Programación

Lectura y Escritura de la Información. Ficheros

Esta presentación ha sido elaborada a partir del material del módulo de Programación del IES El Majuelo (Gines)

ÍNDICE

1. Ficheros de datos

- Un **fichero** o archivo es la agrupación de una serie de datos, todos del mismo tipo, almacenados en una memoria secundaria.
- Cada dato almacenado en un fichero se denomina registro.
- Un registro está formado, a su vez, por uno o varios datos que pueden ser de diferentes tipos. Cada dato que contiene el registro se le conoce con el nombre de campo.
- El tamaño de un fichero es variable, ya que puede crecer o decrecer según se van insertando o eliminando registros.

2. Utilización de los sistemas de ficheros

- Gracias a la clase `File` podemos obtener información importante sobre el fichero con el que vamos a trabajar.

- Dicha clase tiene tres posibles constructores:
  - `File(String nomFicheDirec);`
  - `File(String camino, String nomFiche);`
  - `File(File camino, String nomFiche);`

Métodos de la clase `File`

- `boolean canRead()`: Informa si se puede leer la información que contiene.
- `boolean canWrite()`: Análogo para escribir.
- `boolean exists()`: Informa si el fichero o directorio existe.
- `boolean isFile()`: Informa si es un archivo.
- `long lastModified()`: Devuelve la fecha de la última modificación.
- Específicos para ficheros:
  - `delete()`: Borra el archivo.
  - `long length()`: Devuelve el tamaño en bytes.
  - `boolean renameTo(File)`: Renombra el archivo.
- Específicos para directorios:
  - `boolean mkdir()`: Crea el directorio asociado a la instancia File que lo invoca.
  - `String [] list()`: Devuelve un listado de los archivos que se encuentran en el directorio asociado a la instancia File que lo invoca.

Localización de los archivos

- Si se indica el nombre del fichero sin su camino, lo buscará en el directorio actual (carpeta del proyecto).

- Si se indica el camino y el fichero:
  - Si no empieza por “/” se tratará de una ruta relativa.
  - Si empieza por “/” se tratará de una ruta absoluta.

3. Apertura y cierre

- JAVA tiene varias clases que permiten el manejo de ficheros dentro del paquete **`java.io`**. Las más importantes son:
  - **`FileInputStream`** y **`FileOutputStream`**. Dichas clases permiten leer y escribir *bytes* en los archivos, respectivamente.
  - Si vamos a trabajar con archivos de texto es preferible trabajar con las clases **`FileReader`** y **`FileWriter`**. Dichas clases permiten leer y escribir caracteres en los archivos, respectivamente.
- Para abrir un archivo basta con instanciar un objeto de una de esas clases.
- Para cerrar un archivo se emplea el método **`close()`**.
- Al alcanzar el final del archivo, se lanza un error de tipo **`EOFExcepcion`**.

4. Escritura y lectura de información
4.1. FileInputStream

- Cuando instanciamos un objeto con este tipo de clase, lo que hace el programa es abrir el fichero que se envía como argumento para lectura. Una vez abierto, se podrá leer la información que contiene de forma secuencial *byte* a *byte*.
- Constructores:
  - Argumento: nombre del fichero que se quiere abrir.
  - Argumento: objeto `File` que representa el fichero con el que queremos trabajar.
- Lanza la excepción **`FileNotFoundException`** si el fichero no existe.

4. Escritura y lectura de información
4.1. FileInputStream

Principales métodos:

- **`int read()`**:
  - Devuelve el código ASCII del siguiente *byte* que hay después de donde está situado el puntero del fichero.
  - Dicho puntero se va moviendo secuencialmente por el fichero según vamos leyendo los *bytes*.
  - Devuelve -1 si no hay ningún *byte* más que leer.
- **`int read(byte cadByte[])`**:
  - Lee hasta **`cadByte.length bytes`**, guardándolos en la tabla que se envía como parámetro.
  - Devuelve -1 si no hay ningún *byte* más que leer.

4. Escritura y lectura de información
4.2. FileOutputStream

- Cuando instanciamos un objeto con este tipo de clase, lo que hace el programa es abrir el fichero que se envía como argumento para escritura. Una vez abierto, se podrá guardar información *byte* a *byte*. Si el fichero no existiera, lo crearía.
- Constructores:
  - Argumento: nombre del fichero que se quiere abrir.
  - Argumento: objeto `File` que representa el fichero con el que queremos trabajar.
- Si el fichero ya existe se puede:
  - empezar desde el principio, borrando la información que contiene.

```
FileOutputStream fich = new FileOutputStream(“hola.txt”);
```

  - añadir información nueva al final de lo que ya contiene.

```
FileOutputStream fich = new FileOutputStream(“hola.txt”,
  true);
```

4. Escritura y lectura de información
4.2. FileOutputStream

Principales métodos:

- **`int write(int byte)`**:
  - Escribe el byte que recibe como argumento en el fichero.
- **`int write(byte cadByte[])`**:
  - Escribe todos los bytes que contiene la tabla `cadByte` en el fichero.

