# Entornos de desarrollo
# Bloque 2
# Tema 9: Acceso a ficheros de texto

por Nacho Cabanes y Javier Carrasco

Una de las partes importantes de toda aplicación es poder almacenar información de alguna manera. Es habitual que los programas almacén los datos que manipulan, de modo que estos puedan ser consultados posteriormente. Una primera forma sencilla de conseguirlo es empleando ficheros.

## 2.9.1. Escritura en ficheros de texto

Java incluye una gran cantidad de clases relacionadas con la entrada y salida en fichero. No las veremos todas, solo aquellas que consideramos más útiles.

Un fichero de texto se puede manejar de forma muy similar a la consola, volcando datos línea a línea con "*println*". Una de las formas más simples de conseguirlo en Java es empleando un "*PrintWriter*". A diferencia de otros lenguajes como C#, es obligatorio usar un *try-catch* para interceptar los posibles errores (al menos *FileNotFoundException*, que se puede reemplazar por *IOException*, más general):

```
import java.io.PrintWriter;
import java.io.FileNotFoundException;

public class PrintWriter1 {
    public static void main(String[] args) {
        try {
            PrintWriter printWriter = new PrintWriter ("ejemplo.txt");
            printWriter.println ("Hola!");
            printWriter.close ();
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Una alternativa menos elegante, pero más compacta, por lo que puede ser útil en fuentes de pequeño tamaño, es no interceptar la excepción, sino permitir que el propio "*main*" la pueda lanzar, añadiendo "*throws*", así:

```
import java.io.PrintWriter;
import java.io.FileNotFoundException;

public class PrintWriter1b {
    public static void main(String[] args) throws FileNotFoundException {
        PrintWriter printWriter = new PrintWriter ("ejemplo.txt");
```

```
        printWriter.println ("Hola!");
        printWriter.close ();
    }
}
```

Siendo estrictos, puede existir un problema con ese primer programa: si se produce un error durante la escritura, el fichero podría quedar abierto y existir una pequeña fuga de memoria. Para evitarlo, es preferible cerrar el fichero en el bloque final opcional de "*try-catch-finally*".

De paso, podemos interceptar cualquier error de entrada / salida con **IOException**, e importar todo lo relacionado con la entrada y salida haciendo **"import java.io.*;"** en vez de emplear varios "*import*" individuales:

```
import java.io.*;

public class PrintWriter2 {
    public static void main(String[] args) {

        PrintWriter printWriter = null;
        try {
            printWriter = new PrintWriter ("ejemplo.txt");
            printWriter.println ("Hola!");
            printWriter.println ("y...");
            printWriter.println ("hasta luego!");
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            if ( printWriter != null ) {
                printWriter.close();
            }
        }
    }
}
```

El constructor habitual de *PrintWriter* destruye el contenido fichero, en caso de que éste existiera. Si, por el contrario, se desea añadir al final de un fichero existente, se debe emplear otro constructor, que no recibirá la ruta del fichero, sino un "*BufferedWriter*", que se apoya en un FileWriter que tiene "*true*" como segundo parámetro: "*new FileWriter("ejemplo.txt", true))*", así:

```
import java.io.*;

public class PrintWriter3 {
    public static void main(String[] args) {

        PrintWriter printWriter = null;
```

```
        try {
            printWriter = new PrintWriter(new BufferedWriter(
                new FileWriter("ejemplo3.txt", true)));
            printWriter.println ("Hola otra vez!");
            printWriter.println ("y...");
            printWriter.println ("hasta luego!");
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            if ( printWriter != null )  {
                printWriter.close();
            }
        }
    }
}
```

También se podría haber empleado directamente un "*BufferedWriter*" para escribir, sin necesidad de usar la clase "*PrintWriter*", pero se trata de una clase de más bajo nivel, que se parece menos al manejo habitual de la consola, porque no incluye un método "*println*", sino que se debe usar "*write*" para escribir y "*newLine*" para avanzar de línea, como muestra este ejemplo básico:

```
import java.io.*;

class BufferedWriter1 {

    public static void main( String[] args ) {
        try {
            BufferedWriter ficheroSalida = new BufferedWriter(
                new FileWriter(new File("ejemplo2.txt")));

            ficheroSalida.write("Línea 1");
            ficheroSalida.newLine();
            ficheroSalida.write("Línea 2");
            ficheroSalida.newLine();

            ficheroSalida.close();
        }
        catch (IOException errorDeFichero) {
            System.out.println(
                "Ha habido problemas: " +
                errorDeFichero.getMessage() );
        }
    }
}
```

> **Ejercicios propuestos**

**2.9.1.1.** Crea un programa que pida dos frases al usuario y las guarde en un fichero llamado "dosFrases.txt".

**2.9.1.2.** Crea un programa que pida frases al usuario (una cantidad indeterminada, hasta que introduzca una línea vacía) y las almacene en un fichero llamado "frases.txt". Cada vez que se lance el programa, se destruirá el fichero "frases.txt" y se creará uno nuevo que lo reemplace.

**2.9.1.3.** Crea una variante del ejercicio anterior, en la que el fichero se llamará "anotaciones.txt" y no se destruirá en cada nueva ejecución, sino que se añadirán las nuevas frases al final de las existentes.

**2.9.1.4.** Crea una versión mejorada del ejercicio anterior, en la que antes de cada frase se anotará la fecha y la hora en la que se ha realizado dicha anotación. Ayúdate de `LocalDateTime.now()` para conseguir la fecha.

**2.9.1.5.** Crea un programa que pida al usuario una anchura y una altura y cree un fichero llamado "rectangulo.txt" que contenga un rectángulo de asteriscos de ese anchura y el altura. Por ejemplo, si el usuario indica anchura 5 y altura 3, el fichero contendría:

