Unidad 12

Lectura y escritura de información

Programación

12

Índice

12.1. Flujos de comunicación

12.1.1. Tipos de flujo
12.1.2. Flujos predeterminados
12.1.3. Utilización de flujos
12.1.4. Clases relativas a flujos

12.2. Aplicaciones del almacenamiento de información en ficheros

12.2.1. Tipos de ficheros. Registro
12.2.2. Creación y eliminación de ficheros y directorios
12.2.3. Apertura y cierre de ficheros. Modo de acceso
12.2.4. Escritura y lectura de información en ficheros
12.2.5. Almacenamiento de objetos en ficheros. Serialización
12.2.6. Utilización de los sistemas de ficheros

Introducción

Gran parte de la lógica de una programa es el tratamiento de la información, por ejemplo, para realizar una operación en una calculadora, tenemos que conocer que operación quiere el usuario y los valores de los operandos a calcular, y mostrarle el resultado. Toda la información tiene que estar definida por un flujo y conocer si la información ha de ser conservada, puede ser modificada o simplemente se usa y se descarta. En esta unidad se profundizará en el acceso a datos, su tratamiento y las distintas formas de conservación.

Al finalizar esta unidad

- Sabremos los flujos de datos y los distintos tipos.
- Aprenderemos a aplicar el tipo de flujo mas adecuado según la situación.
- Conoceremos los sistemas de ficheros.
- Aplicaremos la persistencia y el uso de la serialización
- Conoceremos las principales herramientas para trabajar con ficheros

12.1. Flujos de comunicación

En cualquier programa existe una interacción con elementos externos para la entrada de información y/o salida. Por ejemplo, cualquier entrada de información puede ser por teclado, fichero, red, etc. y de salida, por pantalla, fichero, etc.

Todos los lenguajes de programación ofrecen herramientas o mecanismos para poder hacer uso de elementos de entrada y salida. Estas herramientas trabajan sobre flujos de datos, que son una secuencia ordenada de datos que se transmiten desde una fuente hacia un destino, en Java se denomina **Stream**.

12.1.1. Tipos de flujo

Se puede clasificar los tipos de flujos según el tipo de datos, la dirección del flujo o la forma en que se accede.

1. **Según el tipo el datos.**
   a. Flujos de bytes o byte streams.

      Lectura y escritura de 0 a 255 bytes. Se suele usar para envio de datos binarios.

   b. Flujo de caracteres o character streams.

      Transmite datos tipo caracter codificados en Unicode. Se suele usar para ficheros de texto plano.

   c. Flujos estándar o predeterminados.

      Entrada por teclado y salida por pantalla.
2. **Según la dirección del flujo de datos.**

   Puede ser flujos de entrada, flujos de salida o entrada/salida.
3. **Según en la forma en que se accede.**

   Puede ser forma secuencial o acceso directo

12.1.2. Flujos predeterminados

Ya existen algunos flujos estándar de datos definidos sin necesidad de crarlos. En este caso son los flujos de entrada y salida de datos por teclado, haciendo uso de (System.out, System.err y System.in).

```
Scanner sc = new Scanner(System.in);
System.out.println("Escriba un dato:");
String dato = sc.nextLine();
System.out.println("El dato introducido es: " + dato);
// La salida err es independiente de out puede ejecutarse antes que out
System.err.println("FIN");
```

Imagen 1. Ejemplo de flujo predeterminados

12.1.3. Utilización de flujos

El funcionamiento de utilización de flujos se basa en la **apertura inicial**, hay que conectar con la fuente del flujo, la **transmisión** del flujo y su utilización, lectura o escritura y finalmente el **cierre** de la fuente de transmisión.

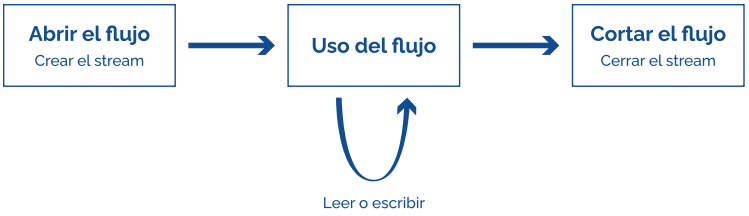

Imagen 2. Esquema de utilización de los flujos

Para los flujos predeterminados o estandar no es necesario crear o cerrar el stream debido a que estas herramientas ya estan desarrolladas.

12.1.4. Clases relativas a flujos

En java disponemos de funcionalidad para trabajar con flujos. En el paquete java.io se encuentran todas las clases que implementan la funcionalidad de entrada y salida para abstraernos y no tener que configurar los dispositivos.

