ACCESO-A-DATOS-TEMA-2.pdf

user_4383848

Acceso a datos

1º Desarrollo de Aplicaciones Multiplataforma

Estudios España

ACCESO A DATOS - TEMA 2
FLUJOS

1. DEFINICIÓN Y TIPOS DE FICHERO:

Flujo de datos → STREAM → Secuencia ordenada de información que posee un recurso de entrada y un recurso de salida. → **UNIDIRECCIONALES**.

STREAMS		
USABILIDAD	**BYTES (E/S)**	**CARACTERES (E/S)**
Ficheros	FileInputStream, FileOutputStream	FileReader, FileWriter
Arrays	ByteArrayInputStream, ByteArrayOutputStream	CharArrayReader, CharArrayWriter
Tuberías	PipedInputStream, PipedOutputStream	PipedReader, PipedWriter
Buffer	BufferedInputStream, BufferedOutputStream	BufferedReader, BufferedWriter
Análisis	PushbackInputStram, StreamTokenizer	PushbackReader, LineNumberReader
Información	DataInputStream, DataOutputStream, PrintStream	PrintWriter

2. CLASES DE FLUJOS BASADOS EN TUBERÍAS:

En el caso de Java, las tuberías proporcionan la capacidad de **ejecutar dos hilos (*threads*)** en la misma máquina virtual (JVM). Esto significa que las tuberías pueden ser tanto «orígenes», como «destinos» de datos.
En Java, las partes que se ejecutan deben pertenecer al mismo proceso, y deben ser hilos diferentes.

3. FLUJOS BASADOS EN ARRAYS:

Acceso basado en bytes → ByteArrayInputStream / ByteArrayOutputStream
Acceso basado en carácteres → CharrArrayReader / CharrArrayWriter

La clase **CharrArrayReader** nos permite leer el contenido de un *array* de caracteres (*char*) como un *stream* de caracteres. Esta clase nos será útil en casos en los que tengamos información en un *array* de caracteres y necesitemos pasarlo a algún componente que solo pueda leerse desde una clase *reader* o una de sus subclases.

4. CLASES DE ANÁLISIS PARA FLUJOS DE DATOS:

Las clases **PushBackInputStream y PushbackReader** están diseñadas para el análisis de datos previo de un InputStream. En algunas ocasiones se necesita leer algunos bytes con anticipación para saber qué se aproxima, para así poder interpretar el byte actual.

La diferencia básica entre las dos clases mencionadas anteriormente, es que básicamente PushbackInputStream está orientada a trabajar byte a byte y PushbackReader a leer directamente con caracteres.

5. CLASES DE ANÁLISIS PARA FLUJOS DE DATOS II:

Un **token** viene a estar relacionado con un "fragmento", "una ficha", "un trozo" ...
La clase StreamTokenizer tiene la capacidad de analizar el fichero por 'trozos' o 'fragmentos'. Dichos fragmentos más tarde tendremos que evaluarlos y comprobar si son palabras o números.
De hecho, **StreamTokenizer es capaz de reconocer identificadores: números, comillas, espacios, etc.,** lo cual nos puede ser de gran utilidad en aplicativos de análisis de ficheros según su tipología sea números o caracteres.

StreamTokenizer dispone de **algunos métodos estáticos** que nos ofrecen información de la tipología de los distintos token:

- TT_EOF: indica el final del fichero (*End Of File*)
- TT_EOL: indica el final de la línea (*End Of Line*)
- TT_WORD: indica que el *token* es de tipo palabra o un conjunto de letras.
- TT_NUMBER: indica que el *token* evaluado es un número o una asociación de ellos.

La lógica básica de este código: disponemos de un bucle de tipo *while* que se ejecutará mientras el siguiente *token* no suponga el final de nuestro fichero. A continuación, se observan una serie de condicionales:

6. CLASES DE ANÁLISIS PARA FLUJOS DE DATOS III:

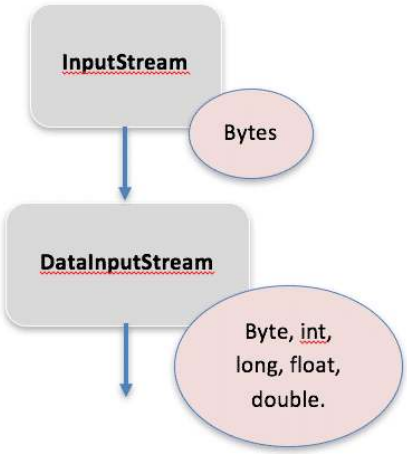
La clase **LineNumberReader,** que es básicamente un objeto de tipo BufferedReader que almacena y cuenta el número de líneas de caracteres leídas. Está orientada, además, a trabajar y analizar líneas completas. → Empieza con el contador a 0 la primera línea.

getLineNumber(): nos devolverá el número de la línea en la que estemos actualmente leyendo.

setLineNumber(): se le puede pasar como parámetro un entero, y se convertirá en la línea actual.

Otro de los métodos clave de esta clase es el **readLine()**, que nos devolverá un conjunto de caracteres al cual se le debe asignar una variable *string*. Evidentemente, dispone de su método **read()**, como ya hemos visto en clases anteriores.

7. CLASES PARA EL TRATAMIENTO DE INFORMACIÓN:



Suele usarse la clase **DataInputStream** para leer ficheros que previamente han sido escritos con **DataOutputStream**.

DataInputStream, al ser una subclase de **InputStream**, hereda los métodos que esta pone a su disposición. Por este motivo, si queremos leer *byte* a *byte* con el método **read()**, también lo tendremos disponible, al igual que la lectura con un *array* de *bytes*.

8. CLASES PARA EL TRATAMIENTO DE INFORMACIÓN II:

Para poder realizar una lectura de un fichero por medio de la clase DataInputStream, antes hemos tenido que realizar la escritura ordenada de dicho fichero y debemos ser conocedores del tipo de datos y la cantidad de ellos que ha sido insertada.

Para ello usamos DataOutputStream. El objeto es instanciado en su constructor con un FileOutputStream con la ruta del fichero que vamos a escribir.
Podemos usar diferentes métodos:

• **writeInt() :** con este método introduciremos enteros, por lo tanto solo aceptara entero como parámetro.
• **writeFloat() :** método que acepta como parámetro un objeto de tipo float. Igual que en el caso anterior solo podremos pasarle dicho tipo primitivo.
• **writeLong() :** en este caso solo podremos introducir un objeto de tipo long. En la última línea podemos observar cómo simplemente cerramos nuestro flujo y liberamos recursos.