2i002 Flux (surtout les fichiers)

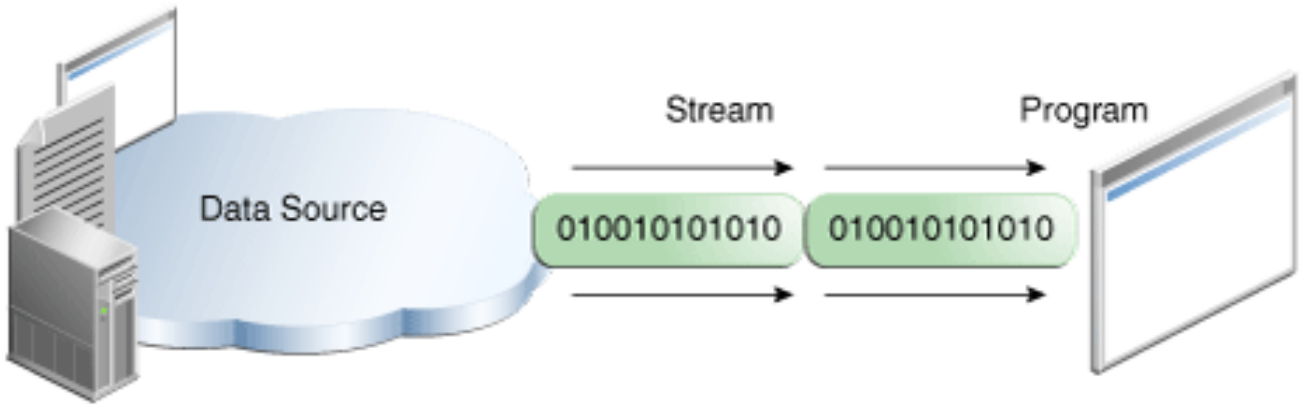
Vincent Guigue - vincent.guigue@lip6.fr

Le cours est inspiré de sources diverses: L. Denoyer, F Peschanski...
Illustrations et idées du tutoriel officiel:
http://docs.oracle.com/javase/tutorial/essential/io/

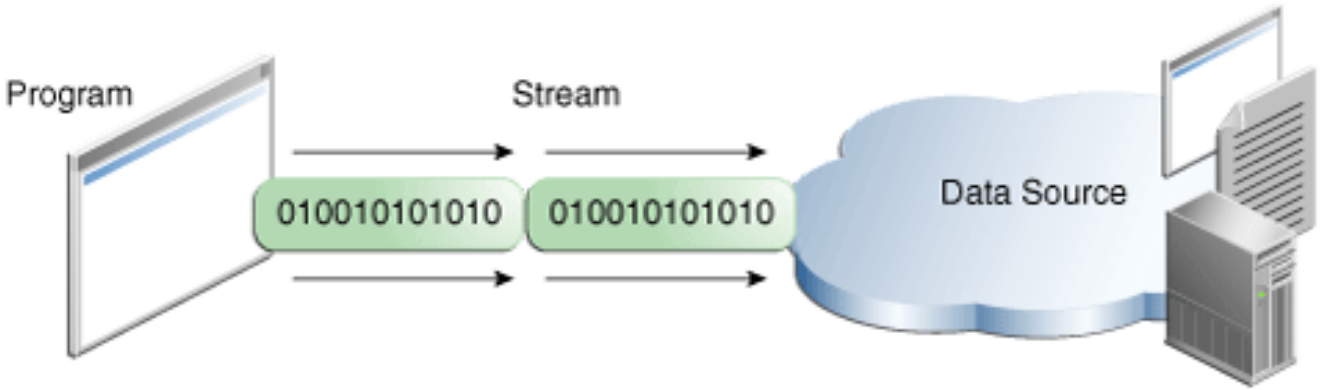
Entrées/Sorties

Les entrées et sorties sont gérées séparément :

IN :



OUT :



JAVA = panoplie d'outils pour communiquer dans les deux sens avec toutes sortes de sources

Basic I/O

Définition

Les flux (=*stream*) désignent les entrées/sorties des programmes (autre que les arguments du main)

