Lectura de un fichero de texto en java

Podemos *abrir un fichero de texto* para leer usando la clase *FileReader*. Esta clase tiene métodos que nos permiten leer caracteres. Sin embargo, suele ser habitual querer las líneas completas, bien porque nos interesa la línea completa, bien para poder analizarla luego y extraer campos de ella. *FileReader* no contiene métodos que nos permitan leer líneas completas, pero sí *BufferedReader*. Afortunadamente, podemos construir un *BufferedReader* a partir del *FileReader* de la siguiente forma:

```
File archivo = new File ("C:\\archivo.txt");
FileReader fr = new FileReader (archivo);
BufferedReader br = new BufferedReader(fr);
...
String linea = br.readLine();
```

La apertura del fichero y su posterior lectura pueden lanzar excepciones que debemos capturar. Por ello, la apertura del fichero y la lectura debe meterse en un bloque *try-catch*.

Además, el fichero hay que cerrarlo cuando terminemos con él, tanto si todo ha ido bien como si ha habido algún error en la lectura después de haberlo abierto. Por ello, se suele poner al *try-catch* un bloque *finally* y dentro de él, el *close()* del fichero. Solo hacer notar que el fichero se mete dentro de un **FileReader** y este dentro de un **BufferedReader**. Todos tienen dentro el mismo fichero abierto una sola vez. Por ello, basta con hacer **close()** de cualquiera de ellos para cerrar el fichero.

El siguiente es un código completo con todo lo mencionado.

```
import java.io.*;

class LeeFichero {
   public static void main(String [] arg) {
      File archivo = null;
      FileReader fr = null;
      BufferedReader br = null;

      try {
         // Apertura del fichero y creacion de BufferedReader para
poder
         // hacer una lectura comoda (disponer del metodo readLine()).
         archivo = new File ("C:\\archivo.txt");
         fr = new FileReader (archivo);
         br = new BufferedReader(fr);

         // Lectura del fichero
         String linea;
         while((linea=br.readLine())!=null)
            System.out.println(linea);
      }
      catch(Exception e){
         e.printStackTrace();
      }finally{
         // En el finally cerramos el fichero, para asegurarnos
         // que se cierra tanto si todo va bien como si salta
         // una excepcion.
         try{
```

```
            if( null != fr ){
               fr.close();
            }
         }catch (Exception e2){
            e2.printStackTrace();
         }
      }
   }
}
```

Como opción para leer un fichero de texto línea por línea, podría usarse la clase *Scanner* en vez de el *FileReader* y el *BufferedReader*. Ver el ejemplo del Ejemplo de lectura de un fichero con Scanner

Escritura de un fichero de texto en java

El siguiente código **escribe un fichero de texto** desde cero. Pone en él 10 líneas

```
import java.io.*;

public class EscribeFichero
{
    public static void main(String[] args)
    {
        FileWriter fichero = null;
        PrintWriter pw = null;
        try
        {
            fichero = new FileWriter("c:/prueba.txt");
            pw = new PrintWriter(fichero);

            for (int i = 0; i < 10; i++)
                pw.println("Linea " + i);

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
           try {
           // Nuevamente aprovechamos el finally para
           // asegurarnos que se cierra el fichero.
           if (null != fichero)
              fichero.close();
           } catch (Exception e2) {
              e2.printStackTrace();
           }
        }
    }
}
```

Si queremos **añadir al final de un fichero** ya existente, simplemente debemos poner un flag a true como segundo parámetro del constructor de **FileWriter**.

```
 FileWriter fichero = new FileWriter("c:/prueba.txt",'''true''');
```

Ficheros binarios

Para ficheros binarios se hace exactamente igual, pero en vez de usar los "**Reader**" y los "**Writer**", se usan los "**InputStream**" y los "**OutputStream**". En lugar de los **readLine()** y **println()**, hay que usar los métodos **read()** y **write()** de array de bytes.

El siguiente ejemplo hace una copia binaria de un fichero

```
package chuidiang.ejemplos;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;

public class CopiaFicheros {

   public static void main(String[] args) {
      copia ("c:/ficheroOrigen.bin", "c:/ficheroDestino.bin");
   }

   public static void copia (String ficheroOriginal, String
ficheroCopia)
   {
      // Se declaran fuera del try porque los necesitamos dentro del
finally para poder cerrarlos.
      BufferedInputStream bufferedInput = null;
      BufferedOutputStream bufferedOutput = null;

      try {
         // Se abre el fichero original para lectura
         FileInputStream fileInput = new
FileInputStream(ficheroOriginal);
         bufferedInput = new BufferedInputStream(fileInput);

         // Se abre el fichero donde se hará la copia
         FileOutputStream fileOutput = new FileOutputStream
(ficheroCopia);
         bufferedOutput = new BufferedOutputStream(fileOutput);

         // Bucle para leer de un fichero y escribir en el otro.
         byte [] array = new byte[1000];
         int leidos = bufferedInput.read(array);
         while (leidos > 0)
         {
            bufferedOutput.write(array,0,leidos);
            leidos=bufferedInput.read(array);
         }

      } catch (Exception e) {
         e.printStackTrace();
      } finally {
         try {
            if (null != bufferedInput) {
               bufferedInput.close();
            }
         } catch (Exception e2) {
            e2.printStackTrace();
         }
```

```
            try {
               if (null != bufferedOutput) {
                  bufferedOutput.close();
               } catch (Exception e3) {
                  e3.printStackTrace();
               }
            }
         }
      }
```

Los Buffered*

Si usamos sólo **FileInputStream**, **FileOuputStream**, **FileReader** o **FileWriter**, cada vez que hagamos una lectura o escritura, se hará fisicamente en el disco duro. Si escribimos o leemos pocos caracteres cada vez, el proceso se hace costoso y lento, con muchos accesos a disco duro.

Los **BufferedReader**, **BufferedInputStream**, **BufferedWriter** y **BufferedOutputStream** añaden un buffer intermedio. Cuando leamos o escribamos, esta clase controlará los accesos a disco.

- Si vamos escribiendo, se guardará los datos hasta que tenga basantes datos como para hacer la escritura eficiente.
- Si queremos leer, la clase leerá muchos datos de golpe, aunque sólo nos dé los que hayamos pedido. En las siguientes lecturas nos dará lo que tiene almacenado, hasta que necesite leer otra vez.

Esta forma de trabajar hace los accesos a disco más eficientes y el programa correrá más rápido. La diferencia se notará más cuanto mayor sea el fichero que queremos leer o escribir.