Ficheros de texto en Java

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

Todos los datos que aparecen en estos ficheros están formados por caracteres.

CREAR Y ESCRIBIR EN FICHEROS DE TEXTO EN JAVA

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.

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("c:/ficheros/datos.txt");
```

```
PrintWriter salida = new PrintWriter(fw);
```

En este ejemplo se ha creado un fichero de texto llamado datos.txt. El fichero se encuentra dentro de la carpeta ficheros en la unidad C:

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("c:/ficheros/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 **excepciónFileNotFoundException**.

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

Ejemplo de escritura de un fichero de texto:

Programa Java que lee texto por teclado y lo escribe en un fichero de texto llamado datos.txt. El proceso consiste en leer una línea de texto por teclado y escribirla en el fichero. Este proceso se repite hasta que se introduce por teclado la cadena FIN. La cadena FIN que indica el final de lectura no se debe escribir en el fichero.

```
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;
public class File11 {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      PrintWriter salida = null;
      try {
         salida = new PrintWriter("c:/ficheros/datos.txt"); //se crea el fichero
         String cadena;
         System.out.println("Introduce texto. Para acabar introduce la cadena FIN:");
         cadena = sc.nextLine(); //se introduce por teclado una cadena de texto
         while (!cadena.equalsIgnoreCase("FIN")) {
            salida.println(cadena);  //se escribe la cadena en el fichero
            cadena = sc.nextLine(); //se introduce por teclado una cadena de texto
         }
         salida.flush();
      } 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.

A partir de Java 7 se puede usar la instrucción *try-with-resources*. Un *resource* (recurso) es un objeto que necesita ser cerrado después de usarlo. Una instrucción *try-with-resources* asegura que estos objetos serán cerrados al final de la instrucción try. Cualquier objeto que implemente la interface java.lang.AutoCloseable, entre ellos los que implementan la interface java.io.Closeable pueden ser usados como resources.

El ejemplo anterior se escribiría así:

```
import java.io.FileNotFoundException;

import java.io.PrintWriter;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        String cadena;

        try (PrintWriter salida = new PrintWriter("c:/ficheros/datos.txt")) {

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

            cadena = sc.nextLine();

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

```
            salida.println(cadena);
            cadena = sc.nextLine();
        }
    } catch (FileNotFoundException e) {
        System.out.println(e.getMessage());
    }
  }
}
```

A continuación del try se escribe el recurso entre paréntesis. Ahora no es necesario escribir el bloque finally para cerrar el fichero.

La instrucción try puede contener varios recursos, en este caso irán separados por punto y coma.

Ejemplo: Programa que leer por teclado líneas de texto y las añade al final del ficheros datos.txt. Para resolverlo vamos a modificar el programa anterior para que añada texto al fichero datos.txt, es decir, al volver a ejecutar el programa el contenido anterior del fichero no se pierde y el contenido nuevo se añade al final.

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String cadena;
```

```
    try (FileWriter fw = new FileWriter("c:/ficheros/datos.txt", true);
         PrintWriter salida = new PrintWriter(fw)) {
        System.out.println("Introduce texto. Para acabar introduce la cadena FIN:");
        cadena = sc.nextLine();
        while (!cadena.equalsIgnoreCase("FIN")) {
            salida.println(cadena);
            cadena = sc.nextLine();
        }
    } catch (IOException ex) {
        System.out.println(ex.getMessage());
    }
  }
}
```

LECTURA DE FICHEROS 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("c:/ficheros/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**.

Ejemplo de lectura de un fichero de texto:

Programa Java que lee el contenido del fichero datos.txt creado en el ejemplo anterior y lo muestra por pantalla. El proceso consiste en leer una línea del fichero y mostrarla por pantalla. El proceso se repite hasta que se llegue al final del fichero y no hayan más líneas que leer. Cuando esto ocurre el método readLine() devuelve *null*.

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class File13 {
   public static void main(String[] args) {
      FileReader fr = null;
      try {
         fr = new FileReader("c:/ficheros/datos.txt");
         BufferedReader entrada = new BufferedReader(fr);
         String cadena = entrada.readLine(); //se lee la primera línea del fichero
```

```
        while (cadena != null) {  //mientras no se llegue al final del fichero
            System.out.println(cadena); //se nuestra por pantalla
            cadena = entrada.readLine(); //se lee la siguiente línea del fichero
        }
    } 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:

Mostrar por pantalla el contenido del fichero de texto datos.txt pero en este caso lo vamos a leer carácter a carcater. El proceso consiste en leer un carácter del fichero y mostrarlo por pantalla. Este proceso se repite hasta que no queden más caracteres que leer en el fichero, es decir, hasta que se alcance el finall del fichero. En este caso el método read() devuelve -1.

