Flujos o Streams. Tipos

Flujos a ficheros de texto

Introducción

- Se usa la abstracción de flujo (stream) en java para tratar la comunicación entre una fuente y un destino.
- La fuente puede ser un fichero ubicado en cualquier dispositivo, un elemento periférico o cualquier otro programa.
- Dos tipos de flujos:
  - **Flujos de bytes**(8 bits): su uso está orientado a la lectura/escritura de datos binarios. Todas las clases de flujos de bytes descienden de las clases InputStream y OutputStream. Cada una de estas clases tienen varias subclases que controlan las diferencias entre los distintos dispositivos de entrada/salida que se pueden utilizar
  - **Flujos de caracteres** (16 bits): realizan operaciones de entrada/salida de caracteres. Estos flujos vienen gobernados por las clases Reader y Writer. La razón de ser de estas clases es la internacionalización; la antigua jerarquía de flujos de E/S solo soporta flujos de 8 bits y, por tanto, no manejaba caracteres Unicode de 16bits

Flujos de bytes (Byte stream)

- **InputStream** representa las clases que producen entradas de distintas fuentes (fichero, conexión, tubería, etc).
- Tipos:
  - ByteArrayInputStream: permite usar un espacio de almacenamiento intermedio de memoria. (Buffer)
  - StringBufferInputStream: convierte un String en un InputStream
  - FileInputStream: flujo de entrada hacia fichero; lo usaremos para leer información de un fichero
  - PipedInputStream: implementa el concepto de tubería
  - FilterInputStream: proporciona funcionalidades a otras clases InputStream
  - SequenceInputStream: convierte dos o más objetos InputStream en un InputStream único

- **OutputStream:** incluye las clases que deciden dónde irá la salida.
- Tipos:
  - ByteArrayOutputStream: crea un espacio intermedio en memoria (buffer)
  - FileOutputStream: flujo de salida hacia fichero; lo usaremos para enviar información a fichero
  - PipedOutputStream: cualquier información que se escriba aquí acaba en un PipedInputStream asociado. Implementan tuberías.
  - FilterOutputStream: proporciona funcionalidades a otras OutputStream

Flujos de caracteres (Character streams)

- Las clase **Reader** y **Writer** manejan flujos de caracteres Unicode.
- Hay ocasiones que conviene usar las clases que manejan flujos de byte con las clases que manejan caracteres. Para lograrlo, existen clases puente:
  - **InputStreamReader** que convierte un InputStream en un Reader
  - **OutputStreamWriter** que convierte un OutputStream en un Writer
- Las clases de flujos de caracteres más importantes son:
  - **FileReader** y **FileWriter:** para acceso a ficheros de caracteres
  - CharArrayReader y CharArrayWriter: leen y escriben un flujo de caracteres en un array de caracteres
  - BufferedReader y BufferedWriter: evitan el acceso directo al fichero utilizando un buffer intermedio entre la memoria y el stream

Formas de acceso a un fichero

- **Acceso secuencial:** los datos se leen y se escriben en orden (secuencialmente). Es parecido a una cinta de video antigua: si quieres acceder a un dato que está hacia la mitad de la cinta debes leer todos los anteriores. La escritura se hará a partir del último dato escrito; no es posible hacer inserciones entre datos ya escritos.
- **Acceso directo o aleatorio:** los datos se acceden directamente sin necesidad de leer todos los datos anteriores. Los datos están almacenados en registros de tamaño conocido, nos podemos mover de un registro a otro de forma aleatoria para leerlos o modificarlos.
- En Java el acceso secuencial se implementa con **FileInputStream / FileOutputStream** en el caso de datos binarios. Y **FileReader / FileWriter** para acceder a texto. Para el acceso aleatorio se utiliza **RandomAccessFile**

Flujos desde/hacia ficheros de texto

- Los ficheros de texto almacenan caracteres alfanuméricos en un formato estándar (ASCII, UNICODE, UTF8,...).
- Usaremos las clases **FileReader** y **FileWriter**. Debemos manejar siempre las excepciones con try-catch ya que:
  - Al leer un fichero se puede generar FileNotFoundException si el nombre del fichero no es válido o no existe
  - Al escribir se puede generar una IOException si no disponemos de permisos o si el disco está lleno

FileReader

- Los métodos que proporciona para lectura son:
  - int read() → lee un carácter y lo devuelve como entero
  - int read (char[ ] buf) → lee hasta *buf.length* caracteres. Los caracteres leídos se van almacenando en *buf*
  - int read (char [ ] buf, int desplazamiento, int n) → lee hasta *n* caracteres guardándolos en *buf* comenzando por *buf[desplazamiento]* y devuelve el número leído de caracteres.
- Estos métodos devuelven el número de caracteres leídos o -1 si se ha llegado al final del fichero.
- En un programa la secuencia de pasos será:
  - Primero: invocar a la clase File para acceder al fichero
  - Segundo: crear el flujo de entrada hacia el fichero con la clase FileReader
  - Tercero: operaciones de lectura
  - Cuarto: cerraremos el flujo mediante el método close().

```
import java.io.*;

public class LeerFicheroTexto {

        public static void main ( String [] args) throws IOException {

                File fichero = new File (“LeerFicheroTexto.java”);  // declaración fichero

                FileReader flu = new FileReader (fichero); // creamos flujo de entrada hacia el fichero

                int i;

                while ((i=flu.read())!=-1)     //Vamos leyendo carácter a carácter

                        System.out.println ((char) i); //hacemos cast a char del entero leído

                flu.close();

        }

}
```

