ENSAH

جامعة عبد المالك السعدي
Université Abdelmalek Essaadi

Gestion de fichiers, flux d'entrées/sortie, sérialisation

Langage Java

2ème année Génie Informatique

Ecole Nationale des Sciences Appliquées – Al Hoceima

Prof A. Bahri
abahri@uae.ac.ma

A.U 2020/2021

Gestion de fichiers

- La gestion de fichiers se fait par l'intermédiaire de la classe *java.io.File*.

- Cette classe possède des méthodes qui permettent d'interroger ou d'agir sur le système de fichiers du système d'exploitation.

  - Un objet de la classe *java.io.File* peut représenter un fichier ou un répertoire.

Quelques méthodes de la classe java.io.File

- *File (String name):* prend en paramètre une chaîne de caractères qui indique un chemin, relatif ou absolu, vers un fichier ou un répertoire.
- *File (String path, String name):* prend deux chaînes de caractères en paramètre. La première indique le chemin vers le fichier ou le répertoire. La seconde le nom de ce ficher ou répertoire.
- *File (File dir, String name):ce constructeur est analogue au précédent, sauf que le chemin est exprimé sous la forme d'une instance de File.*
- *boolean isFile (), boolean isDirectory ():* *permettent de tester si cette instance de File représente un fichier ou un répertoire.*
- *boolean mkdir () et boolean mkdirs() :* *ces deux méthodes créent le répertoire représenté par cette instance de File.* ***mkdirs()*** *peut créer une série de répertoires imbriqués, ce qui n'est pas le cas de mkdir(). Retourne false si la création n'a pas pu se faire.*
- *boolean exists () canRead(), canWrite(), canExecute() :* *permettent de tester différents éléments sur le fichier ou le répertoire représenté par cette instance de File.*
- *boolean delete ():demande l'effacement de ce fichier ou répertoire. Retourne false si cet effacement n'a pu avoir lieu.*

Gestion de fichiers

- *getName() : retourne le nom de ce fichier ou répertoire, sans son chemin d'accès s'il est précisé. Ce nom correspond au dernier élément de la séquence de nom de cette instance de File.*
- *getPath() : retourne le chemin complet de ce fichier. Le retour de cette méthode est le même que celui de la méthode toString().*
- *getAbsolutePath() : le retour de cette méthode diffère si l'instance de File représente un chemin absolu ou pas. Si ce chemin est absolu, le retour de cette méthode est le même que getPath(). S'il ne l'est pas, un chemin absolu est calculé, relativement au répertoire courant de l'application.*
- *etc,.*

Notion de chemin

- Il existe 3 sortes de chemins (uniques) :
  - Relatif ../tp/fichier.txt
  - Chemin absolue /tp/fichier.txt
  - Chemin canonique c:\Program Files\Java\
- Une représentation en String (path) ainsi qu'une représentation en File
  - Chemin absolu en String : getAbsolutePath()
  - Chemin absolu en File : getAbsoluteFile()

Gestion des chemins

- Constante java.io.separatorChar
  / (Unix), \ (Windows) et : ou / (Mac) (ajouté automatiquement)
- Manipulation des chemins :
  - nom du fichier : getName()
  - chemin local vers le fichier : getPath()
  - chemin absolu vers le fichier : getAbsolutePath()/getAbsoluteFile()/isAbsolute()
  - chemin canonique : getCanonicalPath()/getCanonicalFile()
  - chemin vers le repertoire père getParent()/getParentFile()
- Il n'y a rien pour gérer les extensions !

Les répertoires

- Savoir si un fichier est un répertoire :
  - **isDirectory**()
- Lister les fichiers présents :
  - String[] **list**() ou File[] **listFiles**()
- Il es possible de filtrer les fichiers :
  - filtre sur le nom : **listFile**(FilenameFilter)
  - filtre sur le fichier : **listFiles**(FileFilter filter)

Les flux

- Un flux (stream) est un chemin de communication entre la source d'une information et sa destination

- Un processus consommateur n'a pas besoin de connaître la source de son information; un processus producteur n'a pas besoin de connaître la destination

- Difficulté d'un langage d'avoir un bon système d'entrées/sorties.