Il s'agit d'outils essentiels dès que l'on souhaite

- Lire/écrire des fichiers
- Sauver les paramètres d'un programme (=écrire un fichier !)
- Communiquer avec d'autres postes de travail (en réseau)
- Travailler à plusieurs sur un projet (=svt gérer des fichiers)
- Lire/écrire dans des BD
- Saisie clavier
- Ecriture console

Plan général

1. **Fichiers** : lire les noms, vérifier l'existence, vérifier la possibilité d'écriture...
   - Equivalent des fonctions `dir, cd, ...` Mais à l'intérieur de JAVA
2. Une fois le fichier ciblé, **l'ouvrir** et **lire** ce qu'il y a dedans
3. **Créer** un fichier et/ou **écrire** dedans
4. ... D'autres choses se gèrent comme les fichiers
   - Clavier, Réseau...

Classe File

Cette classe permet de gérer les fichiers :

- test d'existance
- distinction fichier/répertoire
- copie/effacement
- ...

- boolean canExecute()
- boolean canRead()
- boolean canWrite()
- boolean delete()
- boolean isDirectory()
- boolean isFile()
- File[] listFiles()
- boolean mkdir()

Nombreuses opérations très intéressantes concernant la manipulation des fichiers

Lecture de fichiers

1. `File` : désigner un fichier
2. `FileInputStream` : création de cet objet = ouverture en lecture du fichier
   - Des **exceptions** à gérer
   - Penser à **fermer** les fichiers ouverts

```
1  FileInputStream in = null;
2  File f = new File("xanadu.txt");
3   try {
4       in = new FileInputStream(f); // ouverture du fichier
5       // Throws: FileNotFoundException : => try/catch
6
7       // OPERATIONS DE LECTURE
8
9       }
10  } finally {
11      if (in != null) {
12            in.close();
13      }
14  }
```

Les petits pièges... La fermeture des fichiers

1. Toujours fermer un fichier ouvert...

```
1  try { FileInputStream in = new FileInputStream(
2                       new File(filename));
3           ... // LECTURE
4           in.close();
5  }catch(...){...}
```

Les petits pièges... La fermeture des fichiers

1. Toujours fermer un fichier ouvert...

```
1 try { FileInputStream in = new FileInputStream(
2                     new File(filename));
3         ... // LECTURE
4         in.close();
5 }catch(...){...}
```

2. Même s'il y a des erreurs pendant la lecture !

```
1 try { FileInputStream in = new FileInputStream(
2                                 new File(filename));
3         ... // LECTURE
4         in.close();
5 }catch(...){ in.close(); }
```

Les petits pièges... La fermeture des fichiers

1. Toujours fermer un fichier ouvert...

```
1  try { FileInputStream in = new FileInputStream(
2                       new File(filename));
3          ... // LECTURE
4          in.close();
5  }catch(...){...}
```

2. Même s'il y a des erreurs pendant la lecture !

```
1  try { FileInputStream in = new FileInputStream(
2                                  new File(filename));
3          ... // LECTURE
4          in.close();
5  }catch(...){ in.close(); }
```

3. Mais ça ne compile pas !

```
1  FileInputStream in = null;
2  try { in = new FileInputStream(new File(filename));
3          ... // LECTURE
4          in.close();
5  }catch(...){   in.close(); }
```

Les petits pièges... La fermeture des fichiers

1. Toujours fermer un fichier ouvert...

```
1  try { FileInputStream in = new FileInputStream(
2                       new File(filename));
3           ... // LECTURE
4           in.close();
5  }catch(...){...}
```

2. Même s'il y a des erreurs pendant la lecture !

```
1  try { FileInputStream in = new FileInputStream(
2                                   new File(filename));
3           ... // LECTURE
4           in.close();
5  }catch(...){ in.close(); }
```

3. Mais ça ne compile pas !

```
1  FileInputStream in = null;
2  try { in = new FileInputStream(new File(filename));
3          ... // LECTURE
4          in.close();
5  }catch(...){   in.close(); }
```

4. Plus élégant : lignes 4-5 $\Rightarrow$ `finally{in.close()}`

Les petits pièges... La fermeture des fichiers

5. La solution précédente ne marche pas encore !

```
1 FileInputStream in = null;
2 try { in = new FileInputStream(new File(filename));
3         ... // LECTURE
4 }finally{  in.close(); }
```

