**Tema11.FICHEROS**

### Ficheros de texto en Java (Emplean flujos de caracteres ( 16 bits) )

Un fichero de texto está formado por secuencias de caracteres, organizados en líneas de igual o distinta longitud.

## Escritura de un fichero de texto en java (crear y escribir)

Para escribir en un fichero de texto utilizaremos dos clases:

**FileWriter** y **PrintWriter**.

La clase FileWriter permite tener acceso al fichero en modo escritura.

Para **crear** objetos FileWriter podemos utilizar los constructores:

FileWriter(String path)

FileWriter(File objetoFile);

El fichero se crea y si ya existe su contenido se pierde.

Si lo que necesitamos es abrir un fichero de texto existente sin perder su contenido y añadir más contenido al final utilizaremos los constructores:

FileWriter(String path, boolean append)

FileWriter(File objetoFile, boolean append)

Si el parámetro **append** es **true** significa que los datos se van a añadir a los existentes. Si es **false** los datos existentes se pierden.

FileWriter fichero = new FileWriter("prueba.txt",**true**);

*Lo mejor es utilizar el constructor del path, de esta forma el fichero se creará en la misma carpeta donde se encuentre el archivo.java*

*FileWriter fw = new FileWriter("src\\texto\\datos.txt");*

*(supongo que lo tengo en un package o carpeta texto)*

La clase **FileWriter**proporciona el **método write()** para escribir cadenas de caracteres aunque lo normal es utilizar esta clase junto con la clase PrintWriter para facilitar la escritura.

La clase **PrintWriter** permite **escribir** caracteres en el fichero **de la misma forma que en la pantalla**.

Un objeto PrintWriter se crea a partir de un objeto FileWriter.

**Ejemplo**:

FileWriter fw = new FileWriter("src\\carpeta\\datos.txt");

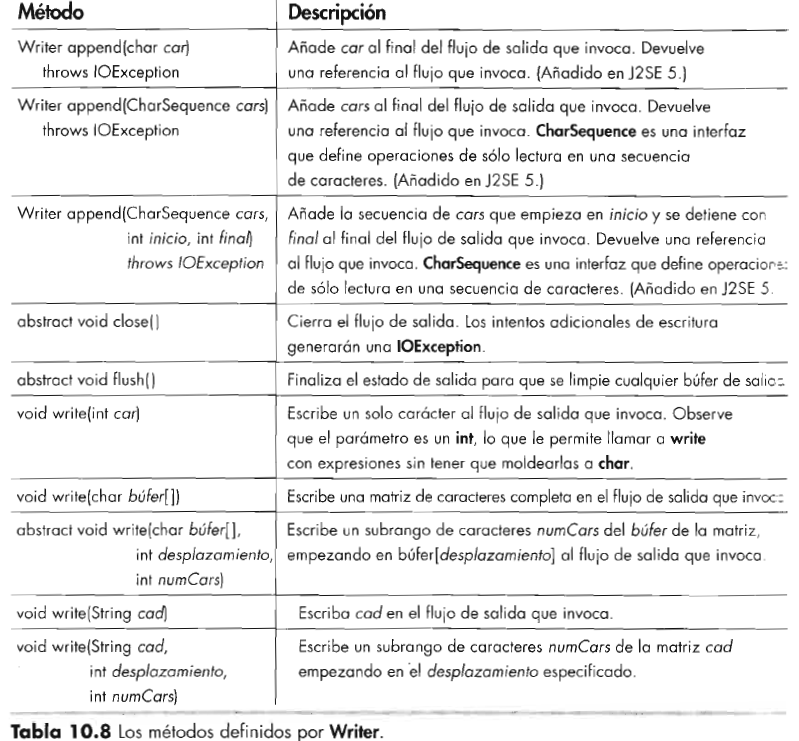
PrintWriter salida = new PrintWriter(fw);

A partir de Java 5 se puede crear un objeto PrintWriter directamente a partir de un objeto File o de la ruta:

**PrintWriter salida = new PrintWriter("src\\carpeta\\datos.txt");**

En este caso, si el fichero no existe se crea. Si no se puede crear un archivo con ese nombre o si ocurre algún error se lanza una **excepción** **FileNotFoundException**.