```
*****
*****
*****
```

**2.9.1.6.** Crea un programa que pregunte al usuario la cantidad de días de un mes, el número del primer día (1 para lunes, 7 para domingo) y el nombre del mes (por ejemplo, "Marzo") y cree un fichero de texto llamado "agendaMarzo.txt" (o el nombre del mes que corresponde), que contendrá un esqueleto de agenda para ese mes. Por ejemplo, si el mes es Septiembre, tiene 31 días y empieza en el día 4, el contenido del fichero sería algo como:

```
Septiembre

------------------------------------------------------------
Jueves 1:
------------------------------------------------------------
Viernes 2:
------------------------------------------------------------
Sábado 3:
------------------------------------------------------------
Domingo 4:
------------------------------------------------------------
Lunes 5:
------------------------------------------------------------
(...)
------------------------------------------------------------
Viernes 30:
------------------------------------------------------------
```

> **2.9.1.7.** Crea un programa que pregunte al usuario la cantidad de días de un mes, el número del primer día (1 para lunes, 7 para domingo) y el nombre del mes (por ejemplo, "Marzo") y cree un fichero de texto llamado "calendarioMarzo.txt" (o el nombre del mes que corresponde), con el calendario de ese mes. Por ejemplo, si el mes es Septiembre, tiene 31 días y empieza en el día 4, el contenido del fichero sería algo como:

```
Septiembre

lun mar mie jue vie sab dom

                1   2   3   4
  5   6   7   8   9  10  11
 12  13  14  15  16  17  18
 19  20  21  22  23  24  25
 26  27  28  29  30
```

## 2.9.2. Lectura de ficheros de texto

Una forma sencilla de leer de un fichero, si nos basta con hacerlo línea a línea, es emplear un "BufferedReader", que se apoyará en un "*FileReader*" y que incluye un método "*readLine*" que nos devuelve una cadena de texto (un "*String*").

Si ese String es null, quiere decir que se ha acabado el fichero y no se ha podido leer nada. Por eso, lo habitual es usar un "*while*" (o un "*do-while*") para leer todo el contenido de un fichero:

```
import java.io.*;

class BufferedReader1 {

    public static void main( String[] args ) {

        // Primero vemos si el fichero existe
        if (! (new File("ejemplo.txt")).exists() ) {
            System.out.println("No he encontrado ejemplo.txt");
            return;
        }

        // En caso de que exista, intentamos leer
        System.out.println("Leyendo fichero...");

        try {
            BufferedReader ficheroEntrada = new BufferedReader(
                    new FileReader(new File("ejemplo.txt")));

            String linea = ficheroEntrada.readLine();
            while (linea != null) {
                System.out.println(linea);
```

```
                linea = ficheroEntrada.readLine();
            }

            ficheroEntrada.close();
        }
        catch (IOException errorDeFichero) {
            System.out.println(
                "Ha habido problemas: " +
                errorDeFichero.getMessage() );
        }

        System.out.println("Fin de la lectura.");
    }
}
```

Es habitual abreviar la lectura y la comprobación, haciéndolas en una misma orden, lo que resulta más compacto pero menos legible:

```
String linea=null;
while ((linea=ficheroEntrada.readLine()) != null) {
    System.out.println(linea);
}
```

**Ejercicios propuestos**

**2.9.2.1.** Crea un programa que muestre la primera línea del contenido del fichero "DosFrases.txt".

**2.9.2.2.** Crea un programa que muestre todo el contenido del fichero "anotaciones.txt".

**2.9.2.3.** Crea un programa que muestre paginado el contenido del fichero "anotaciones.txt": tras cada 23 líneas se realizará una pausa hasta que el usuario pulse Intro.

**2.9.2.4.** Crea un programa que muestre el contenido de un fichero de texto, cuyo nombre deberá introducir el usuario. Debe avisar si el fichero no existe.

**2.9.2.5.** Crea un programa que lea el contenido de un fichero de texto y lo vuelque a otro fichero de texto, pero convirtiendo cada línea a mayúsculas.

**2.9.2.6.** Crea un programa que pida al usuario el nombre de un fichero y una palabra a buscar en él. Debe mostrar en pantalla todas las líneas del fichero que contengan esa palabra.

**2.9.2.7.** Crea un programa que lea el contenido del fichero "rectangulo.txt" que creaste en el ejercicio 2.9.1.5. y que calcule y muestre en pantalla cual es la anchura y la altura del rectángulo de asteriscos.

**2.9.2.8.** Crea un programa que pida al usuario el nombre de un fichero. Almacenará el contenido del fichero en un ArrayList y, a partir de ese momento, de forma repetitiva, pedirá al usuario una palabra a buscar y mostrará en pantalla todas las líneas contengan la contengan, o el texto "No encontrada" según corresponda. Se repetirá hasta que el usuario introduzca una cadena vacía.

## 2.9.3. Para profundizar

Si quieres saber más sobre el tratamiento de ficheros en JAVA puedes consultar las siguientes referencias oficiales:

- Entrada y salida mediante ficheros