Les petits pièges... La fermeture des fichiers

5. La solution précédente ne marche pas encore !

```
1 FileInputStream in = null;
2 try { in = new FileInputStream(new File(filename));
3         ... // LECTURE
4 }finally{  in.close(); }
```

6. ... le `close` est susceptible de lever une exception si le fichier n'est pas ouvert (NullPointerException) !

```
1  FileInputStream in = null;
2  File f = new File("xanadu.txt");
3   try {
4       in = new FileInputStream(f); // ouverture du fichier
5       // Throws: FileNotFoundException : => try/catch
6
7       // OPERATIONS DE LECTURE
8
9       }
10  } finally {              // On est sûr de passer par là
11      if (in != null) { // vérifier que le fichier est ouvert
12            in.close();
13      }
14  }
```

Lecture de fichiers (byte stream)

public int read() throws IOException

Reads a byte of data from this input stream. This method blocks if no input is yet available.

Returns :

the next byte of data, or -1 if the end of the file is reached.

Throws :

IOException - if an I/O error occurs.

```
1     while ((c = in.read()) != -1) {
2        System.out.print(c);
3      }
4      // signifie en fait:
5      c = in.read(); // lecture d'un octet
6           // susceptible de lever IOException => try/catch
7      while (c  != -1) { // tant que fin de fichier non atteinte
8        System.out.print(c); // affichage dans la console
9        c = in.read(); // lecture du caractère suivant
10     }
```

Limites → Nouvelles classes

- Des classes supplémentaires enrichissent les `FileStream` :
- Processus de décoration d'objets (cf LI314)
- Fonctions de lecture

```
1  DataInputStream istream = null;
2  try {
3      istream = new DataInputStream(
4        new FileInputStream(new File("toto.dat")));
5
6      System.out.println(istream.readChar());
7      System.out.println(istream.readDouble());
8      System.out.println(istream.readInt());
9      System.out.println(istream.readChar());
10     // pas de fonctions pour les String...
11 } finally{
12     if(istream != null)
13         istream.close();
14 }
```

- ... Ces fonctions ont évidemment des fonctions symétriques pour l'écriture

Ecriture de fichiers (byte stream)

public FileOutputStream(String name) throws FileNotFoundException

Creates an output file stream to write to the file with the specified name.
Parameters :
name - the system-dependent filename
Throws :
FileNotFoundException - if the file exists but is a directory rather than a regular file, does not exist but cannot be created, or cannot be opened for any other reason

- La fonction est proche de celle d'ouverture en lecture... Avec une option supplémentaire : ajouter des choses dans un fichier...
- `public FileOutputStream(String name, boolean append) throws FileNotFoundException`

Ecriture de fichiers

- Exemple d'utilisation (Oracle Java Tutorials) :

```
1  FileInputStream in = null;
2  FileOutputStream out = null;
3
4  try {
5      in = new FileInputStream("xanadu.txt");
6      out = new FileOutputStream("outagain.txt");
7      int c = in.read();
8
9      while (c != -1) {
10         out.write(c);
11         c = in.read();
12     }
13 } finally {
14     if (in != null) {
15         in.close();
16     }
17     if (out != null) {
18         out.close();
19     }
20 }
```

- Que fait ce programme ?

Limites → Nouvelles classes

- 1 Octet = 1 char (en fonction du codage)... On voudrait lire des choses de plus haut niveaux
  - Entier/Double = 4 octets
  - String...

- Des classes supplémentaires enrichissent les `FileStream` :
  - `DataInputStream/DataOutputStream`

```
1  // Ecriture dans un fichier des types de base
2  DataOutputStream ostream = null;
3  try {
4      ostream = new DataOutputStream(
5        new FileOutputStream(
6          new File("toto.dat")));
7      ostream.writeChar('r');
8      ostream.writeDouble(2.5);
9      ostream.writeInt(6);
10     ostream.writeChars("toto");
11 } finally{
12     if(ostream != null)
13         ostream.close();
14 }
```

Ouverture en écriture...

Par défaut : remplacement des fichiers existants...

Gare aux catastrophes !

