PRÁCTICA 3

3.1. Objetivo

El objetivo de esta práctica es la familiarización con alguna de las clases que ofrece Java para trabajar de una manera más sencilla con los datos primitivos de Java, más allá de los bytes que trabajamos en la práctica anterior.

3.2. Flujos de salida (escritura): escritura de tipos de datos primitivos de Java

Hasta ahora nos hemos dedicado simplemente a escribir bytes en un flujo de datos. Aunque con únicamente estas técnicas podríamos conseguir leer y escribir cualquier cosa en un flujo (ligado a un fichero, a una conexión TCP, etc.), esta forma de trabajar es relativamente pesada ya que cada vez que quisiéramos escribir, por ejemplo, un entero de 32 bits en un archivo, tendríamos que dividir los 32 bits en cuatro paquetes de 8 bits cada uno e ir transmitiéndolos a través del flujo de datos. Para leer ese dato, el proceso sería el inverso: leer cuatro bytes del flujo y combinarlos para obtener el número de 32 bits.

Como veremos, el paquete java.io nos proporciona varias herramientas para facilitarnos este trabajo.

La clase *DataOutputStream*, heredera indirecta de *OutputStream*, añade a ésta última la posibilidad de escribir datos "complejos" en un flujo de salida. Cuando hablamos de datos "complejos", en realidad nos referimos a tipo de datos primitivos pero no restringidos únicamente a bytes y a matrices de bytes, como en el caso de *OutputStream*.

Mediante la clase *DataOutputStream* podemos escribir datos de tipo *int*, *float*, *double*, *char*, etc. Incluso podemos escribir algunos objetos, como datos de tipo *String*, en una gran cantidad de formatos.

La forma general de trabajar con objetos de tipo *DataOutputStream* será la siguiente: obtenemos un objeto *OutputStream* (cuyo origen puede ser cualquiera: un archivo, un socket, una matriz en memoria, la salida standard, etc) y lo "envolvemos" en un objeto *DataOutputStream*, de forma que podamos usar la interfaz que nos proporciona este último. Para crear este *DataOutputStream*, le pasamos como parámetro el *OutputStream* a su constructor.

Cada vez que escribamos un dato "complejo" en un objeto *DataOutputStream*, éste lo traducirá a bytes individuales, y los escribirá en el *OutputStream* subyacente, sin que nosotros tengamos que preocuparnos de la forma en que lo hace.

Ejercicio 1

Ejercicio 1.1

Crea un objeto *DataOutputStream* a partir del objeto *System.out* (“envolviendo” al objeto *System.out*). Para hacerlo:

```
DataOutputStream dos= new DataOutputStream(System.out);
```

Una vez creado, inicializa una variable de tipo entero, otra de tipo *float* y otra de tipo *double* y escribe a través del flujo de salida formateado (el objeto de tipo *DataOutputStream*) los datos. Consulta la documentación de la clase *DataOutputStream* (http://docs.oracle.com/javase/7/docs/api/) para ver qué métodos podemos utilizar para cada uno de los tipos de datos de las variables creadas.

¿Qué aparece por pantalla?

¿A qué es debido?

Ejercicio 1.2

Modifica ahora el programa anterior para escribir los datos en un fichero. Para ello tendréis que crear un objeto de tipo *OutputStream* asociado a un fichero de la forma siguiente:

```
OutputStream os= new FileOutputStream(“d:\\temp\\datos.dat”);
DataOutputStream dos= new DataOutputStream(os);
```

o

```
FileOutputStream fileOut=new FileOutputStream("d:\\temp\\datos.dat");
DataOutputStream salida=new DataOutputStream(fileOut);
```

Ejercicio 1.3

Modifica ahora el programa del ejercicio 1.2 de forma que lea correctamente los números enteros almacenados en el fichero.

3.3. Clase *File*: interacción con el sistema de ficheros en Java

En la práctica 2 vimos uno de los posibles constructores de la clase *FileInputStream*: el que recibe como parámetro un nombre de archivo para leer. Existe otras dos variantes: una que recibe un objeto de tipo *File* y otra que recibe un objeto de tipo *FileDescriptor*. Veamos en qué consiste la clase *File*.

La clase *File* sirve para encapsular la interacción de nuestros programas con el sistema de archivos. Mediante la clase *File* no nos limitamos a leer el contenido de un archivo, como ocurría con la clase *FileInputStream*, sinó que podemos obtener información adicional, como el tamaño del archivo, su tipo, su fecha de creación, los permisos de acceso que tenemos sobre él, etc.

Además, la clase *File* es la única forma que tenemos de trabajar con directorios (crearlos, ver los archivos que contienen, cambiar el nombre o borrar los archivos, etc.)

La forma más sencilla de crear un objeto *File* es:

```
File miArchivo = new File("c:\\temp\\archivo.txt");
```

Para poder leer del fichero al que hace referencia el objeto *File* podemos pasar como parámetro el objeto *File* al contructor de la clase *FileInputStream*:

```
FileInputStream fos = new FileInputStream(miArchivo);
```

De la misma forma, para poder escribir en el fichero al que hace referencia el objeto *File* podemos pasarlo como parámetro al constructor de la clase *FileOutputStream*:

```
FileOutputStream fos = new FileOutputStream(miArchivo);
```

Para saber si un objeto *File* se refiere a un archivo existente podemos usar el método *exists()*:

Ejercicio 2

Ejercicio 2.1

Realiza un programa que cree un objeto *File* que haga referencia al fichero *d:\tmp\archivo.txt* e indique por pantalla si el fichero existe o si no existe.

```
File miArchivo = new File("d:\\temp\\archivo.txt");
...
```

Ejercicio 2.2

Añade al programa anterior el código necesario para averiguar y sacar por pantalla:

- el nombre del fichero al que hace referencia el objeto *File* creado
- el nombre del directorio en el que está el objeto *File* creado

Para ello utilizad los métodos *getName();* y *getParent();*

Ejercicio 2.3

Escribe un nuevo programa que saque por pantalla el listado de todos los ficheros existentes en el directorio *d:\tmp*. Para ello deberéis crear un array de strings donde almacenar los nombres de los ficheros y utilizar el método *list();*