Clases para el manejo de flujos de bytes

Todas las clases de manejo de flujos de byte heredan de las clases InputStream y OutputStream. A continuación, se describen las principales clases.

FileOutputStream y FileInputStream

Permite escribir y leer bytes en un fichero binario.

Escritura

```
char[] vocales = { 'A', 'E', 'I', 'O', 'U' };

String rutaFichero = "C:\\documentos\\fichero.bin";

FileOutputStream fos = new FileOutputStream(rutaFichero);
for (int i = 0; i < vocales.length; i++) {
    fos.write((byte) vocales[i]);
}

fos.close();
```

fichero.bin
1 AEIOU
2

Lectura

```
// Tiene que existir previamente el fichero, si no, lanzará una excepción
String rutaFichero = "C:\\documentos\\fichero.bin";

FileInputStream fis = new FileInputStream(rutaFichero);
int i;
while ((i = fis.read()) != -1) {
    System.out.print((char) i);
}

fis.close();
```

Console
<terminated> Fluj
AEIOU

Imagen 3. Ejemplo de FileOutputStream y FileInputStream

BufferedOutputStream y BufferedInputStream

Se usa un buffer intermedio para mejorar el rendimiento.

Escritura

```
char[] vocales = { 'A', 'E', 'I', 'O', 'U' };
String rutaFichero = "C:\\documentos\\fichero.bin";

FileOutputStream fos = new FileOutputStream(rutaFichero);
BufferedOutputStream bos = new BufferedOutputStream(fos);
for (int i = 0; i < vocales.length; i++) {
    bos.write((byte) vocales[i]);
}

bos.close();
fos.close();
```

fichero.bin
1 AEIOU
2

Lectura

```
// Tiene que existir previamente el fichero, si no, lanzará una excepción
String rutaFichero = "C:\\documentos\\fichero.bin";

FileInputStream fis = new FileInputStream(rutaFichero);
BufferedInputStream bis = new BufferedInputStream(fis);

int i;
while ((i = bis.read()) != -1) {
    System.out.println((char) i);
}

bis.close();
fis.close();
```

Console
<terminated> Fluj
A
E
I
O
U

Imagen 4. Ejemplo de BufferedOutputStream y BufferedInputStream

DataOutputStream y DataInputStream

En un fichero binario todos sus datos estan representados por bytes, con esta clase, es posible trabajar directamente con tipo de datos primitivos sobre un fichero binario.

Escritura

```
//Tiene que existir previamente el fichero, si no, lanzará una excepción
String rutaFichero="C:\\documentos\\fichero.bin";

FileOutputStream fos = new FileOutputStream(rutaFichero);
DataOutputStream dos = new DataOutputStream(fos);
dos.writeChars("AEIOU");

dos.close();
fos.close();
```

Lectura

```
String rutaFichero="C:\\documentos\\fichero.bin";

FileInputStream fis = new FileInputStream(rutaFichero);
DataInputStream dis = new DataInputStream(fis);

while(dis.available()>0) {

    System.out.print(dis.readChar());
}

dis.close();
fis.close();
```

Imagen 5. Ejemplo de DataOutputStream y DataInputStream

ByteArrayOutputStream y ByteArrayInputStream

Con estas clases podemos trabajar directamente con arrays de Byte y luego cargarlas al flujo para que se escriba o se lea. Para este tipo de clases hace falta invocar el metodo flush para consolidar los datos escritos en el fichero.

```
char[] vocales = { 'A', 'E', 'I', 'O', 'U' };
String rutaFichero = "C:\\documentos\\fichero.bin";

FileOutputStream fos = new FileOutputStream(rutaFichero);
ByteArrayOutputStream bos = new ByteArrayOutputStream();

for (int i = 0; i < vocales.length; i++) {
    bos.write((byte) vocales[i]);
}

bos.writeTo(fos);
bos.flush();
bos.close();
fos.close();
```

```
// Tiene que existir previamente el fichero, si no, lanzará una excepción
String rutaFichero = "C:\\documentos\\fichero.bin";

FileInputStream fis = new FileInputStream(rutaFichero);
ByteArrayInputStream bis = new ByteArrayInputStream(fis.readAllBytes());

int contenido = 0;
while ((contenido = bis.read()) != -1) {
    System.out.print((char) contenido);
}

bis.close();
fis.close();
```

Imagen 6. Ejemplo de ByteArrayOutputStrem y ByteArrayInputStream

PrintStream

Permite agregar la capacidad de imprimir datos a un flujo de bytes de datos concreto.

```
String rutaFichero = "C:\\documentos\\fichero.bin";

FileOutputStream fos = new FileOutputStream(rutaFichero);
PrintStream ps = new PrintStream(fos);
ps.println("AEIOU");
ps.printf("AE%sOU", 'I');
ps.close();
fos.close();

ps = new PrintStream(System.out);
ps.println("AEIOU");

ps.close();
fos.close();
```

Imagen 7. Ejemplo de PrintStream

Clases para el manejo de flujos de caracteres

Todas las clases que manejan flujos de caracteres heredan de Writer y Reader. A continuación, se describen las principales clases.