Il existe une option pour ajouter des choses dans un fichier **sans** écraser le contenu :

```
1 try {
2     ostream = new DataOutputStream(
3       new FileOutputStream(
4         new File("toto.dat"), true));
5                   // le boolean correspond à l'option append
6 ...
```

ASCII or not ASCII...

Dans les opérations précédentes, voici le fichier manipuler (ouvert avec emacs) :

```
^@r@^@^@^@^@^@^@^@^@^F^@^@^@t^@o^@t^@o
```

Ce qui n'est pas très convivial...
En fait il y a des avantages et des inconvénients.

Gestion de l'ASCII

Distinction entre ASCII et données brutes

Il faut distinguer un fichier **ASCII** d'une ***donnée brute*** :

- On peut pas lire directement les int/double sauver par la méthode précédente...
- On peut tricher :

```
1 ostream.writeChars(((Double) 2.5).toString());
```

  $\Rightarrow$ lisible mais perte de précision

Problème de lecture des String

- Lecture caractère par caractère fastidieuse
- Problème des sauts de ligne (codage différent selon le système)

$\Rightarrow$ Choisir en fonction des situations (possibilité de mélanger)

ASCII or not ASCII...

ASCII

- Fichiers de Textes
- Entêtes des fichiers
- XML, HTML...
- Parfois pour quelques chiffres
  - lorsque la précision n'est pas importante
  - pour les fichiers excels...

not ASCII

- Fichiers de chiffres
  - Volume
  - Précision

Gestion en lecture de fichier ASCII

- Encore une nouvelle classe : `BufferedReader`...

`public String readLine() throws IOException`

Reads a line of text. A line is considered to be terminated by any one of a line feed or a carriage return.
Returns : A String containing the contents of the line, not including any line-termination characters, or null if the end of the stream has been reached
Throws : IOException - If an I/O error occurs

```
1  BufferedReader in = null;
2  try {
3      in = new BufferedReader( new InputStreamReader(
4          new FileInputStream( // décorations multiples
5              new File("xanadu.txt"))));
6
7      String buf = in.readLine();
8      while(buf != null){
9          System.out.println(buf);
10         buf = in.readLine();
11     }
12 }...
```

Stratégie de lecture des fichiers ASCII

Une fois la ligne lue, il est nécessaire de la traiter...

- Séparer les mots : `StringTokenizer`
  - Choix des séparateurs (espace, tabulation, virgule...), accès aux sous-chaines
- Conversion en Int, Double...
  - `public static Double valueOf(String s) throws NumberFormatException`
  - `public static Double parseDouble(String s) throws NumberFormatException`

Stratégie alternative

Idée

Combiner la lecture de fichier avec un Tokenizer... Ca donne un `Scanner`

```java
Scanner s = null;
try {
    s = new Scanner(
      new BufferedReader(
        new FileReader("xanadu.txt")));

    while (s.hasNext()) {
        System.out.println(s.next());
    }
} finally {
    if (s != null) {
        s.close();
    }
}
```

Par défaut, le séparateur est ",\s*" mais on peut le changer...

```java
s.useDelimiter(",\\s*");
```

Ecriture de fichiers ASCII

BufferedReader ⇒ BufferedWriter

```
1 BufferedWriter writer =
2       new BufferedWriter(
3             new FileWriter(
4                   new File("monFichierASCII.txt")));
5
6  writer.write("toto");
7  writer.close();
```

Fonctions utiles de formatage du texte

JAVA propose des outils de formatage du texte issu directement de la syntaxe C : c'est utile pour faire des affichages tabulés, ou pour améliorer la lisibilité

- Syntaxe usuelle (implicite) de conversion Double → String :

```
1 int i = 2;
2 System.out.println("i = "+i);
```

- Syntaxe explicite :

```
1 int i = 2;
2 System.out.println("i = "+((Integer) i).toString());
```

- Formatage (disponible dans la classe String, entre autre) :
  `static String format(String format, Object... args)`