- Beaucoup de sources d'E/S de natures différentes (console, fichier, socket,…).

- Beaucoup d'accès différents (accès séquentiel, accès aléatoire, mise en mémoire tampon, binaire, caractère, par ligne, par mot, etc.).

Les flux proposés par java

- Flux d'entrée/sortie de bytes.
- Flux d'entrée/sortie de caractères depuis la version 1.1 de java.
- Toutes les classes d'entrée/sortie sont dans le package *java.io*
- Toutes les méthodes peuvent générer une *java.io.IOException*

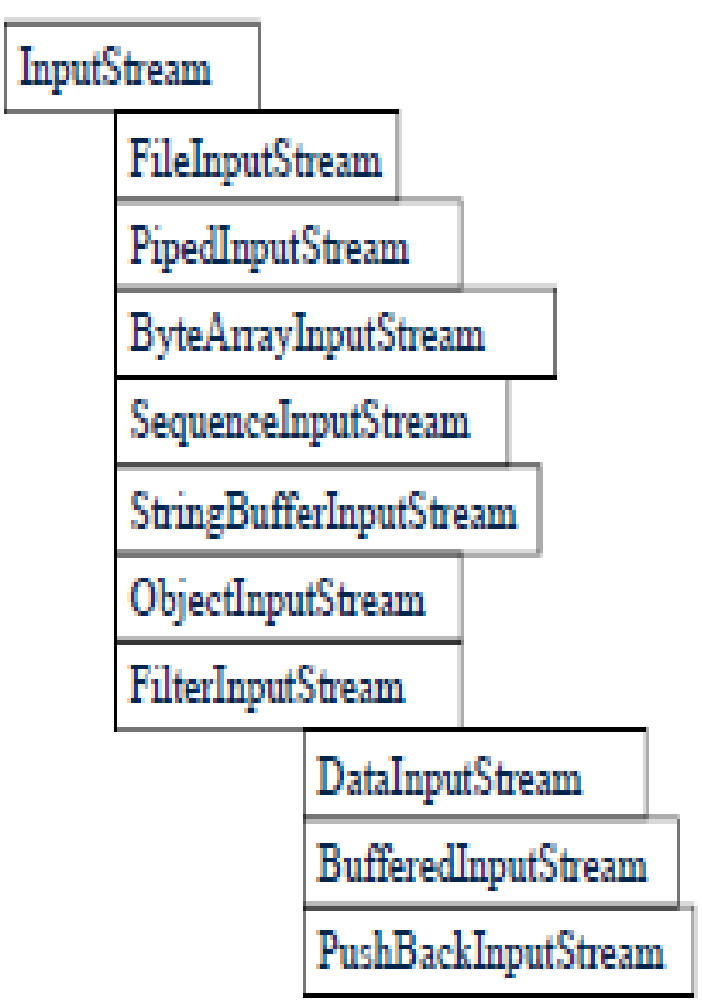
Classes de base abstraites des flux

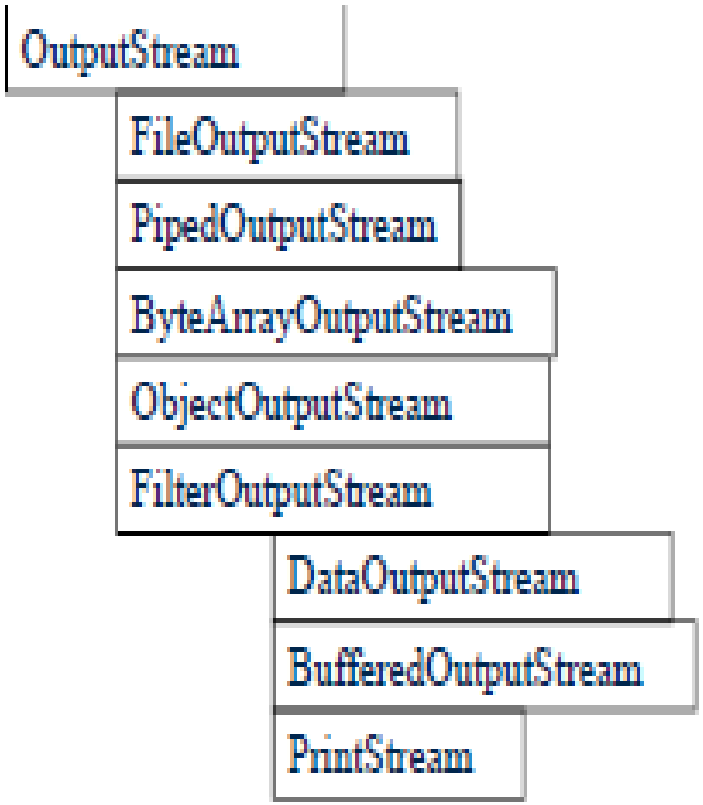
Il existe des flux de bas niveau et des flux de plus haut niveau (travaillant sur des données plus évoluées que les simples octets). Citons :

- Les flux de caractères
  - classes abstraites **Reader** et **Writer** et leurs sous-classes concrètes respectives.
- Les flux d'octets
  - classes abstraites **InputStream** et **OutputStream** et leurs sous-classes concrètes respectives,

	Flux d'octets	Flux de caractères
Flux d'entrée	*java.io.InputStream*	*java.io.Reader*
Flux de sortie	*java.io.OutputStream*	*java.io.Writer*

Classes de flux de bytes





La classe java.io.InputStream

- **Les méthodes de lecture :**

```
public int read () ;
public int read (byte b [ ]) ;
public int read (byte b [ ], int off, int len) ;
```

- **Exemple :**

```
InputStream s = ….. ;
byte buffer [ ] = new byte [1024] ;
try {
        s.read (buffer) ;
} catch (IOException e){ }
```

La classe java.io.InputStream

- Sauter des octets : *public long skip (long n) ;*
- Combien d'octets dans le flux : *public int available ()* ;
- Le flux supporte-t'il le marquage ? *public boolean markSupported () ;*
- Marquage d'un flux : *public void mark (int readlimit) ;*
- Revenir sur la marque: *public void reset () ;*
- Fermer un flux : *public void close () ;*

