UD6

LENGUAJE DE PROGRAMACIÓN JAVA

6.2 Lectura y escritura de Ficheros

MP_0485
Programación

Introducción

En este apartado vamos a trabajar con la lectura y escritura de ficheros UNICODE y binarios, para ello utilizaremos los buffer de entrada y salida.

Escritura en un fichero de texto

En este apartado tendrás oportunidad de practicar con los flujos de datos para la escritura en ficheros de texto, principalmente en formato Unicode de 16 bits.

Generalmente las operaciones de lectura / escritura requieren de un objeto iniciador que se comunique directamente con el dispositivo y un filtro con el que realizar la lectura / escritura eficientemente.

Escritura en formato Unicode con FileWriter y BufferedWriter

En el siguiente programa utilizaremos las clases FileWriter (iniciador) y BufferedWriter (filtro) para escribir el título de tres películas en un fichero de texto. Para ello realizaremos los tres pasos típicos asociados a cualquier tarea de escritura en ficheros:

- Abrir fichero para escritura.
- Escribir líneas en el fichero.
- Cerrar el fichero.

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class Main {  no usages
    public class Principal {
        public static void main(String args[]) {

            // Abrir fichero para escritura
            FileWriter file;
            try {
                file = new FileWriter( fileName: "C:\\cine\\peliculas.txt");
            } catch (IOException e) {
                System.out.println("No se puede abrir el fichero");
                System.out.println(e.getMessage());
                return;
            }
```

```
    // Abrir buffer y escribir tres líneas
    BufferedWriter buffer = new BufferedWriter(file);
    try {
        buffer.write( str: "¡Bienvenido, Mister Marshall!");
        buffer.write( str: "Con la muerte en los talones");
        buffer.newLine();
        buffer.write( str: "Muerte de un ciclista");
        buffer.newLine();
    } catch (IOException e) {
        System.out.println("Error al escribir en el fichero");
        System.out.println(e.getMessage());
    }
```

```
            // Cerrar el buffer y el fichero
            try {
                buffer.close();
                file.close();
            } catch (IOException e) {
                System.out.println("Error al cerrar el fichero");
                System.out.println(e.getMessage());
            }
        }
    }
```

Ahora vamos a analizar las partes más importantes del programa:

```
file = new FileWriter("C:\\cine\\peliculas.txt");
```

Al construir un objeto de la clase `FileWriter` estamos abriendo el fichero peliculas.txt, dejándolo preparado para escritura.

```
BufferedWriter buffer = new BufferedWriter(file);
```

Las operaciones de escritura las realizaremos a través del objeto buffer de la clase `BufferedWriter`, que nos proporciona métodos para la escritura eficiente. El constructor de la clase `BufferedWriter` requiere un objeto `FileWriter`.

```
buffer.write("¡Bienvenido, Mister Marshall!");
```

El método write(String texto) de la clase BufferedWriter escribe texto en el fichero.

```
buffer.newLine();
```

El método `newLine()` de la clase BufferedWriter escribe un retorno de carro en el fichero.

```
buffer.close();
file.close();
```

Por último, tenemos que cerrar los objetos `BufferedWriter` y `FileWriter`.

Habrás observado que, aunque el fichero peliculas.txt no existía previamente, el constructor de la clase `FileWriter` lo ha creado.

La sentencia: `file = new FileWriter("C:\\cine\\peliculas.txt");`
no solo nos ha permitido abrir el fichero para escritura, sino que también lo ha creado físicamente. Si pruebas a ejecutar varias veces el programa verás que no se van añadiendo nuevas líneas, siempre hay tres. Cada vez que ejecutamos vuelve a crear el fichero sobrescribiendo el anterior. **Si lo que deseamos es añadir líneas a un fichero existente solo tenemos que pasar un argumento más al constructor de `FileWriter` con el valor true.**

```
file = new FileWriter("C:\\cine\\peliculas.txt", true);
```

Así abrimos el fichero para añadir nuevas líneas. Si el fichero no existe, lo crea, pero si existe lo abre para añadir.

El método write de la clase `BufferedWriter` también puede recibir un número entero, que escribirá en el fichero el carácter asociado en Unicode con dicho número.