Una vez creado el objeto podemos utilizar **print(), println() y printf()** para escribir en el fichero como si fuese en pantalla.



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

|  |
| --- |
| package writerYreader;  import java.io.\*;    public class EscribeFichero1  {      public static void main(String[] args)      {          FileWriter fichero = null;          PrintWriter salida = null;          try          {              fichero = new FileWriter("src\\writerYreader\\fichero1.txt");              salida = new PrintWriter(fichero);                for (int i = 0; i < 10; i++)                  salida.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();             }          }      }  } |

**Otro Ejemplo:**

Programa Java que lee texto por teclado y lo escribe en un fichero de texto. La lectura de texto por teclado acaba cuando se lee la cadena FIN.

import java.io.\*;

public class EscribeFichero2 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        PrintWriter salida = null;//objeto que permite escribir en fichero

        try {

**salida = new PrintWriter("fichero2.txt");//comprimido**

**salida = new PrintWriter("src\\paquete\\fichero2.txt");//MEJOR ruta**

            String cadena;

            System.out.println("Introduce texto. Para acabar introduce la cadena FIN:");

            cadena = sc.nextLine();

            while (!cadena.equalsIgnoreCase("FIN")) {

                salida.println(cadena);

                cadena = sc.nextLine();

            }

            salida.flush();//limpiar buffer

        } catch (FileNotFoundException e) {

            System.out.println(e.getMessage());

        } finally {

            salida.close();

        }

    }

}

El método **flush()** provoca que se escriban en el fichero los datos que puedan haber en el buffer de salida.

El método **close()** cierra la conexión con el fichero y libera los recursos que está usando la conexión.

**Lectura de un fichero de texto en java**

Para leer en un fichero de texto utilizaremos dos clases:

**FileReader** y **BufferedReader**.

La clase FileReader permite tener acceso al fichero en modo lectura.

Para crear objetos FileReader podemos utilizar los constructores:

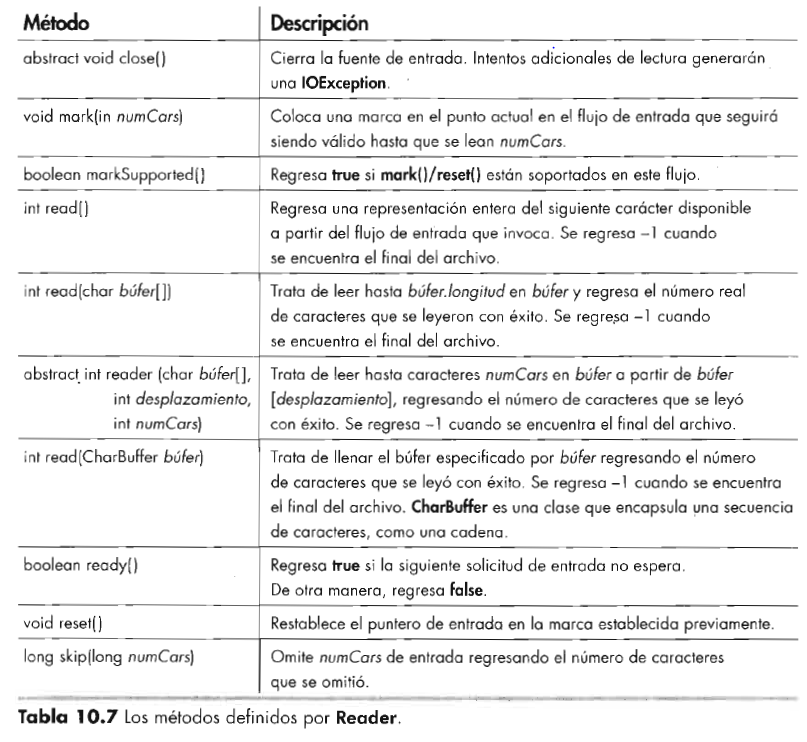
FileReader(String ruta)

FileReader(File objetoFile);

Ambos lanzan una **excepción FileNotFoundException** si el fichero no existe.