Exemple de flux d'entrée

```
import java.io.* ;
public class LitFichier
{
        public static void main (String args []){
            try {
                InputStream s = new FileInputStream ("c:\\temp\\data.txt") ;
                byte buffer [ ] = new byte [s.available()] ;
                s.read (buffer) ;
                for (int i = 0 ; i != buffer.length ; i++)
                        System.out.print ( (char) buffer [i]) ;
                s.close () ;
            } catch (IOException e){
                System.err.println ("Erreur lecture") ;
            }
        }
}
```

La classe java.io.OutputStream

- **Les méthodes d'écriture :**

  public void write (int b) ;
  public void write (byte b [ ]) ;
  public void write (byte b [ ], int off, int len) ;

- **Nettoyage d'un flux, forçant l'écriture des données bufférisées :**

  public void flush ()

- **Fermeture d'un flux**

  public void close () ;

Exemple de flux de sortie

```
import java.io.* ;
public class EcritFichier
{
        static public void main (String args []){
            String Chaine = "Bonjour" ;
            try {
                    FileOutputStream f = new FileOutputStream ("c:\\temp\\data.txt") ;
                    f.write (Chaine.getBytes ()) ;
                    f.close () ;
            } catch (IOException e){
                    System.err.println ("Erreur ecriture") ;
            }
        }
}
```

Les classes FilterInputStream/FilterOutputStream

Ces deux classes servent de classes de base à des classes de gestion d'entrées/sorties plus évoluées:

- *BufferedInputStream* et *BufferedOutputStream* permettent de lire et écrire des données à travers un tampon de lecture/écriture pour améliorer les performances.

- *DataInputStream* et *DataOutputStream* permettent de lire/écrire des données formatées (byte, int, char, float, double, etc.)
- etc.