```
buffer.write(65);
```

Esta sentencia escribiría en el fichero una 'A', carácter asociado al valor numérico 65.

Usando Try-catch con recursos

Vamos a ver repetir el ejemplo anterior utilizando la estructura de un bloque try-catch con recursos, lo cual nos facilitará la sintaxis y el proceso de lectura.

```
public class Main {
        public static void main(String args[]) {
            try (FileWriter file = new FileWriter( fileName: "C:\\cine\\peliculas.txt",  append: true);
                 BufferedWriter buffer = new BufferedWriter(file);) {

                buffer.write( str: "¡Bienvenido, Mister Marshall!");
                buffer.newLine();
                buffer.write( str: "Con la muerte en los talones");
                buffer.newLine();
                buffer.write( str: "Muerte de un ciclista");
                buffer.newLine();

                System.out.println("Se ha escrito correctamente el fichero");
            } catch (IOException e) {
                System.out.println("Error en el manejo del fichero");
                System.out.println(e.getMessage());
            }
        }
}
```

Lectura de un fichero de texto

En este apartado vamos a practicar con los flujos de datos para la lectura de ficheros de texto, principalmente en formato Unicode de 16 bits. Nuestro objetivo será leer el fichero peliculas.txt utilizando flujos de datos en formato UNICODE de 16 bits.

Comenzaremos por utilizar las clases `FileReader` (iniciador) y `BufferedReader` (filtro) para realizar la lectura del fichero peliculas.txt creado en el apartado anterior.

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

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

        // Abrir fichero para lectura
        FileReader file;
        try {
            file = new FileReader( fileName: "C:\\cine\\peliculas.txt");
        } catch (IOException e) {
            System.out.println("No se puede abrir el fichero");
            System.out.println(e.getMessage());
            return;
        }
        // Abrir buffer y escribir tres líneas
        BufferedReader buffer = new BufferedReader(file);
        String linea="";
        try {
            linea = buffer.readLine();
            while (linea != null) {
                System.out.println(linea);
                linea = buffer.readLine();
            }
        } catch (IOException e) {
            System.out.println("Error al leer el fichero");
            System.out.println(e.getMessage());
        }


        // Cerrar el buffer y el fichero
        try {
            buffer.close();
            file.close();
        } catch (IOException e) {
            System.out.println("Error al cerrar el fichero");
            System.out.println(e.getMessage());
        }
    }
}
```

Analicemos ahora las partes más importantes del programa:

```
file = new FileReader("C:\\cine\\peliculas.txt");
```

Con esta sentencia estamos abriendo el fichero peliculas.txt para lectura, desencadenando una excepción del tipo `FileNotFoundException` si el fichero no existe o cualquier otra excepción de tipo `IOException` si se produce otro tipo de error que no permita la apertura del fichero.

```
linea = buffer.readLine();
```

Con esta sentencia estamos leyendo la siguiente línea del fichero de manera secuencial y guardándola en la variable `linea`. Cuando ya no hay más líneas para leer devuelve null; esa es la razón por la que necesitamos una estructura `while` con la condición `linea!=null.`

```
buffer.close();
File.close();
```

Finalmente debemos desbloquear el chero cerrando los flujos de datos de filtro e iniciador.

Como en el apartado de escritura podemos sustituir el código anterior por uno más simple utilizando la estructura try-catch con recursos.

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

public class Principal {
    public static void main(String[] args) {
        try (FileReader file = new FileReader( fileName: "C:\\cine\\peliculas.txt");
             BufferedReader buffer = new BufferedReader(file);){
            String linea="";
            linea = buffer.readLine();
            while (linea != null) {
                System.out.println(linea);
                linea = buffer.readLine();
            }