FileWriter y FileReader:

Permite escribir y leer de un fichero de texto.

Escritura

```
String rutaFichero = "C:\\documentos\\fichero.txt";

FileWriter fw = new FileWriter(rutaFichero);
fw.write("Hola Mundo");
fw.close();
```

Lectura

```
// Tiene que existir previamente el fichero, si no, lanzará una excepción
String rutaFichero = "C:\\documentos\\fichero.txt";

FileReader fr = new FileReader(rutaFichero);

int i;
while ((i = fr.read()) != -1) {
    System.out.print((char) i);
}

fr.close();
```

Imagen 8. Ejemplo deFileWriter y FileReader

BufferedWriter y BufferedReader:

Igual que la anterior, pero haciendo uso de un buffer intermedio para mejorar el rendimiento.

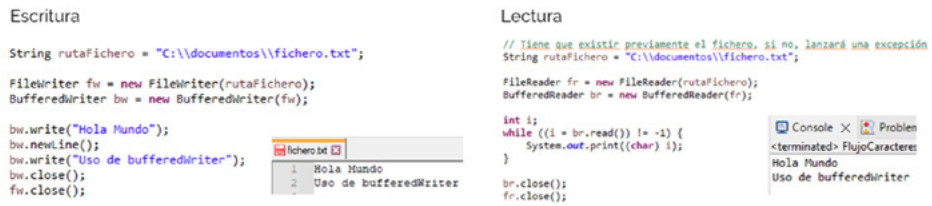

Imagen 9. Ejemplo de BufferedWriter y BufferedReader

CharArrayWriter y CharArrayReader:

Con estas clases podemos trabajar directamente con arrays de caracteres y luego cargarlas al flujo para que se escriba o se lea. Para este tipo de clases hace falta invocar el metodo flush para consolidar los datos escritos en el fichero.

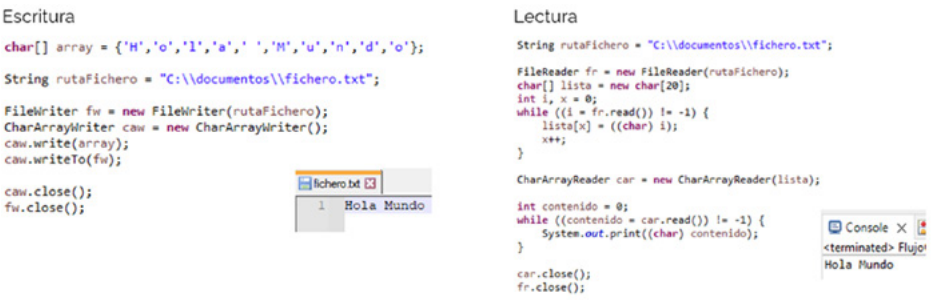

Imagen 10. Ejemplo de CharArrayWriter y CharArrayReader

PrintWriter

Tiene la capacidad de imprimir datos a un flujo de caracteres de datos concreto. Como se puede ver en la imagen, puede imprimir en un fichero o consola.

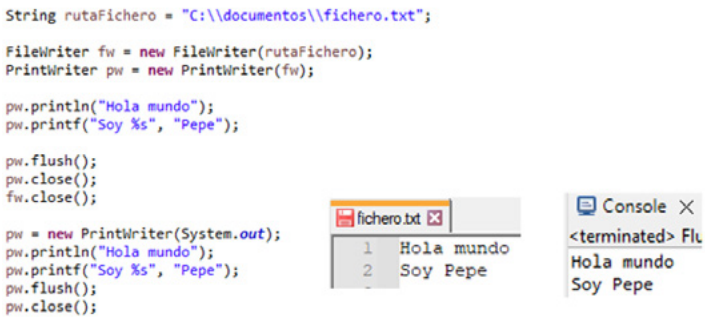

Imagen 11. Ejemplo de PrintWriter

12.2. Aplicaciones del almacenamiento de información en ficheros

Una vez visto los flujos de datos, usando diferentes interfaces como consola, fichero, red, etc., nos vamos a centrar en el almacenamiento en ficheros.