Exemples de lecture/écriture évoluée

Lecture:

```
InputStream s = new FileInputStream ("fichier") ;
DataInputStream data = new DataInputStream (s) ;
double valeur = data.readDouble () ;
```

Ecriture;

```
PrintStream s = new PrintStream (new FileOutputStream ("resultat")) ;
s.println ("On ecrit dans le fichier resultat") ;
```

Accès non-séquentielle ou aléatoire

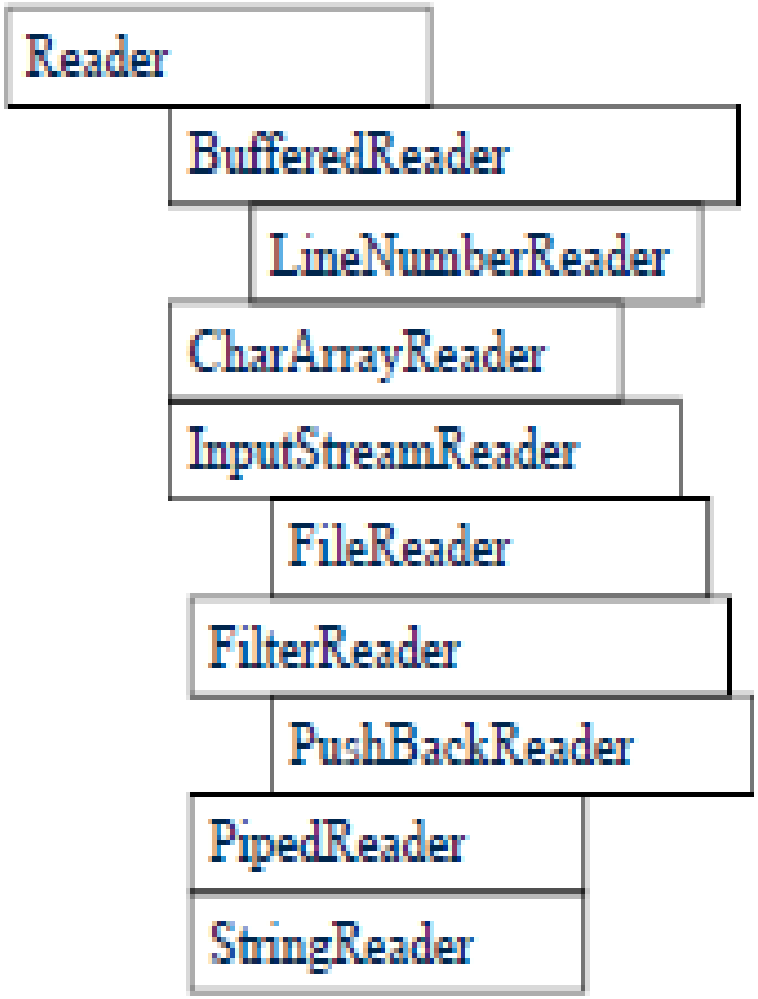
- La classe RandomAccessFile
  - new RandomAccessFile("fichier.txt", "r");