Exemples de formatages

```
1 double[] tab = new double[10];
2 for(int i=0; i<10; i++)
3     tab[i] = Math.random()*1000;
4 for(int i=0; i<10; i++)
5     System.out.println(tab[i]);
```

```
1 30,781400
2 453,926246
3 213,175566
4 18,565873
5 940,045852
6 ...
```

```
1 for(int i=0; i<10; i++)
2     System.out.println(
3       String.format("%12f", tab[i]));
```

```
1   357,600960
2   832,277086
3    89,423706
4   124,335349
5  ...
```

```
1 for(int i=0; i<10; i++)
2     System.out.println(
3       String.format("%10.3f", tab[i]));
```

```
1    870,771
2    735,952
3     44,770
4  ...
```

```
1 for(int i=0; i<10; i++)
2     System.out.println(
3       String.format("%010.3f", tab[i]));
```

```
1 000825,579
2 000611,987
3 000013,986
4  ...
```

Ca marche aussi avec les entiers (%d) et les String (%s)

Vers les bases de données...

Fichier structuré

Dans certains fichiers, on souhaite stocker des données structurées, par exemple :

- des matrices de chiffres,
- des fichiers clients avec des colonnes structurées (eg : nom, prénom, ...)
- ...

En général, on souhaite faire des traitements associés...
Il faut aller assez vite

Classe RandomAccessFile

- Création/Ouverture assez classique

RandomAccessFile

```
public RandomAccessFile(File file,
                        String mode)
                 throws FileNotFoundException
```

Creates a random access file stream to read from, and optionally to write to, the file specified by the `File` argument. A new `FileDescriptor` object is created to represent this file connection.

The `mode` argument specifies the access mode in which the file is to be opened. The permitted values and their meanings are:

Value	Meaning
`"r"`	Open for reading only. Invoking any of the `write` methods of the resulting object will cause an `IOException` to be thrown.
`"rw"`	Open for reading and writing. If the file does not already exist then an attempt will be made to create it.
`"rws"`	Open for reading and writing, as with `"rw"`, and also require that every update to the file's content or metadata be written synchronously to the underlying storage device.
`"rwd"`	Open for reading and writing, as with `"rw"`, and also require that every update to the file's content be written synchronously to the underlying storage device.

  - Option `"r"`, `"w"`, `"rw"`
- Fonction de déplacement du curseur dans le fichier
- Un certain nombre d'outils classique
  - read/write sur plusieurs type de données (String, double, int, long...)
  - `setLength` pour régler la longueur du fichier

Classe RandomAccessFile

- Création/Ouverture assez classique
- Fonction de déplacement du curseur dans le fichier

seek

```
public void seek(long pos)
          throws IOException
```

Sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs. The offset may be set beyond the end of the file. Setting the offset beyond the end of the f
the file length. The file length will change only by writing after the offset has been set beyond the end of the file.

Parameters:

pos - the offset position, measured in bytes from the beginning of the file, at which to set the file pointer.

Throws:

IOException - if pos is less than 0 or if an I/O error occurs.

  - en octets, toujours depuis le début du fichier
- Un certain nombre d'outils classique
  - read/write sur plusieurs type de données (String, double, int, long...)
  - `setLength` pour régler la longueur du fichier

Classe RandomAccessFile

- Création/Ouverture assez classique
- Fonction de déplacement du curseur dans le fichier
- Un certain nombre d'outils classique
  - read/write sur plusieurs type de données (String, double, int, long...)
  - `setLength` pour régler la longueur du fichier

Note : dans ce type de fichier, il faut

1. connaitre la longueur des types usuels (int=4 octets, long = 8 octets, double = 4 octets, char = 1 (ou 2) octet, ...)
2. faire la chasse aux structures de taille variable (comme les String)...
   `String.format("%10s", "toto")`

RandomAccessFile : EXEMPLE D'ÉCRITURE

```
1  import java.io.RandomAccessFile;
2  [...]
3  String fname = "monfichiertest.dat";
4  File f = new File(fname);
5  RandomAccessFile raf = new RandomAccessFile(f, "rw");
6
7  raf.writeChars(String.format("%4s", "titi"));
8  raf.writeChars(String.format("%10s", "toto"));
9  raf.writeDouble(Math.PI);
10 raf.writeChars(String.format("%10s", "tata"));
11 raf.close();
```

