# *Gestion de fichiers, flux d'entrée/sortie*

# 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)*
*File (String path, String name)*
*File (File dir, String name)*
*boolean isFile ()*
*boolean isDirectory ()*
*boolean mkdir ()*
*boolean exists ()* Teste si le fichier ou le répertoire désigné par ce nom de chemin abstrait existe
*boolean delete ()*
*boolean canWrite ()*
*boolean canRead ()*
*File getParentFile ()*
*long lastModified ()*
*String [] list ()* Renvoie un tableau de chaînes nommant les fichiers et répertoires dans le répertoire désigné par ce nom de chemin abstrait.

# Exemple d’utilisation de la classe java.io.File

```
import java.io.* ;
public class ExempleFile
{
     static public void main (String args []) {  new ExempleFile () ; }
     ExempleFile () {    liste (new File ("c:\\")) ; }
     private void liste (File dir)
     {
        if (dir.isDirectory () == true)
        {
           String fichiers [ ] = dir.list () ;
           for (int i = 0 ; i != fichiers.length ; i++) System.out.println (fichiers [i]) ;
        }
       else
       {
          System.err.println (dir + " n'est pas un repertoire") ;
       }
    }
}
```

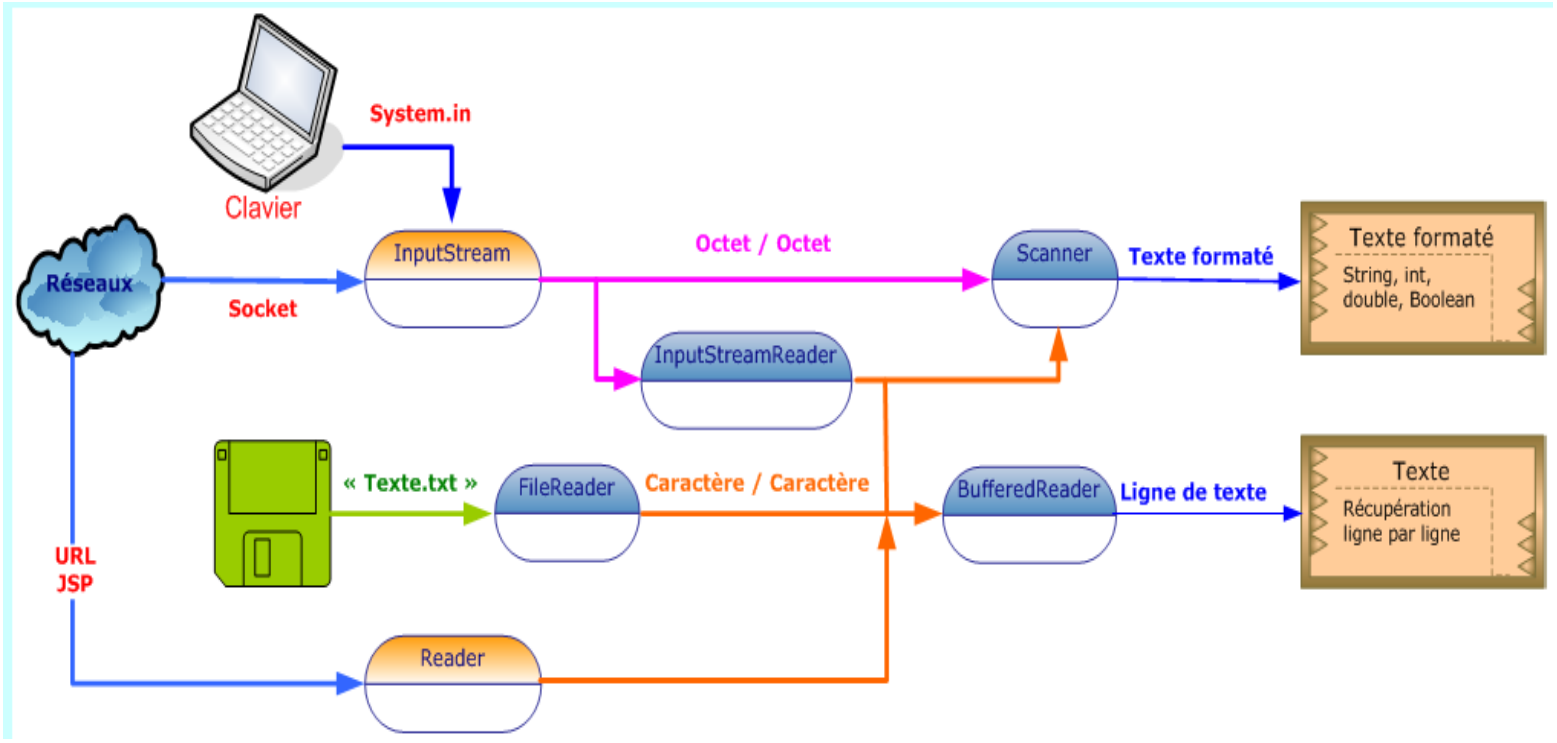
# Les flux

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

# 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***

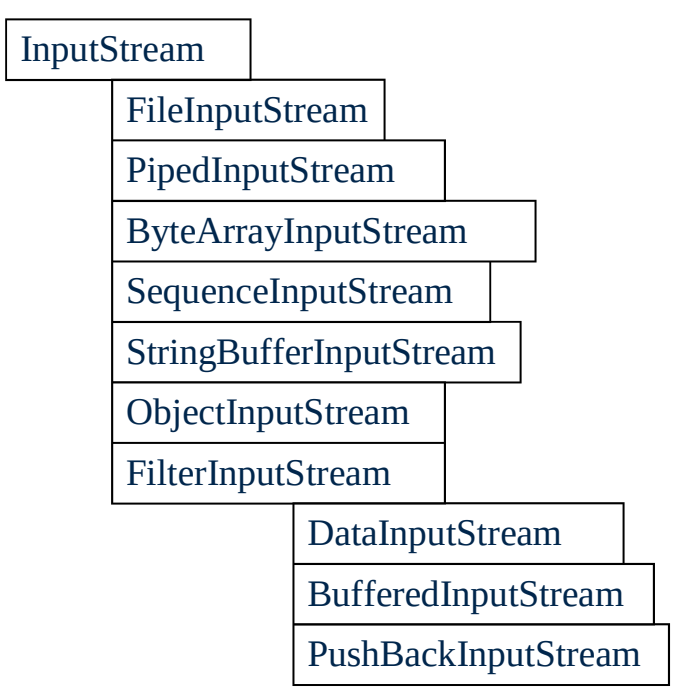