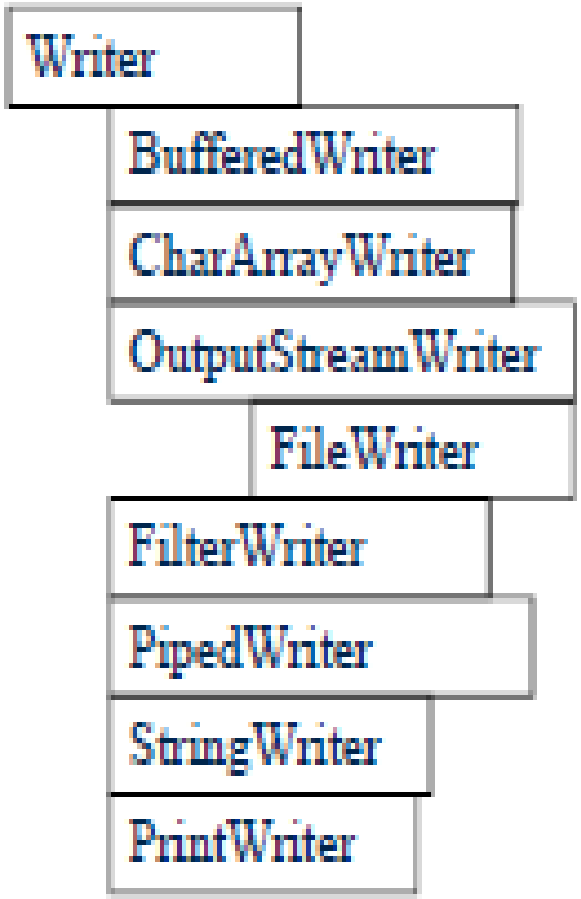
- Après l'ouverture d'un fichier, l'accès aux éléments du fichier est fait à l'aide d'un pointeur qui est traité avec les méthodes suivantes :
  - *skipBytes() :* Positionnement après un nombre spécifié d'octets.
  - *seek()* : Positionnement juste avant l'octet spécifié.
  - *getFilePointer()* : Pour obtenir la position d'un pointeur.

Accès non-séquentielle ou aléatoire

```
//Ecriture dans un fichier des lettres de ‘a’ à ‘z’,
//Puis remplacement d'une des lettres de façon aléatoire par "?".
try{
// écriture
   RandomAccessFile out = new RandomAccessFile ("accesDirect","rw");
   for(char i = 'z'; i>='a'; --i){
      out.seek(i-'a');
      out.write(i);
   }
   out.seek((long)(Math.random()*out.length()));
   out.write('?');
   out.close();
}catch(IOException ex){ ... }
```

Les classes de flux de caractères





Les classes de flux de caractères

Reader

Flux de **char** en entrée (méthode bloquante)

- Lit un char et renvoie celui-ci ou -1 si c'est la fin du flux
  - abstract int **read**()
- Lit un tableau de char (plus efficace)
  - int **read**(char[] b)
  - int **read**(char[] b, int off, int len)
- Saute un nombre de caractères
  - long **skip**(long n)
- Ferme le flux
  - void **close**()

Writer

Flux de caractère en sortie

- Ecrit un caractère, un int pour qu'il marche avec le read
  - abstract void **write**(int c)
- Ecrit un tableau de caractère (plus efficace)
  - void **write**(char[] b)
  - void **write**(char[] b, int off, int len)
- Demande d'écrire ce qu'il y a dans le buffer
  - void **flush**()
- Ferme le flux
  - void **close**()

Les classes de flux de caractères

Copie de flux

Caractère par caractère (mal)

```
public static void copy(Reader in,Writer out) throws IOException {
        int c;
        while((c=in.read())!=-1)
                out.write(c);
}
```

Par buffer de caractère

```
public static void copy(Reader in,Writer out) throws IOException {
        char[] buffer=new char[1000];
        int size;
        while((size=in.read(buffer))!=-1)
                out.write(buffer,0,size);
}
```

Les classes de flux de caractères

Exemple de BufferedReader

```
import java.io.*;
public class TestBufferedReader
{
        public static void main(String args[]){
            try {
                String ligne ;
                BufferedReader fichier = new BufferedReader (new FileReader("data.txt"));
                while ((ligne = fichier.readLine()) != null) System.out.println(ligne);
                fichier.close();
            } catch (IOException e){
                System.err.println ("Erreur lecture") ;
            }
        }
}
```

Les classes de flux de caractères

Writer et chaine de caractère

- Un **Writer** possède des méthodes spéciales pour l'écriture de chaîne de caractères
  - Ecrire une String,
    - void **write**(String s)
    - void **write**(String str, int off, int len)

- Un **writer** est un **Appendable** donc
  - Ecrire un CharSequence
    - Writer **append**(CharSequence csq)
    - Writer **append**(CharSequence csq, int start, int end)