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class File14 {
  public static void main(String[] args) {
    FileReader fr = null;
    try {
        fr = new FileReader("c:/ficheros/datos.txt");
        BufferedReader entrada = new BufferedReader(fr);
        int car = entrada.read(); //se lee el primer carácter del fichero
        while (car != -1) { //mientras no se llegue al final del fichero
```

```
            System.out.print((char) car); //se nuestra por pantalla
            car = entrada.read(); //se lee el siguiente carácter del fichero
        }
      } 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());
        }
      }
    }
}
```

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 línea a línea el contenido del fichero datos.txt utilizando la clase Scanner.

Se utiliza el **método hasNext()** de Scanner para saber si quedan más datos que leer en el fichero. Este método devuelve *false* si se ha llegado al final del fichero y *true* en caso contrario.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class File12 {
  public static void main(String[] args) {
    File f = new File("c:/ficheros/datos.txt");
    String cadena;
```

```
        Scanner entrada = null;
        try {
            entrada = new Scanner(f); //se crea un Scanner asociado al fichero
            while (entrada.hasNext()) { //mientras no se alcance el final del fichero
                cadena = entrada.nextLine(); //se lee una línea del fichero
                System.out.println(cadena); //se muestra por pantalla
            }
        } catch (FileNotFoundException e) {
            System.out.println(e.getMessage());
        } finally{
            entrada.close();
        }
    }
}
```

Ejemplo de lectura de ficheros de texto con Scanner:

Disponemos de un fichero de texto llamado enteros.txt que contiene números enteros. El siguiente programa lee los números y los muestra. Muestra también la cantidad de números leídos y su suma.

Por ejemplo, si el fichero enteros.txt contiene los siguientes números:
323 34 234 990 22 3 1
5463 28 34 0 7

El programa mostrará por pantalla:

323
34
234
990
22
3
1
5463
28
34
0
7
Número leídos: 12
Suma 7139

Se utilizará el **método hasNextInt()** de Scanner para saber si quedan más enteros que leer en el fichero. El método hasNextInt() devuelve *true* cuando lo siguiente que se va a extraer del fichero es un entero y devuelve false en caso contrario.

La lectura acaba cuando no quedan más enteros (se ha llegado al final del fichero) o cuando encuentra un carácter no válido como entero.

Por ejemplo, si el contenido del fichero enteros.txt es el siguiente:

323 34 KKK 234 990 22 3 1
5463 28 34 0 7

El programa mostrará por pantalla:
323
34
Número leídos: 2
Suma 357

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class File15 {
   public static void main(String[] args) {
      File f = new File("c:/ficheros/enteros.txt");
      int numero, suma = 0, cont = 0;
      Scanner entrada = null;
      try {
         entrada = new Scanner(f);
         while (entrada.hasNextInt()) {
               numero = entrada.nextInt();
               System.out.println(numero);
               suma = suma + numero;
               cont++;
         }
         System.out.println("Número leídos: " + cont);
         System.out.println("Suma " + suma);
      } catch (FileNotFoundException e) {
         System.out.println(e.getMessage());
      } finally{
         entrada.close();
```

```
    }
  }
}
```

Ejemplo de lectura de un fichero de texto con Scanner:

Disponemos de un fichero de texto llamado enteros.txt que contiene los siguientes números enteros separados por espacios en blanco o comas:

34,45,23 8, 9

12 23

El siguiente programa Java lee el contenido del fichero y muestra los números. Muestra también la cantidad de números leídos y su suma.

El programa lee líneas completas del fichero y las pasa a un StringTokenizer del que se extraen los números.

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

public class File16 {
  public static void main(String[] args) {
    File f = new File("c:/ficheros/enteros.txt");
    int numero, suma = 0, cont = 0;
    StringTokenizer st;
    Scanner entrada = null;
    String cadena;
    try {
      entrada = new Scanner(f);
      while (entrada.hasNext()) {
        cadena = entrada.nextLine();
        st = new StringTokenizer(cadena, " ,");
        while (st.hasMoreTokens()) {
          numero = Integer.parseInt(st.nextToken());
          System.out.println(numero);
          suma = suma + numero;
          cont++;
        }
```

```
        }
        System.out.println("Número leídos: " + cont);
        System.out.println("Suma " + suma);
    } catch (FileNotFoundException e) {
        System.out.println(e.getMessage());
    } finally {
        entrada.close();
    }
  }
}
```