# Les flux proposés par java

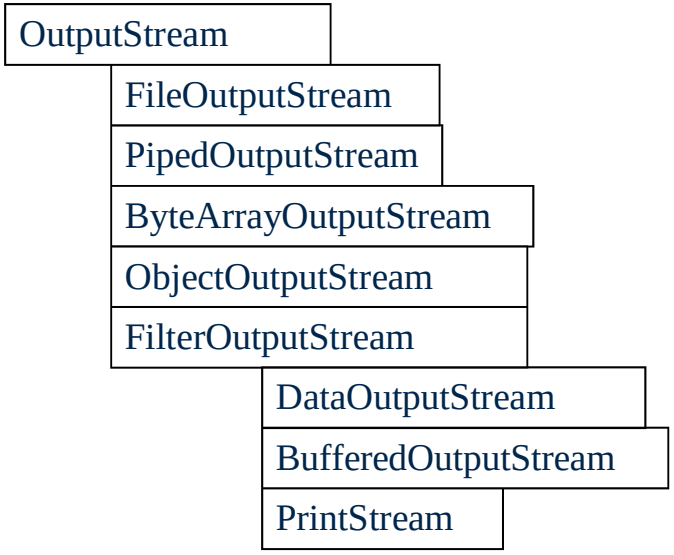


# *Classes de base abstraites des flux*

| | **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) ;***

- **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 ("FichierTest.txt") ;
          byte buffer [ ] = new byte [s.available()] ;
          //s.availble(): Renvoie une estimation du nombre d'octets pouvant être lus à partir de ce flux
                          d'entrée
          s.read (buffer) ; Lit un certain nombre d'octets du flux d'entrée s et les stocke dans le tableau buffer
            for (int i = 0 ; i != buffer.length ; i++)
                    System.out.print ( (char) buffer [i]+ "\t") ;
          s.close () ;
        } catch (IOException e)
          {
            System.err.println ("Erreur lecture") ;
          }
  }
}Bonjour
```

# 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 () ;***

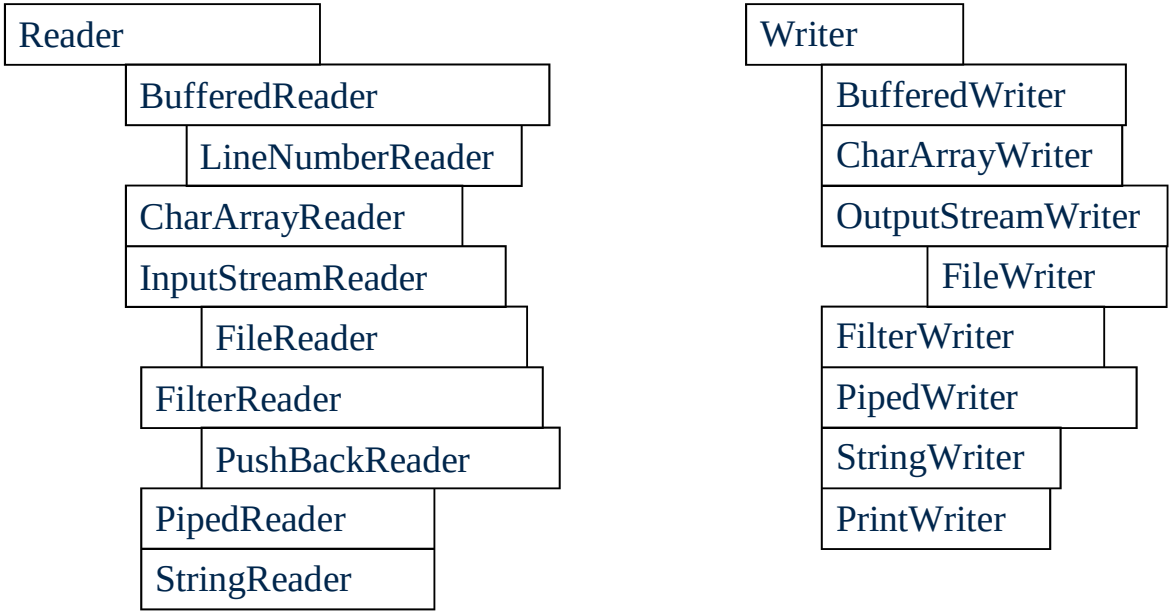
- **Fermeture d'un flux**

  ***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 ("FichierTest.txt") ;
              f.write (Chaine.getBytes ()) ;
              f.close () ;
           } catch (IOException e)
         {
              System.err.println ("Erreur ecriture") ;
         }
  }
}
```

# Les classes de flux de caractères



# Exemple de BufferedWriter

```
import java.io.*;
 class Ecrire
{public static void main(String[] args)