DataInputStream y *DataOutputStream*

- Al acceder a un fichero haciendo uso de estas clases, podremos usar más métodos para la entrada y salida de datos que los que contienen las clases anteriores:
  - `short readShort(), int readInt(), float readFloat(), String readUTF(), …`
  - `void writeShort(short dato), …`
- Para abrir un fichero con estas clases:

```
FileOutputStream fichero = new FileOutputStream(“prueba.txt”);
DataOutputStream lectTipos = new DataOutputStream(fichero);
```

(ídem para `Input`)

4. Escritura y lectura de información
4.3. FileReader

- Cuando instanciamos un objeto con este tipo de clase, lo que hace el programa es abrir el fichero que se envía como argumento para lectura. Una vez abierto, se podrá leer la información que contiene de forma secuencial carácter a carácter.
- Constructores:
  - Argumento: nombre del fichero que se quiere abrir.
  - Argumento: objeto `File` que representa el fichero con el que queremos trabajar.
- Lanza la excepción **`FileNotFoundException`** si el fichero no existe.

4. Escritura y lectura de información
4.4. FileWriter

- Cuando instanciamos un objeto con este tipo de clase, lo que hace el programa es abrir el fichero que se envía como argumento para escritura. Una vez abierto, se podrá guardar información carácter a carácter. Si el fichero no existiera, lo crearía.
- Constructores:
  - Argumento: nombre del fichero que se quiere abrir.
  - Argumento: objeto `File` que representa el fichero con el que queremos trabajar.
- Si el fichero ya existe se puede:
  - empezar desde el principio, borrando la información que contiene.

```
FileWriter fich = new FileWriter(“hola.txt”);
```

  - añadir información nueva al final de lo que ya contiene.

```
FileWriter fich = new FileWriter(“hola.txt”, true);
```

4. Escritura y lectura de información

- Con las clases anteriores cada carácter o byte que lee o escribe del/en el fichero se procesa de uno en uno, lo cual hace que dicha tarea sea lenta.

- Gracias al uso de clases como **`BufferedReader`**, **`BufferedInputStream`**, **`BufferedWriter`** y **`BufferedOutputStream`** el proceso es más rápido ya que se utiliza un buffer intermedio y el procesamiento no es por unidades individuales.

5. Modos de acceso

- Lectura secuencial: Implica tener que leer el fichero desde el principio e ir pasando por toda la información que contiene para poder llegar a un determinado dato.

- Lectura aleatoria: Permite acceder a los datos directamente, indicando para ello la posición donde se encuentran.
  - Para poder realizarla, el fichero tiene que estar estructurado de forma que permita dicho acceso.
  - Clase **`RandomAccessFile`**.

RandomAccessFile

- Con el nombre del fichero:

```
RandomAccessFile miRAFile = new RandomAccessFile
                    (String noFich, String opePer);
```

- Con un objeto `File`:

```
RandomAccessFile miRAFile = new RandomAccessFile
                        (File fich, String opePer);
```

El argumento `opePer` determinará si se va a poder leer el contenido del fichero (*r*) o leer y escribir (*rw*).

Métodos *RandomAccessFile*

- Implementa las interfaces **`DataInput`** y **`DataOutput`**, de manera que puede usar los métodos **`write()`** y **`read()`**.
- Además, tiene otros métodos:
  - **`long getFilePointer()`**: Indica dónde está situado el puntero del fichero.
  - **`void seek(long pos)`**: El argumento que recibe determina lo que se tiene que desplazar el puntero del fichero, partiendo desde su inicio.
  - **`long length()`**: Indica la posición donde acaba el fichero.
  - **`int skipbytes(int d)`**: Desplaza el puntero del fichero (desde su posición actual) tantos *bytes* como el valor que recibe como argumento.

6. Almacenamiento de objetos en ficheros

- Persistencia: Almacenamiento de toda la información que contiene un objeto, manteniendo toda su estructura, dentro de una memoria secundaria.
- Puede hacerse en ficheros y en una base de datos.
- Dos tipos de representaciones:
  - Textual.
  - Binaria.

Representación textual

- Consiste en guardar y leer los objetos en un fichero XML. Para ello, se usa la clase **`XMLEncoder`** que permite guardar los objetos en dichos ficheros. Con ello, se permite la posibilidad de poder guardar los objetos como si fuera texto.
- Para leer la información de los objetos guardados en ese fichero, se usa la clase **`XMLDecoder`**.
- Todas estas clases están en el paquete **`java.beans`**.

Representación binaria

- Consiste en guardar el estado de un objeto usando un flujo de *bytes* para poder guardarlo en un fichero. A esto se le llama **serialización**.
- Se usa la clase **`ObjectOutputStream`** para guardar objetos en un fichero usando el método **`writeObject()`**.
- Se usa la clase **`ObjectInputStream`** para leer dichos objetos usando el método **`readObject()`**.
- Todos los objetos que se guarden tienen que implementar la interfaz **`Serializable`**.