Problemas Streams1

1. Prueba el código anterior
2. Modifica el código anterior para que el programa vaya leyendo caracteres de 20 en 20.
3. Modifica el código anterior para que se le puede pasar el nombre del fichero al programa.

FileWriter

- Los métodos que proporciona para escritura son:
  - void write (int c) → Escribe un carácter
  - void write (char [ ] buf) → Escribe un array de caracteres
  - void write (char [ ] buf, int desplazamiento, int n) → Escribe *n* caracteres comenzando por *buf[desplazamiento]*
  - void write (String str) → Escribe una cadena de caracteres
  - void append (char c) → Añade un carácter a un fichero
- Estos métodos pueden lanzar la excepción IOException
- Igual que antes declaramos el fichero mediante la clase File y a continuación se crea el flujo de salida hacia el fichero con la clase FileWriter

```
import java.io.*;
public class EscribirFicheroTexto {
        public static void main (String [] args) throws IOException {
                File fichero = new File(“FicheroTexto.txt”);
                FileWriter fic = new FileWriter (fichero);
                String cadena = “ Esto es una prueba con FileWriter”;
                char [] cad = cadena.toCharArray () ;
                for ( int i=0; i < cad.length ; i++)
                        fic.write (cad[i]); // se va escribiendo carácter a carácter
                fic.append (‘*’); // añadimos un asterisco al final
                fic.close ();   // cerramos fichero
        }
}
```

Problemas Streams2

1. Copia el ejemplo anterior y pruébalo
2. Modifica el ejemplo anterior para, en vez de escribir los caracteres uno a uno, escribir todo el array usando el método *write (char [] buf)*

NOTA: si queremos añadir caracteres al final de un fichero de texto podemos utilizar el siguiente constructor de FileWriter → FileWriter fic = new FileWriter (fichero , true)

3. Crea el siguiente array de String e inserta en el fichero las cadenas una a una usando el método *write (String str)*

String prov[] = {“Albacete”, “Avila”, “Badajoz”, “Caceres”, “Huelva”, “Jaen”, “Madrid”, “Segovia”, “Soria”, “Toledo”, “Valladolid”, “Zamora”}

BufferedReader / BufferedWriter

BufferedReader

- FileReader no contiene métodos que nos permitan leer líneas completas.
- **BufferedReader**, en cambio, sí dispone. En concreto, el método **readLine()** que lee una línea del fichero y la devuelve, o devuelve *null* si no hay nada que leer o llegamos al final del fichero. También dispone del método **read()** para leer un carácter.
- Para construir un BufferedReader necesitamos un objeto FileReader.

```
import java.io.*;
public class LeerFicheroTextoBuf {
        public static void main (String [] args) {
                try {
                        BufferedReader fichero = new BufferedReader ( new FileReader
                        (“LeerFicheroTextoBuf.java”)); // Atención: utilizamos otro constructor de FileReader. En
                        está ocasión no utilizamos un objeto File sino le pasamos el nombre del archivo
                        String linea;
                        while ((linea = fichero.readLine()) != null)  // leemos de línea en línea
                        System.out.println (linea);
                        fichero.close();
                }
                catch (FileNotFoundException fn) {
                        System.out.println (“No se encuentra el fichero”); }
                catch (IOException io) {
                        System.out.println (“Error de E/S”); }
        }
}
```

BufferedWriter

- Al igual que BufferedReader, BufferedWriter aporta funcionalidades de escritura a FileWriter. Dispone, por ejemplo, del método *newLine()* que inserta un salto de línea.
- Sigue el mismo modelo de construcción que BufferedReader; es decir, requiere un objeto FileWriter para instanciarse.

PrintWriter

- Otra clase que deriva de Writer. Posee los métodos *print (String)* y *println (String)* (idénticos a los de System.out) para escribir en un fichero.
- Para construir un PrintWriter necesitamos un FileWriter

```
PrintWriter fichero = new PrintWriter ( new FileWriter (NombreFichero));
```

Problemas Streams3

1. Escribe un programa en java que muestre por pantalla un fichero de texto que le pasamos como argumento (o utilizando scanner) utilizando la clase BufferedReader
2. Escribe un programa que, utilizando la clase BufferedWriter, escriba 10 filas de caracteres en un fichero de texto y después de escribir cada fila salta una línea con el método newLine()
3. Repite el ejercicio anterior pero ahora utilizando la clase PrintWriter