        } catch (IOException e) {
            System.out.println("Se ha producido un error en el proceso de lectura");
            System.out.println(e.getMessage());
        }
    }
}
```

Escritura de un fichero binario

Los ficheros binarios están formados por secuencias de bytes, pueden contener datos de tipo elemental (int, float, double, etc.) u objetos. El formato binario de datos es muy útil cuando se quieren representar datos en forma de registros y campos, como si de una base de datos se tratara. En esta ocasión, como ejemplo, vamos a crear un programa que permita añadir los datos de los artículos disponibles en un almacén. Comenzaremos por revisar los pasos que tenemos que seguir para lograrlo:

- Crear una clase llamada Producto, que servirá para representar la estructura de cada uno de los artículos del almacén y utilizarla para crear todos los objetos Producto que sean necesarios.

- Crearemos un objeto de la clase `FileOutputStream`, que servirá como flujo iniciador para abrir el chero almacen.dat.

- Crearemos un objeto de la clase `DataOutputStream`, que servirá como filtro proporcionando un buffer de escritura y lo vincularemos al objeto `FileOutputStream`.

- Escribiremos registros en el buffer proporcionado por el objeto `DataOutputStream`.

- Cerraremos los flujos `DataOutputStream` y `FileOutputStream`.

Vamos a crear un proyecto en nuestro con el nombre que desees y crea las clases Producto y Principal:

```
public class Producto {  no usages
    private String nombre;  3 usages
    private double precio;  3 usages
    private double unidadesEnExistencia;  4 usages

    public Producto(String nombre, double precio, double unidadesEnExiste){  no usages
     this.nombre = nombre;
     this.precio = precio;
     this.unidadesEnExistencia = unidadesEnExistencia;
    }

    public String getNombre() {  no usages
        return nombre;
    }

    public double getPrecio() {  no usages
        return precio;
    }
```

```
    public double getUnidadesEnExistencia() {  no usages
        return unidadesEnExistencia;
    }

    @Override
    public String toString() {
        return nombre + " Stock: " + this.unidadesEnExistencia + "precio: " +
                this.precio;
    }
}
```

Clase que sirve para representar la estructura de cada artículo del almacén.

```
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class Principal {
    public static void main(String[] args) {
        // Creación de 3 objetos producto
        Producto p1 = new Producto( nombre: "Manzanas Royal Gala", precio: 2.50, unidadesEnExiste: 7);
        Producto p2 = new Producto( nombre: "Dátiles de la tía Julita", precio: 3.25, unidadesEnExiste: 5);
        Producto p3 = new Producto( nombre: "Mandarinas Clementinas", precio: 2.20, unidadesEnExiste: 2);

        FileOutputStream fichero;
        DataOutputStream escritor;

    // Apertura del fichero almacen.dat
    try {
        fichero = new FileOutputStream( name: "almacen.dat", append: true);
        escritor = new DataOutputStream(fichero);
    } catch (IOException e) {
        System.out.println("No se ha podido abrir el fichero");
        System.out.println(e.getMessage());
        return;
    }
```

```
        // Escribir datos en el fichero almacen.dat
        try {
            escritor.writeUTF(p1.getNombre());
            escritor.writeDouble(p1.getPrecio());
            escritor.writeDouble(p1.getUnidadesEnExistencia());

            escritor.writeUTF(p2.getNombre());
            escritor.writeDouble(p2.getPrecio());
            escritor.writeDouble(p2.getUnidadesEnExistencia());

            escritor.writeUTF(p3.getNombre());
            escritor.writeDouble(p3.getPrecio());
            escritor.writeDouble(p3.getUnidadesEnExistencia());

        } catch (IOException e) {
            System.out.println("Ha ocurrido un error al escribir");
                    System.out.println(e.getMessage());
        }
         try {
             escritor.close();
             fichero.close();
         } catch (IOException e) {
             System.out.println("Ha ocurrido un error al cerrar");
                     System.out.println(e.getMessage());
         }

    }
}
```

Vamos a repasar las acciones que hemos llevado a cabo:

Hemos creado tres objetos de la clase Producto con los datos que posteriormente guardaremos en el fichero almacen.dat

```
// Creación de 3 objetos producto
Producto p1 = new Producto("Manzanas Royal Gala",2.50,7);
Producto p2 = new Producto("Dátiles de la tía Julita",3.25,12);
Producto p3 = new Producto("Mandarinas Clementinas",2.20,25);
```

Hemos creado un objeto de la clase `FileOutputStream`

```
fichero = new FileOutputStream("almacen.dat", true);
```

La construcción del objeto `FileOutputStream`, que actúa como flujo de datos iniciador, nos ha dejado el archivo almacen.dat abierto para escritura. Con el segundo argumento asignado a true logramos que, si el fichero ya existe, agregue los nuevos datos sin sobrescribir lo anterior. Como no hemos especificado ninguna ruta para el fichero almacen.dat, se creará en la misma carpeta donde se encuentra el proyecto.

Hemos creado un objeto `DataOutputStream`.

