instance qui sait formater des nombres (ajouter des virgules)

Parsing

- Tokenisation
  - Activité basique du parsing
  - Séparer les constituants de base de la syntaxe à interpréter
  - Notion de séparateur
  - Produit une liste de « token »

Parsing

- Un tokeniser est un objet !!
  - On construit un « Scanner » (Java 5) avec ...
  - … un flux de texte à manger (Reader)
  - On configure les paramètres de fonctionnement
  - On l'utilise alors comme une liste :
    - hasNext() ?
    - next ()
  - Ancienne implémentation : StringTokenizer

Parsing

- Conversions
  - La conversion reconnaît une syntaxe « numérique littérale » dans du texte
  - "1.34e+24" est une chaîne. Elle ne correspond au nombre 1.34 10^{24} qu'après conversion
  - Un objet Scanner permet de reconnaître ces formes dans le flux « tokenisé ».
  - L'interprétation demande de connaître la « Locale »
  - une « Locale » est une classe dont les instances donnent les variantes typographiques et lexicales propres à un pays, une culture.

Entrées sorties console

- Les entrées sorties console fournit en Java les accès aux "flux standard" **`stdin`**, **`stdout`** et **`stderr`**.
- **`stdin`** : flux d'entrée (classe de base : InputStream). Instance unique accessible comme **`System.in`**.
- **`Stdout`** et **`stderr`** : flux de sortie orientés texte (classe de base : PrintStream). Instances uniques accessibles comme **`System.out`** et **`System.err`**.
- Depuis Java 6, un nouvel objet **`java.io.Console`**.

Flux de données

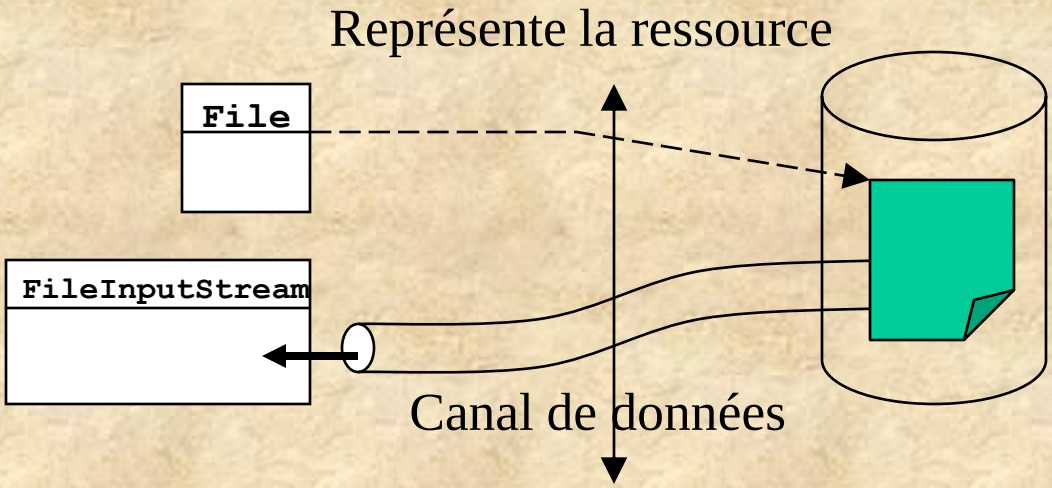
- Les flux de donnée prennent en charge les types de données constitués du Java par des méthodes de lecture/écriture dédiées.
- classe **`DataInputStream`** et **`DataOutputStream`**.
- Les données doivent être de **type primitif** ou de type String. On admet les types **wrappers** des types primitifs.
- L'écriture dans le flux est binaire.
- La lecture et l'écriture doivent correspondre. C'est au programmeur de prendre en charge cette cohérence.

Flux d’objets

- Les flux d’objet permettent de **sérialiser** des objets complexes.
- Une seule opération permet "d’écrire" dans un flux (fichier) tout ce qui constitue l’état d’un objet.
  - Elle écrit ses membres simples
  - Elle sérialise ses sous-objets => processus récursif
- Une opération de lecture permet de "lire" dans un flux (fichier) tout ce qui reconstitue l’état d’un objet.
  - Elle récupère ses membres simples
  - Elle récupère ses sous-objets => processus récursif