{try{FileWriter fw=new FileWriter("essai.txt");
BufferedWriter bw= new BufferedWriter(fw);
bw.write("Ceci est mon fichier \n");
bw.newLine();
bw.write("Il est à moi...");
bw.close();
}
catch (Exception e)
{ System.out.println("Erreur "+e);}
}}
```

# Exemple de BufferedReader

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

# Java 7: try-with-ressources

- Les ressources( fichiers, flux, connexion, …) doivent être fermées explicitement par le développeur.
- La clause *finally* peut être utilisée à cet effet mais:
  - La ressource doit être déclarée en dehors du bloc *try* pour être utilisable dans la clause *finally*
  - Une méthode *close ()* peut elle-même générer une exception entrainant une complexité dans le code

- Java 7 propose le mécanisme ARM (**A**utomatic **R**esource **M**anagement) applicable sur les objets qui implémentent l'interface *AutoCloseable* (une ressource non utilisée sera automatiquement fermée).

# La classe StringTokenizer

- **La classe StringTokenizer**

La classe StringTokenizer permet à une application de diviser une chaîne en jetons.

La méthode de tokenisation est beaucoup plus simple que celle utilisée par la classe StreamTokenizer.

Les méthodes StringTokenizer ne font pas de distinction entre les identificateurs, les nombres et les chaînes entre guillemets et ne reconnaissent pas et ignorent les commentaires.