Fichier produit :

```
1  titi      toto            !ûTD      tata
```

RandomAccessFile : EXEMPLE DE LECTURE

Fichier produit :

```
titi      toto          !ûTD      tata
```

```
        RandomAccessFile raf2 = new RandomAccessFile(f, "rw");

        for(int i=0; i<4; i++)
            System.out.print(raf2.readChar());
        System.out.println(); // titi

        raf2.seek(28);
        System.out.println(raf2.readDouble()); // 3.14...

        raf2.seek(8);
        for(int i=0; i<10; i++)
            System.out.print(raf2.readChar()); //      toto
        System.out.println();

        raf2.seek(0);
        System.out.println(raf2.readLine());
        //titi      toto          !ûTD      tata

        raf2.close();
```

Interface Serializable

Si un objet `implements Serializable`, il devient sauvegardable/chargeable automatiquement

Exemple avec la classe Vecteur :

```
1  public class Vecteur implements Serializable{
2     private static final long serialVersionUID =
3              -2056278751781864 36L;
4     private double x,y;
5
6     public Vecteur(double x, double y) {
7        super();
8        this.x = x;
9        this.y = y;
10    }
11
12    (...)
```

Serializable : sauvegarde/chargement

```java
// sauvegarde
try{
   FileOutputStream fos =
      new FileOutputStream(filename);
   ObjectOutputStream oos =
      new ObjectOutputStream(fos);
   oos.writeObject(new Vecteur(2,2));
   fos.close();
}catch (Exception e){
         System.out.println(e.toString());
}
```

```java
// chargement
try{
   FileInputStream fin =
      new FileInputStream(filename);
   ObjectInputStream oos =
      new ObjectInputStream(fin);
   Vecteur v = (Vecteur) oos.readObject();
   fin.close();
}catch (Exception e){
         System.out.println(e.toString());
}
```

- on charge des Object → il faut un cast pour avoir ce que vous voulez
- problèmes avec les attributs static
- Il faut que les attributs soient sérializable (!)
  - Eliminer un attribut de la sauvegarde : mot clé `transient`
- Problème de version très sensible : `serialVersionUID` dans les objets

Limites d'usage

Lors de la sauvegarde, la Serialization enregistre le type exact de l'objet. Le cas suivant pose problème :

1. Sauvegarde d'un objet
2. Modification (même très légère) de la classe
3. Chargement de l'objet précédent : `ClassNotFoundException`

Ajouter `serialVersionUID` dans les objets

Bilan

- Très facile à utiliser
- On ne maitrise pas le format de sauvegarde

Interface Externalizable

- Utilisable comme `Serializable`
  - `Externalizable extends Serializable`
  - Un objet `Externalizable` EST UN `Serializable`
- Maitrise du format de sauvegarde à travers deux fonctions de lecture/écriture dans les flux...
- Dans la classe `Vecteur`

```
1  public class Vecteur implements Externalizable{
2   (...)
3
4     public void readExternal(ObjectInput in) throws IOException,
5        ClassNotFoundException {
6        x = in.readDouble();
7        y = in.readDouble();
8     }
9
10    public void writeExternal(ObjectOutput out) throws IOException
11       out.writeDouble(x);
12       out.writeDouble(y);
13    }
```

Les flux dépassent les fichiers

- Lire une String comme un flux avec la classe `StringReader`

```
1 String s = "blablabla...";
2 // initialisation à partir d'une String
3 StringReader sread = new StringReader(s);
4 System.out.println(sread.read()); // -> 98 (code de 'b')
```

- Fonctionnalité très utile pour unifier une chaine de traitements
  - tous les types d'entrées peuvent être des flux, on peut jouer avec l'héritage

La console est un flux

- Entrée console = un flux particulier (`System.in`)

```
1  // créer un Scanner, choisir un délimiteur
2  try {
3      s = new Scanner(System.in);
4      s.useDelimiter("␣");
5      while (s.hasNext()) {
6          System.out.println(s.next());
7      }
8  } finally {
9      if (s != null) {
10         s.close();
11     }
12 }
```

NB : les entrées consoles ne sont validées qu'après un retour chariot

1. Tapons : `le lapin + RETOUR`
2. Après l'espace, rien ne se passe
3. Après le retour, on voit :

```
1 le // la séparation a été prise en compte
2 lapin
```