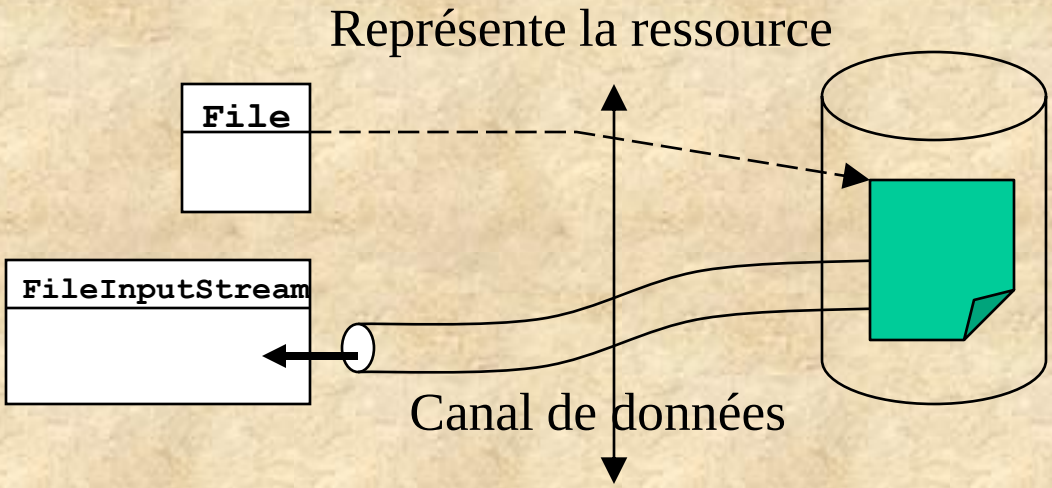
Fichiers

- Les fichiers sont manipulables à travers des flux. L'accès à un fichier est donc un flux.
- Un fichier est représentable par un objet.



Fichiers

- Les fichiers sont manipulables à travers des flux. L'accès à un fichier est donc un flux.
- Un fichier est représentable par un objet.



Fichiers

- Les répertoires sont représentables comme des fichiers.
- La classe File propose un jeu de méthodes statiques pour gérer l'aspect « répertoire » d'un certain chemin.

Exemple d'utilisation d'un objet file

- Créer un objet représentant un fichier existant
  - File f = new File(''Monprog.txt'');
- Créer un répertoire
  - File rep = new File(''JAVA'');
  - Rep.mkdir();
- Obtenir le chemin absolu d'un fichier ou d'un répertoire
  - System.out.println(rep.getAbsolutePath());
- …

Exemple: création d'un fichier binaire: « toto.dat »

```
import java.io.*;
import java.util.*;
public class Crsfic1 {
    public static void main(String args[]) throws IOException {
        String nomfich;
        Scanner sc = new Scanner(System.in);
        int n;
        DataOutputStream sortie = new DataOutputStream(
                                        new  FileOutputStream("toto.dat"));
        do {    System.out.print("donner un entier:");
                n = sc.nextInt();
                System.out.println("n="+n);
                if (n != 0) { sortie.writeInt(n); }
            } while (n != 0);
            sortie.close();
            System.out.println("******* Fin de création ********");
        }
}
```

Lecture d'un fichier binaire

```
import java.io.*;
public class Lecfic1 {
      public static void main(String args[]) throws IOException {
                String nomfich;
                int n=0;
                DataInputStream entree = new DataInputStream(
                                                        new FileInputStream("toto.dat"));
                boolean eof = false;
                while (!eof) {
                                try {
                                                n = entree.readInt();
                                    }
                                catch (EOFException e) {
                                                                eof = true;
                                 }
                                 if (!eof) System.out.println(n);
                }
                entree.close();
       }
}
```

Ecriture d'un fichier texte

- Classe FileWriter permet de manipuler un flux texte associé à un fichier
  - Méthodes permettent d'écrire des caractères, chaînes
  - Classe PrintWriter possède des méthodes plus élaborées (print, println…)

```
import java.io.*;
import java.util.*;
public class Crstxt1 {
    public static void main(String args[]) throws IOException {
        String nomfich;
        Scanner sc = new Scanner(System.in);
        int n;
        PrintWriter sortie = new PrintWriter(new FileWriter("toto.txt"));
        do {
            System.out.print("donner un entier:");
            n = sc.nextInt();

            if (n != 0) { sortie.println(n+ " a pour carre"+n*n); }
            } while (n != 0);
            sortie.close();
            System.out.println("******** Fin de création *********");
        }
}
```

Lecture d'un fichier texte

```
import java.io.*;

public class Lecftxt1 {
    public static void main(String args[]) throws IOException {
        String ligne;

        BufferedReader entree = new BufferedReader (new
    FileReader("toto.txt"));
        boolean eof = false;
        do {
                ligne = entree.readLine();
                if (ligne != null) System.out.println(ligne);
                } while (ligne != null);
                entree.close();
                }
        }
```

Résultats

- Fichier in.txt

50.0

3

10 60 item1

20 100 item2

30 120 item3

- Fichier out.txt

Item3

30.0

FIN DU COURS