Los ficheros permiten almacenar datos y darles persistencia, pudiendo ser consultados en otro momento o por diferentes programas, sin tenerlo en memoria principal constantemente. Los ficheros también almacenan metainformación, como propietario del fichero, tamaño, permisos, fecha, etc.. necesarios para el sistema operativo y util para nuestro programa si tenemos que hacer uso de esta información.

Los tipos de ficheros que se verán serán de ficheros de texto, contiene texto legible y ficheros binarios con contenido legible por programas o sistemas.

12.2.1. Tipos de ficheros. Registro

Dependiendo del tipo de ficheros se tendrá que manejar de una forma u otra.

- **Ficheros de texto**
  Almacena cadena de caracteres. Su acceso habitual es de forma secuencial. Si se quiere acceder al final del fichero hay que recorrer todo el fichero.
- **Ficheros binarios**
  Almacena bytes. El acceso puede realizarse de forma secuencial o acceso directo.

El bloque de datos que permanecen juntos al ser leídos o escritos de le denomina registro.

12.2.2. Creación y eliminación de ficheros y directorios

Para la creación, eliminación y manejo de ficheros-directorios se usará la clase File con las siguientes métodos:

Operación	Descripción
File (String pathname)	Constructor, inicializa el objeto file con la ruta indicada.
boolean createNew-File()	Crea un fichero vacio en la ruta que contiene File
boolean mkdir()	Crea un directorio en la ruta que contiene File
boolean delete()	Elimina el fichero o directorio en la ruta que contiene File

Cuadro 1. Principales métodos y constructores de la clase File

```
String ruta = "C:\\documentos\\configuracion";

File rutaPrincipal = new File(ruta);
System.out.println("Creando directorio: "+ rutaPrincipal.mkdir());

File ficheroConfiguracion = new File(ruta, "config.txt");
System.out.println("Creando fichero: "+ ficheroConfiguracion.createNewFile());
```

```
<terminated> FicherosDirectorio
Creando directorio: true
Creando fichero: true
```

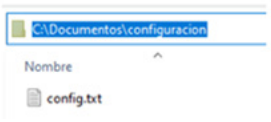

Imagen 12. Creación de directorios y ficheros

12.2.3. Apertura y cierre de ficheros. Modo de acceso

La apertura de un fichero consta en crear el fichero si no existe y abrirlo para su lectura o escritura. Dependiendo de la acción la apertura también determina desde donde empieza a escribir o leer.

El cierre consta de terminar de usar el fichero, liberando la memoria, punteros, cerrando el flujo de datos y otros referencias.

La apertura y cierre de ficheros dependerá del modo de acceso.

Modo de acceso secuencial

Para este tipo de acceso se podrá usar las clase FileReader, FileWriter, BufferedReader y BufferedWriter para ficheros tipo texto y FileInputStream, FileOutputStream, BufferedInputStream y BufferedOutputStream para ficheros tipo binario.

Para la apertura:

- Si no existe el fichero, se creará y se abrirá para la escritura.
- Si existe el fichero se abrirá para la escritura. Y si se indicó previamente el argumento append como true, la escritura empezará al final del fichero. De lo contrario el fichero será sobrescrito.

Para el cierre solo se invocara el método close().

Se pueden ver los ejemplos de apertura y cierre en el apartado 12.1.4.

Modo de acceso directo

El acceso directo solo se puede realizare para tipo de ficheros binarios, y se usará la clase RandomAccessFile.

Se podra crear objetos de RandomAccessFile a partir de la ruta o un objeto File y añadiremos el modo, si es para lectura (r) o lectura y escritura (rw).

```
String ruta = "C:\\documentos\\configuracion\\config.bin";
RandomAccessFile raf = new RandomAccessFile(ruta, "rw");

raf.writeChars("HOLA MUNDO");

raf.seek(0);
for (int i = 0; i < raf.length() / 2; i++)
    System.out.print("" + raf.readChar());

raf.close();
```

Imagen 13. Ejemplo de instanciación de RandomAccessFile

12.2.4. Escritura y lectura de información en ficheros

Una vez el fichero este abierto se puede proceder a su lectura o escritura, la forma de proceder dependerá del tipo de acceso y de información.

Modo de acceso secuencial

Para este tipo de acceso se podrá usar las clase FileReader, FileWriter, BufferedReader y BufferedWriter para ficheros tipo texto y FileInputStream, FileOutputStream, BufferedInputStream y BufferedOutputStream para ficheros tipo binario.

Se pueden ver los ejemplos de lectura y escritura en el apartado 12.1.4.