La clase **FileReader** proporciona el **método read() para leer caracteres** del fichero aunque lo normal es realizar la lectura mediante la clase BufferedReader.

Para leer utilizando la clase BufferedReader se debe crear un objeto BufferedReader a partir de un objeto FileReader:

****

**Ejemplo:**

FileReader fr = new FileReader("datos.txt");

BufferedReader entrada = new BufferedReader (fr);

Una vez creado el objeto BufferedReader podemos utilizar:

-    El **método readLine()** para leer líneas de texto del fichero (String). Este método devuelve null cuando no hay más líneas para leer.

-      El **método read()** para leer carácter a carácter. Devuelve un entero que representa el código Unicode del carácter leído. Devuelve -1 si no hay más caracteres.

Ambos métodos lanzan una **excepción IOException** si ocurre un error de lectura.

El fichero se debe cerrar cuando ya no se use, mediante el método close(). Este método lanza una**excepción IOException**.

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 LeeFichero1 {

   public static void main(String [] arg) {

     // File archivo = null; //no es necesario

      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 ("fichero1.txt");//\*

         //fr = new FileReader (archivo);//\*\*

fr = new FileReader("src\\writerYreader\\fichero1.txt");//equivale a \*con\*\*

         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();

         }

      }

   }

}

Programa Java que lee el contenido del fichero creado en el ejemplo anterior.

import java.io.\*;

public class LeeFichero2 {

public static void main(String[] args) {

FileReader fr = null;

try {

fr = new FileReader("fichero2.txt");//MEJOR poner ruta

BufferedReader entrada = new BufferedReader(fr);//permite leer de fichero

String cadena = entrada.readLine();

while (cadena != null) {

System.out.println(cadena);

cadena = entrada.readLine();

}

} catch (FileNotFoundException e) {

System.out.println(e.getMessage());

} catch (IOException e) {

System.out.println(e.getMessage());

} finally {

try {

if (fr != null) {

fr.close();

}

} catch (IOException e) {

System.out.println(e.getMessage());

}

}

}

}

**Ejemplo:**

Programa java que lee carácter a carácter el contenido del fichero de texto fichero2.txt ya creado.

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

public class LeeFicheroCaracter {

    public static void main(String[] args) {

        FileReader fr = null;

        try {

            fr = new FileReader("fichero2.txt");

            BufferedReader entrada = new BufferedReader(fr);

            int car = entrada.read();

            while (car != -1) {

                System.out.print((char) car);

                car = entrada.read();

            }

        } catch (FileNotFoundException e) {

            System.out.println(e.getMessage());

        } catch (IOException e) {

            System.out.println(e.getMessage());

        } finally {

            try {

                if (fr != null) {

                    fr.close();

                }

            } catch (IOException e) {

                System.out.println(e.getMessage());

            }

        }

    }

}

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

**LECTURA DE FICHEROS DE TEXTO CON SCANNER**

A partir de Java 5 se puede leer un fichero de texto utilizando la clase Scanner igual que si leyéramos por teclado.

Para ello se le pasa al constructor de Scanner el objeto File asociado al fichero.

Esta operación lanza una **excepción FileNotFoundException**.

**Ejemplo de lectura de un fichero de texto con Scanner**:

Programa que lee el contenido del fichero fichero2.txt utilizando la clase Scanner.

Se utiliza el **método hasNext()** de Scanner para saber si quedan más datos que leer en el fichero.

import java.io.File;

import java.io.\*;

import java.util.Scanner;

public class LeeFicheroconScan {

public static void main(String[] args) {

File f = new File("fichero2.txt");

String cadena;

Scanner entrada = null;

try {

entrada = new Scanner(f);

while (entrada.hasNext()) {

cadena = entrada.nextLine();

System.out.println(cadena);

}

} catch (FileNotFoundException e) {

System.out.println(e.getMessage());

} finally{

entrada.close();

}

}

}

**Los Buffered\***

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.