Les classes de flux de caractères

java.lang.Appendable

- Interface **générique** de tout les objets qui savent écrire des chaines de caractères
  - Ajoute un caractère ou une chaine de caractère
    - **Appendable append(char c)**
    - **Appendable append(CharSequence csq)**
    - **Appendable append(CharSequence csq, int start, int end)**

- Est utilisé par le **Formatter** comme flux de caractères

La sérialisation

- La sérialisation est un mécanisme permettant de rendre un objet persistant. Il peut être ensuite:
  - Stocké dans un fichier
  - Transmis sur le réseau (exemple: RMI)
  - …

- Le processus inverse est la désérialisation.
- Un objet sérialisé est dit persistant.
- Cette fonctionnalité est apparue dans la version 1.1 de Java

Pourquoi sérialiser ?

- Rendre un objet persistant nécessite une convention de format pour la lecture/écriture (cela peut être une opération complexe et difficile à maintenir) ou transmettre un objet via le réseau à une application distante.
- La sérialisation permet de rendre un objet persistant de manière simple et naturelle.
- Si un objet contient d'autres objets sérialisables, ceux-ci seront automatiquement sérialisés.
- La plupart des classes de base (mais pas toutes) du langage Java sont sérialisables.
- Si la classe a été modifiée entre la sérialisation et la désérialisation, l'exception *java.io.InvalidClassException* est déclenchée.

Comment sérialiser?

- Une classe est sérialisable si elle implémente l'interface *java.io.Serializable*.
- Des objets de type *java.io.ReadObjectStream* et *java.io.WriteObjectStream* vont permettre de sérialiser/désérialiser.
- Les données membres que l'on ne souhaite pas sauvegarder doivent être déclarées *transient*.
- Des méthodes de lecture/écriture peuvent être redéfinies le cas échéant:

private void writeObject (java.io.ObjectOutputStream out) throws IOException ;

private void readObject (java.io.ObjectInputStream in) throws IOException,ClassNotFoundException ;

Exemple Sérialisation/Désérialisation

```
public class A {
    int x,y;
    public A() {
        this(3,3);
    }
}
```

```
public class B extends A implements Serializable {
    transient int a = 8;
    int z;
}
```

```
B b=new B();
b.x=10;b.y=11;b.a=12;b.z=13;
out.writeObject(b);
out.close(); // auto-flush
```

```
B b=(B)in.readObject();
System.out.printf("x="+x+" y="+y+ "
a="+a+ " z="+z);
in.close();
```

x=3 y=3 a=0 z=13

Exemple Sérialisation/Désérialisation

```
import java.io.* ;
class Info implements Serializable{
                private String Nom = "" ;
                private String MotPasse = "" ;
                public Info(String n, String m){
                    Nom=n ; MotPasse = m ;
                }
                public String getNom () {
                     return Nom ;
                }
           public String getPassword () {
                     return MotPasse ;
                }
}
```

```
public class ExempleSerialisation{
                static public void main (String args []){
                      new ExempleSerialisation () ;
                }
                public ExempleSerialisation (){
                 Info User = new Info ("Pierre","password") ;
                    Ecrire (User) ;
                    User = Lire () ;
                    if (User != null)
                         System.out.println ("nom = "
                        +User.getNom () + " mot de passe = " +
                         User.getPassword ()) ;
                }
```

```
Info Lire (){
      Info User = null ;
      try {
        FileInputStream file = new FileInputStream
                                    ("c:\\travail\\info.txt") ;
     ObjectInputStream in = new  ObjectInputStream
                                                  (file) ;
        User = (Info) in.readObject() ;
     } catch (Exception ex){
       System.err.println ("Erreur de lecture " + ex) ;  }
    return User
}
} // Fin classe ExempleSerialisation
```

```
          void Ecrire (Info user){
              try { FileOutputStream file = new
                 FileOutputStream ("c:\\travail\\info.txt") ;
                 ObjectOutputStream out = new
                  ObjectOutputStream (file) ;
                   out.writeObject (user) ;
                    out.flush () ;
                    out.close () ;
                    file.close () ;
              } catch (IOException ex){
                  System.err.println ("Erreur d'ecriture " +
                                                  ex) ;}
           }
```