Modo de acceso directo

Para el acceso directo se usará la clase RandomAccessFile.

```
String ruta = "C:\\documentos\\configuracion\\config.bin";
RandomAccessFile raf = new RandomAccessFile(ruta, "rw");

raf.writeChars("HOLA MUNDO");

raf.seek(0);
for (int i = 0; i < raf.length() / 2; i++)
    System.out.print("" + raf.readChar());

raf.close();
```

Imagen 14. Ejemplo de RandomAccessFile

12.2.5. Almacenamiento de objetos en ficheros. Serialización

Hemos visto que podemos guardar en ficheros cualquier tipo de datos, en un tipo de fichero de texto podemos guardar cualquier información legible y en un tipo de fichero binario cualquier representación de tipos primitivos. También es posible guardar objetos de una clase transformandolo a un conjunto de bytes. Esta transformación se consigue con la **serialización**.

No todos los objetos se pueden serializar para poder aplicar la serialización se tiene que complir las siguientes condiciones:

1. La clase tiene que implementar la interfaz java.io.Serializable.
2. Todos los campos tienen que ser serializables. Los tipos primitivos ya son serializables y las clases compuestas por otras clases tienen que implementar serializable.
3. Para campos que no sean serializables hay que declararlos como transient. Los campos static no serán serializados.
4. Tiene que existir un campo estatico y privado llamado serialVersionUID. Este campo establece un número de versión de la clase serializable que se usa para deserializar.

Para escribir o leer se usarán las clases ObjectOutputStream y ObjectInputStream. Con el método writeObjet() se realizará la serialización y con readObjet() la deserialización.

```
String ruta = "C:\\documentos\\configuracion\\config.bin";
RandomAccessFile raf = new RandomAccessFile(ruta, "rw");

// ESCRITURA
raf.writeChars("HOLA MUNDO");

// LECTURA
raf.seek(0);
for (int i = 0; i < raf.length() / 2; i++)
    System.out.print("" + raf.readChar());
System.out.println();

// ESCRITURA
// Cambiamos MUNDO por A TODOS
raf.seek(8);
raf.writeChars(" A TODOS");

// LECTURA
raf.seek(0);
for (int i = 0; i < raf.length() / 2; i++) {
    System.out.print("" + raf.readChar());
}

raf.close();
```

```
<terminated> Fich
HOLA MUNDO
HOLA A TODOS
```

```
import java.io.*;
class MiObjeto implements Serializable
{
    int numero;
    String texto;
    public MiObjeto(int numero, String
texto) {
        this.numero = numero;
        this.texto = texto;
    }
}
public class SerializacionEjemplo {
    public static void main(String[] args) {
        // Crear un objeto
        MiObjeto objeto = new
MiObjeto(42, "Hola, mundo");
        // Serializar el objeto
        try {
            FileOutputStream fileOut = new
FileOutputStream("miobjeto.ser");
            ObjectOutputStream out = new
ObjectOutputStream(fileOut);
            out.writeObject(objeto);
            out.close();
            fileOut.close();
            System.out.println("El objeto ha
sido serializado y guardado en
miobjeto.ser");
        } catch (IOException e) {
            e.printStackTrace();
        }
        // Deserializar el objeto
        MiObjeto objetoDeserializado =
null;
        try {
            FileInputStream fileIn = new
FileInputStream("miobjeto.ser");
            ObjectInputStream in = new
ObjectInputStream(fileIn);
            objetoDeserializado =
(MiObjeto) in.readObject();
            in.close();
            fileIn.close();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException
e) {
            e.printStackTrace();
        }
        // Imprimir el objeto deserializado
        if (objetoDeserializado != null) {
            System.out.println("Objeto
deserializado - Número: " +
objetoDeserializado.numero + ", Texto:
" + objetoDeserializado.texto);
        }
    }
}
```

Imagen 15. Ejemplo de serialización

12.2.6. Utilización de los sistemas de ficheros

La clase File tienen más metodos de utilidad para operar con sistemas de ficheros. En el siguiente cuadro se pueden ver los métodos más comunes.

Operación	Descripción
boolean exists()	Comprueba si la ruta existe.
boolean isFile()	Comprueba si es un fichero.
boolean isDirectory()	Comprueba si es un fichero.
File[] listFiles()	Obtiene un listado de tipo File de los ficheros que hay en la ruta.
String getName()	Obtiene el nombre del fichero o directorio.
String getParent()	Obtiene el directorio padre.
String getPath()	Obtiene la ruta.
Boolean canRead()	Comprueba si se puede leer.
Boolean canWrite()	Comprueba si se puede escribir.
Boolean canExecute()	Comprueba si se puede ejecutar.

Imagen 16. Métodos para operar en un sistema de fichero de la clase File

www.universae.com