```
escritor = new DataOutputStream (fichero);
```

Construimos el objeto `DataOutputStream (filtro)` pasando como argumento el nuevo objeto `FileOutputStream` para proporcionar un sistema eficiente que permite guardar en disco datos de tipo elemental. En nuestro caso guardaremos un texto en formato UTF y dos valores tipo double.

Escribimos registros con ayuda del objeto `DataOutputStream (buffer)`

```
escritor.writeUTF(p1.getNombre());
escritor.writeDouble(p1.getPrecio());
escritor.writeDouble(p1.getUnidadesEnExistencia());
escritor.writeUTF(p2.getNombre());
escritor.writeDouble(p2.getPrecio());
escritor.writeDouble(p2.getUnidadesEnExistencia());
escritor.writeUTF(p3.getNombre());
escritor.writeDouble(p3.getPrecio());
escritor.writeDouble(p3.getUnidadesEnExistencia());
```

Recuerda que la clase `DataOutputStream` proporciona un `buffer` que permite escribir datos de tipo primitivo en un fichero (int, float, double, etc.) para lo cual utilizamos los siguientes métodos:

- writeBoolean(boolean valor): escribe un valor de tipo boolean en el fichero.
- writeByte(byte valor): escribe un valor de tipo byte en el fichero.

- writeBytes(String cadena): escribe cada uno de los bytes que forman parte de la cadena especificada en el argumento.
- writeChar(int valor): escribe el carácter asociado al código pasado como argumento.
- writeChars(String cadena): escribe cada uno de los caracteres de la cadena.
- writeDouble(double valor): escribe un valor de tipo double.
- writeFloat(float valor): escribe un valor de tipo float.
- writeInt(int valor): escribe un valor de tipo int.
- writeLong(long valor): escribe un valor de tipo long.
- writeShort(int valor): escribe el valor del argumento y lo almacena como un short.
- writeUTF(String cadena): escribe la cadena en formato UTF.

Cerramos los dos flujos de datos (filtro e iniciador) cerrando de esta forma el fichero.

```
escritor.close();
fichero.close();
```

Igual que en el apartado anterior podemos simplificar este código utilizando un try-catch con recursos.

```
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class Principal {
    public static void main(String[] args) {
        // Creación de 3 objetos producto
        Producto p1 = new Producto( nombre: "Manzanas Royal Gala", precio: 2.50, unidadesEnExiste: 7);
        Producto p2 = new Producto( nombre: "Dátiles de la tía Julita", precio: 3.25, unidadesEnExiste: 5);
        Producto p3 = new Producto( nombre: "Mandarinas Clementinas", precio: 2.20, unidadesEnExiste: 2);

        try (FileOutputStream fichero = new FileOutputStream( name: "almacen.dat", append: true);
             DataOutputStream escritor = new DataOutputStream(fichero);){

            escritor.writeUTF(p1.getNombre());
            escritor.writeDouble(p1.getPrecio());
            escritor.writeDouble(p1.getUnidadesEnExistencia());
```

```
            escritor.writeUTF(p2.getNombre());
            escritor.writeDouble(p2.getPrecio());
            escritor.writeDouble(p2.getUnidadesEnExistencia());

            escritor.writeUTF(p3.getNombre());
            escritor.writeDouble(p3.getPrecio());
            escritor.writeDouble(p3.getUnidadesEnExistencia());

