La gestion des fichiers

- La Classe File:
  - Cette classe offre des fonctionnalités de gestion de fichiers comparables à celles auxquelles on accède par le biais de commandes système de l'environnement.
  - Parmis ses fonctionnalités:
    - Création/suppression, renommage de fichier ou de répertoire
    - Tester l'existante d'un fichier ou d'un répertoire
    - Lister les noms de fichiers d'un répertoire
    - ...
- Création d'un objet de type File:

```
File monFichier = new File('nomDuFichier.txt');
// Attention monFichier est un objet créer en mémoire a ne pas confondre
avec la création du fichier correspondant.

// Pour créer notre fichier on peut faire comme suit:
boolean ok = monFichier.createNewFile();

//ok = true si la création a eu lieu.
// si le fichier existe déjà, la création n'aura pas lieu
```

La gestion des fichiers

- On peut créer un fichier/répertoire en précisant son chemin comme argument au constructeur de la classe File

```
File monRep1 = new File('c:\\java\\formation\\exemple');
// Nom de répertoire absolu sous Windows
File monRep2 = new File('/home/lina/java/test');
// Nom de répertoire absolu sous Unix
```

- Si vous voulez obtimiser la portabilité de votre code, il vaux mieux utiliser : File.separator

```
String s = File.separator;
File monRep3 = new File('java'+ s +'essais')
```

La gestion des fichiers

Type	Méthode	Descriptif
Boolean	createNewFile()	Créer un nouveau fichier
Boolean	Delete()	Supprime le ficheir
Boolean	Mkdir()	Créer un répertoire
Boolean	Exists()	Vérifie si le fichier existe
Boolean	isFile()	Vérifie si l'objet est un fichier
Boolean	isDirectory()	Vérifie si l'objet est un répertoire
Long	Lenth()	Fournie la taille du fichier
Boolean	canRead()	Fournie *true* si l'objet correspond à un fichier en lecture
…	…	…

Les fichiers

- Au sens linux, un fichier peut être
  - un fichier régulier : un fichier qui stocke de la donnée
  - un répertoire, un device, un socket...

- Un fichier régulier peut-être
  - un fichier texte
    - texte brut : .txt
    - texte tabulé/séparé : .csv, .tsv
    - .xml
      - Cas particulier : .xhtml
  - un fichier binaire (suite de 0 et de 1 qui affiche des caractères cabalistiques dans un éditeur texte)
    - vidéo, son, image...

Les fichiers - Introduction

- Les flux permettent à un programme Java de lire ou écrire dans :
  - des fichiers,
  - des sockets,
  - les flux standards associés à un processus (`System.in`, `System.out`, `System.err`)

- Utilisation d'un flux :
  - **Ouverture du flux,**
  - Si le flux a été ouvert avec succès :
    - **Lecture ou écriture dans le flux,**
    - **Fermeture du flux.**

Les fichiers - Introduction

- Deux types de flux :
  - in / reader : flux lu par le programme.
  - out / writer : flux dans lequel le programme écrit.

FileInputStream

- Cette classe permet de lire un flux octet par octet.
- Si le fichier ne peut pas être **ouvert** (s'il n'existe pas, ou si les droits sont insuffisants), une exception de type FileNotFoundException sera levée.
  - Ainsi on ne passera pas dans le code qui **manipule** et **ferme** le fichier si le flux n'a pas pu être ouvert !

```
import java.io.*;
public class Main {
public static void main(String[] args) {
     FileInputStream fis = null;
     try {
          // Instanciation du FIleInputStream
          fis = new FileInputStream(new File("test.txt"));

          // Tableau de byte de taille 8 pour la lecture du flux
          byte[] buffer = new byte[8];
          int n = 0;
          while((n = fis.read(buffer)) >= 0 ) {
               for(byte b : buffer) System.out.println(b);
          }
          fis.close();
     } catch(FileNotFoundException e) {
          e.printStackTrace();
     } catch(IOException e) {
          e.printStackTrace();
     }
}
}
```

Ouverture

Manipulation

Fermeture

FileOutputStream

- Cette classe permet l'écriture de données dans un fichier octet par octet.
  - Pour écrire à la suite du fichier (append) on passe en 2eme paramètre la valeur **`true`**.
  - *exists()* teste l'existence d'un fichier.
- Si le fichier n'existe pas, il sera créé automatiquement.

```
import java.io.*;

public class Main {
public static void main(String[] args)
{
     FileOutputStream fos = null;
     try {
          fos = new FileOutputStream(new File("test.txt"), true);

          fos.write(new byte[]{'A'}); // Objet anonyme

          fos.close();
     } catch(FileNotFoundException e) {
          e.printStackTrace();
     } catch(IOException e) {
          e.printStackTrace();
     }
}
}
```

Ouverture

Manipulation

Fermeture

BufferedReader

- Cette classe permet la lecture de données dans un fichier ligne par ligne ou octet par octet.
  - En cas de problèmes une exception est levée.

```
import java.io.*;
public class Main {
public static void main(String[] args) {
     BufferedReader br = null;
     String ligne = null;
     try {
          // Instanciation du BufferedReader
          br = new BufferedReader(new FileReader("test.txt"));

          // Affectation de la lecture de la ligne suivante dans ligne
          String ligne = br.readLine();
               System.out.println(ligne);

          br.close();
     } catch(FileNotFoundException e) {
          e.printStackTrace();
     } catch(IOException e) {
          e.printStackTrace();
     }
}
}
```

Ouverture

Manipulation

Fermeture

BufferedWriter

- Cette classe permet l'écriture de données dans un fichier ligne par ligne ou octet par octet.
  - Pour écrire sans effacer le contenu, mettre en 2eme paramètre, la valeur true.
  - `.exists()` de la classe File permet de tester l'existence d'un fichier.
- Si le fichier n'existe pas, il sera créé automatiquement.

```
package test;
import java.io.*;

public class Main {
    public static void main(String[] args) {
        BufferedWriter bw = null;
        try {
            bw = new BufferedWriter(new FileWriter("test.txt") true));

            bw.write("Bonjour");

            bw.close();

        } catch(FileNotFoundException e) {
            e.printStackTrace();
        } catch(IOException e) {
            e.printStackTrace();
        }
    }
}
```

Ouverture

Manipulation

Fermeture