Outil avancé pour la console

- Objet `Console`

```
1 Console c = System.console();
2 if (c == null) {
3     System.err.println("No console.");
4     System.exit(1);
5 }
6
7 String login = c.readLine("Enter your login: ");
8 char [] oldPassword = c.readPassword("Enter your old password: ");
```

- Notons la forme particulière de la construction de la console
- Console est en réalité un Singleton (instance unique de l'objet)... Plus de détail en LI314

Pipe FIFO

- Approche tuyau (FIFO), ce qui rentre en premier sort en premier

```
1  PipedReader read = new PipedReader();
2  PipedWriter write = new PipedWriter(read);
3
4  write.append("toto");
5  write.append("est dans le jardin");
6
7  while(read.ready())
8      System.out.println((char) read.read());
9  // conversion obligatoire, sinon code ASCII
10 write.close();
11 read.close();
```

- Problème de la lecture caractère à caractère...
- Une idée de solution ?

Pipe FIFO

- Approche tuyau (FIFO), ce qui rentre en premier sort en premier

```
1  PipedReader read = new PipedReader();
2  PipedWriter write = new PipedWriter(read);
3
4  write.append("toto");
5  write.append("est dans le jardin");
6
7  while(read.ready())
8      System.out.println((char) read.read());
9  // conversion obligatoire, sinon code ASCII
10 write.close();
11 read.close();
```

- Problème de la lecture caractère à caractère...
- Une idée de solution ?
- Introduire un Scanner sur la source !

```
1 Scanner s = new Scanner(read);
2 while(s.hasNext())
3     System.out.println(s.next());
```

Lecture http sur le web

- Lire sur internet est une simple formalité...
  - Outil pour les URL → flux
  - Ouverture/lecture classique !

```
1 URL url = new URL("http://www.yahoo.fr"); // outil ad'hoc
2 BufferedReader bf=new BufferedReader(
3    new InputStreamReader(url.openStream(), "utf8"));
4 String buf = bf.readLine();
5 while(buf != null){
6     buf = bf.readLine();
7     System.out.println(buf);
8 }
9 bf.close();
```

Collecte de données sur le web

On peut mettre les données lues aux formats :

- String : utilisation des méthodes pour sélectionner les parties pertinentes
- XML : bibliothèque DOM
- brutes dans un fichier pour des traitements ultérieurs
- ...

Dialogue entre machine 1/2

- Etablir un lien entre machine = `Socket`
  - Serveur = `ServerSocket`
  - Client = `Socket` simple pour se connecter

Partie serveur

```
1  ServerSocket socketserver  ;
2  Socket socketduserveur ;
3  try {
4      socketserver = new ServerSocket(2009); // creation serveur
5      socketduserveur = socketserver.accept(); // attente + ecoute
6                  // + acceptation
7      System.out.println("qqn s'est connecté ! Poste message : ");
8
9      // retour aux méthodes déjà vues !
10     DataOutputStream out =
11       new DataOutputStream(socketduserveur.getOutputStream());
12     out.writeBytes("message du serveur");
13
14     socketserver.close();
15     socketduserveur.close();
16 }catch (IOException e) {   e.printStackTrace(); }
```

Coté client

```
1  Socket socket;
2  try {
3      socket = new Socket(InetAddress.getLocalHost(),2009);
4      BufferedReader bf=new BufferedReader(
5        new InputStreamReader(socket.getInputStream()));
6      String buf = bf.readLine();
7
8      System.out.println("Je suis le client");
9      System.out.println("Je me suis connecté");
10     System.out.println("J'ai lu:");
11     System.out.println(buf);
12
13     socket.close();
14 }catch (UnknownHostException e) {
15     e.printStackTrace();
16 }catch (IOException e) {
17     e.printStackTrace();
18 }
```

Vers les BD, SQL

- Connection aux bases de données aisée

```
1  // le driver gère la connection (à MySQL par exemple)
2  Connection con = DriverManager.getConnection(
3                              "jdbc:myDriver:myDatabase",
4                              username,
5                              password);
6
7  // instanciation de la connection
8  Statement stmt = con.createStatement();
9  // envoi d'une requête
10 ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
11 // traitement séquentiel des réponses
12     while (rs.next()) {
13         int x = rs.getInt("a");
14         String s = rs.getString("b");
15         float f = rs.getFloat("c");
16     }
```