        } catch (IOException e) {
            System.out.println("Ha ocurrido un error en el proceso");
            System.out.println(e.getMessage());
        }
    }
}
```

Lectura de un fichero binario

En este apartado abriremos el fichero almacen.dat para lectura y mostraremos en pantalla los registros que contiene, es decir, los artículos del almacén.

Para leer datos de un fichero binario tienes que saber previamente la estructura que tiene dicho fichero. En nuestro caso no hay duda, sabemos que hemos guardado registros con la estructura String, double, double y justo así es como iremos recuperándolos.

Realizaremos los siguientes pasos:

- Crear un objeto de la clase FileInputStream dejando abierto el fichero almacen.dat para lectura.
- Crear un objeto de la clase DataInputStream (filtro) asociado al objeto FileInputStream (iniciador) para obtener un buffer que permita optimizar la lectura, proporcionando métodos que permitan leer datos elementales de tipo int, float, double, etc.
- Leer datos secuencialmente hasta llegar al final del fichero.
- Cerrar los objetos DataInputStream y FileInputStream, dejando así cerrado el fichero.

Vamos a crear otra clase Principal para realizar la lectura y usaremos la misma clase Producto que ya teníamos.

```java
import java.io.DataInputStream;
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.IOException;

public class Principal {
    public static void main(String[] args) {
        FileInputStream fichero;
        DataInputStream lector;

        try{
            fichero = new FileInputStream( name: "almacen.dat");
            lector = new DataInputStream (fichero);
        } catch (IOException e) {
            System.out.println("Ha ocurrido un error al abrir el fichero");
            System.out.println(e.getMessage());
            return;
        }
       boolean eof = false;
       while (!eof) {
           try {
               String pro = lector.readUTF();
               double pre = lector.readDouble();
               double uni = lector.readDouble();
               Producto p = new Producto(pro, pre, uni);
               System.out.println(p);
           } catch (EOFException e1) {
               eof = true;
           } catch (IOException e2) {
               System.out.println("Ha ocurrido un error al leer el fichero");
                       System.out.println(e2.getMessage());
               break; // sale del bucle while
           }
       }

        try {
            lector.close();
            fichero.close();
        } catch (IOException e) {
            System.out.println("Ha ocurrido un error al cerrar el fichero");
                    System.out.println(e.getMessage());
        }
    }
}
```

Si observas el programa anterior comprobarás que tiene la típica estructura a la que ya te has acostumbrado: apertura del fichero, lectura, cierre del fichero.

Igual que en los casos anteriores podemos simplificar este código utilizando la estructura de try-catch con recursos.

```
public class Principal {
    public static void main(String[] args) {
        boolean eof = false;
        try(FileInputStream fichero = new FileInputStream( name: "almacen.dat");
            DataInputStream lector = new DataInputStream (fichero);) {
            while (!eof) {
                String pro = lector.readUTF();
                double pre = lector.readDouble();
                double uni = lector.readDouble();
                Producto p = new Producto(pro, pre, uni);
                System.out.println(p);
            }
        } catch (EOFException e1) {
            eof = true;
            System.out.println("\nSe ha leido todo el fichero");
        } catch (IOException e) {
                    System.out.println("Ha ocurrido un error al manejar el fichero");
                    System.out.println(e.getMessage());
        }
    }
}
```

Lectura de datos con DataInputStream

`DataInputStream` provee métodos para la lectura secuencial de tipos de datos elementales desde un fichero binario:

- readBoolean(): lee un valor de tipo boolean del fichero.
- readByte(): lee un valor de tipo byte del fichero.
- readChar(): lee un valor de tipo char del fichero.
- readDouble(): lee un valor de tipo double del fichero.

- readFloat(): lee un valor de tipo float del fichero.
- readInt(): lee un valor de tipo int del fichero.
- readLong(): lee un valor de tipo long del fichero.
- readShort(): lee un valor de tipo short del fichero.
- readUTF(): lee una cadena en formato UTF del fichero.

Todos estos métodos desencadenan una excepción de tipo EOFException si es el final de fichero y no hay más datos que leer.

Referencias

Apuntes elaborados a partir de la siguiente documentación:

[1] Apuntes Fernando Barber y Ricardo Ferris. Universidad de Valencia.

[2] Apuntes Programación de José Antonio Díaz-Alejo. IES Camp de Morvedre.

[3] Apuntes de Programación de Carlos Cacho y Raquel Torres. Ceedcv.

[4] Apuntes de Programación Edix Digital Workers.