EJERCITACIÓN JAVA

Aplications

2012

Archivos y Flujos

CÁTEDRA DE PROGRAMACIÓN AVANZADA

Ings. Mario Bressano

ENVÍO 01

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 nombreArchivo = new File ("C:\\archivo.txt");
FileReader nombreFileReader = new FileReader (nombreArchivo);
BufferedReader nombreBufferedReader = new BufferedReader(nombreFileReader);
...
String linea = nombreBufferedReader.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.

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
      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,
      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 del **FileReader** y el **BufferedReader.**

Ejemplo de lectura de un fichero con Scanner

Supongamos que tenemos un fichero en el que en cada línea hay los datos de una persona. Pueden ser un id, un nombre y una edad, separados por comas y quizás espacios.

1, Pedro , 33
2, Juan, 44
4, Antonio, 55

Vamos a hacer y explicar un pequeño programa en java usando Scanner que nos permita leer estos datos.

En primer lugar, creamos un **File** con el contenido del fichero y después una instancia de **Scanner** pasándole ese **File.**

```
File f = new File("fichero.txt");
Scanner s;
try {
  s = new Scanner(f);
  //
  // Aquí la lectura del fichero
  //
  s.close();
} catch (FileNotFoundException e) {
  e.printStackTrace();
}
```

Para la lectura del fichero, bastará de momento con un bucle para ir leyendo línea a línea. Para ello, podemos usar el método **hasNextLine()** que nos indica si hay o no una línea más que leer, y el método **nextLine()** que nos la devuelve

```
while (s.hasNextLine()) {
  String linea = s.nextLine();
  //
  // Aquí el tratamiento de la línea
  //
}
```

Para tratar la línea y sacar los tres campos que hay en ella, podemos usar nuevamente otra instancia de la clase Scanner. El delimitador para los campos será una coma, precedida o no de uno o más espacios y seguida o no de uno o más espacios. Eso, usando Expresiones Regulares en Java se expresa así "\\s*,\\s*", donde \\s indica un espacio blanco y con asterisco detrás \\s* indica cero o más espacios en blanco. Por tanto, el código para recoger los tres campos, puede ser como este

```
Scanner sl = new Scanner(linea);
sl.useDelimiter("\\s*,\\s*");
System.out.println(sl.next());
System.out.println(sl.next());
System.out.println(sl.next());
```

Y si ahora ponemos el programa completo

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class FileScanner {
	public static void main(String[] args) {
		File f = new File("fichero.txt");
		Scanner s;
		try {
			s = new Scanner(f);
			while (s.hasNextLine()) {
				String linea = s.nextLine();
				Scanner sl = new Scanner(linea);
				sl.useDelimiter("\\s*,\\s*");
				System.out.println(sl.next());
				System.out.println(sl.next());
				System.out.println(sl.next());
			}
			s.close();
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}
	}
}
```

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

Este pequeño articulo muestra cono es posible escribir cadenas de texto en un archivo existente.

Primero se indica la ruta en donde se localiza el archivo(si el documento no existe lo crea en el lugar en donde esta indicado), y se asigna a un objeto de tipo File:

```
File archivo = new File ("ruta_del_documento");
```

A continuación, se crea un objeto de tipo FileWriter, se asocia con el objeto File y se activa:

FileWriter escribirArchivo = new FileWriter(archivo, true);

y finalmente usando el método writer se escribe las cadenas en el Documento:

fw.write("hola Mundo esto es" +
" una prueba de escritura”);

Una cosa importante es no olvidar cerrar el documento.

fw.close();

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

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

                // Cierre de los ficheros
                bufferedInput.close();
                bufferedOutput.close();
        }
        catch (Exception e)
        {
                e.printStackTrace();
        }
    }
}
```

Si usamos sólo **FileInputStream, FileOuputStream, FileReader o FileWriter,** cada vez que hagamos una lectura o escritura, se hará físicamente 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 bastantes 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.