Développement en JAVA

Système de fichier

Agenda

- Notion de fichier
- La classe Fichier
- Les opérations de lecture
- Les opérations d’écriture
- La sérialization des objets

Notion de fichier

- Un fichier en Java est représenté par la classe `java.io.File`
- Un fichier se caractérise par son chemin sur le système de fichier (dont le format peut varier selon l'OS)
- La classe Fichier offre un certain nombre de méthode pour inspecter le fichier cible:
  - Le fichier est-il un répertoire ?
    boolean isDirectory()
  - le fichier existe-t-il ?
    boolean exists()
  - le fichier est-il caché ?
    boolean isHidden()
  - etc…

La classe Fichier

```
String path = "C:/tmp/unFichier.txt";
File f = new File(path);

if(f.exists(){
  // On peut travailler avec le fichier
}else{
  f.createNewFile();
}
```

Lecture du contenu d'un fichier

- La classe `java.io.FileReader` permet de lire le contenu d'un fichier en mode texte
- Elle hérite de la classe `java.io.InputStreamReader` et notamment des méthodes suivantes:
  - Caractère par caractère avec la méthode
    ```
    int read() // Bloquant si pas de caractères
    ```
  - Bloc par bloc avec la méthode
    ```
    int read(char[] buff, int offset, int lenght)
    ```
  - Il y a-t-il des caractères à lire
    ```
    boolean ready()
    ```

Lecture du contenu d'un fichier

```
File f = new File("C:/tmp/unFichier.txt");
FileReader fr;

try{
  fr = new FileReader("tmp/bonjour.text") ;
  int bufferSize = 1024;
  char[] buffer = new char[bufferSize];
  int n = fr.read(buffer,0, bufferSize);
}
catch(IOException e){
}
catch(FileNotFoundException e){
}finally{
  if(fr != null){
    fr.close();
  }
}
```

Bufferisation

- La bufferisation permet de faire des lectures ou des écritures par bloc plutôt que caractères par caractères
- La bufferisation augmente de beaucoup la performance des I/O car la mémoire et le disque ne travaille pas à la même vitesse
- Les classes `InputStream` et `OutputStream` sont des classes abstraites représentant des flux bufferisés
- Les implémentations `FileInputStream` et `FileOutputStream` permet de lire ou écrire dans un fichier de manière bufferisé

La classe BufferedReader

- La classe Buffered permet de lire dans un fichier en mode texte de manière bufferisée
- Elle possède notamment une méthode `String readLine()` permettant de lire un fichier ligne par ligne

```
fr = new FileReader("tmp/bonjour.text") ;
BufferedReader br =  new BufferedReader(fr);
String line = null;
do{
  line = br.readLine() ;
}while(line != null);
```

Ecriture dans un fichier

- La classe FileWriter permet d'écrire dans un fichier en mode texte

```
File fichier = new File("tmp/bonjour.text");
Writer writer = null;
try{
  Writer writer = new FileWriter(fichier);
  writer.write("Bonjour le monde !");

} catch (IOException e) { }
finally{
  if(writer != null){
    writer.close();
  }
}
```

BufferedWriter et PrintWriter

- La classe BufferedWriter permet d'écrire dans un fichier en mode bufferisé
- La classe PrintWriter permet d'écrire facilement des objets dans un fichier

```
// ouverture d'un flux de sortie sur un fichier
FileWriter writer = new FileWriter(fichier) ;
// création d'un PrintWriter sur ce flux
PrintWriter pw = new PrintWriter(writer) ;
// écriture d'un marin dans le fichier
Marin m = new Marin("Surcouf", "Robert") ;
// la méthode toString() est appelée
pw.println(m) ;
```

Flush du buffer

- La méthode `flush()` permet de vide un buffer
- Ce qui permet notamment de déclencher l'écriture sur le disque avant que celui soit plein

```
FileWriter writer = new FileWriter(fichier);
PrintWriter pw = new PrintWriter(writer);

pw.println("Hello World");
pw.flush(); // Le contenu du buffer est écrit sur le disque
```

L’interface serializable

- L’interface serializable est une interface qui ne comporte aucune méthode
- Elle permet à la JVM de s’assurer que le programmeur sérialize bien du contenu qui peut l’être
- L’ensemble des attributs de la classe doivent eux même être sérializable
  - Si jamais ce n’est pas le cas, une exception `NotSerializableException` sera levée
- Le mot clé `transient` permet de marquer un attribut comme non sérializable, il ne sera pas sauvegardé

L’interface serializable

- La sérialisation transforme un objet et un tableau d’octets
- Notion de serialVersionUID

Serialization d'un object

- La classe ObjectOutputStream permet de sérialiser un objet, elle se construit à partir d'un objet OutputStream

```
File fichier = new File("tmp/marin.ser") ;
// ouverture d'un flux sur un fichier
ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fichier)) ;

// création d'un objet à sérializer
Marin m = new Marin("Surcouf", "Robert") ;

// sérialization de l'objet
oos.writeObject(m) ;
```

Déserialization d'un object

- La classe ObjectInputStream permet de désérialiser un objet. Elle se construit à partir d'un objet de type InputStream

```
File fichier = new File("tmp/marin.ser") ;
// ouverture d'un flux sur un fichier
ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(fichier)) ;
// désérialization de l'objet
Marin m = (Marin)ois.readObject() ;
System.out.println(m) ;
// fermeture du flux dans le bloc finally
```