BOLETÍN 01.01: FLUJOS Y ARCHIVOS

EJERCICIO 1. COPIA DE ARCHIVOS CON BUFFER

1

Se realice un programa para **copiar archivos**. El programa debe recoger el nombre del archivo origen y destino. Se existe debe solicitar confirmación sobrescribir. Úsese I/O con *buffer* y métodos estáticos (tenga en cuenta que los archivos pueden ser binarios).

a) Para la lectura desde teclado puede emplearse la clase Scanner.

b) Realiza el mismo ejercicio, pero empleando entradas desde ventana con *JFileChooser* y mensajes de error en *JOptionPane*, si los hay.

c) Realiza un programa que lea con un *JOptionPane* pida una URL y para posteriormente abrir un *JFileChooser* para guardarlo en el disco local.
Ayuda: para abrir un flujo de entrada a una URL puede hacerse con el método openStream() de URL. Ten en cuenta que puede lanzar excepciones.
`InputStream in = new URL(FILE_URL).openStream();`

d) Mejora el aparado a) para que la lectura de los datos lo haga en bloques (buffer) y no byte a byte.

EJERCICIO 2. ESTADÍSTICAS DE UN ARCHIVO

Realice un programa que recoja el nombre de un fichero y muestre **una estadística de la ruta, número de líneas, número de espacios, número de letras, fecha última modificación, longitud del fichero, ...**

Defina una clase ***EstatisticaFile*** con atributos: ***letras***, ***linhas***, ***espacios***, ***archivo*** (tipo ***File***).

```
private File arquivo;
private int linhas;
private int letras;
private int espazos;
```

Métodos para obtener cada uno de los atributos, ***existe()***, ***ultimaModificacion()***, ***getRuta()***.

El **constructor recoge el nombre del archivo**.

EJERCICIO 3. EDITOR DE TEXTO

Haz un programa que recoja el nombre de un fichero y muestre su contenido si existe o cree un nuevo en el que puedas escribir si no existe. Ejemplo: *java Editor proba.txt*

Para tal fin, además del programa, *Editor.java*, crea la clase ***Documento con*** las siguientes características:

1. Propiedades: **arquivo** (de tipo *File*)
2. Constructores: **recoge el nombre del archivo y** crea el objeto *archivo*. Otro que recoja un Objeto de tipo File.
3. Métodos:
    1. ***exists()***: devuelve verdadero cuando el fichero no es nulo y existe.
    2. ***readFile()***: devuelve una cadena con el contenido del archivo, si existe, obviamente. Emplea StringBuilder.
    3. ***readFileNIO()***: igual al anterior, pero empleado ***Path*** y el método *readString* de ***Files***.
    4. ***writeFromString(...)***: recoge una cadena y la escribe en fichero, al final, empleando *BufferedWriter*.
    5. ***writeFromStringPrintWriter(…)***: recoge una cadena y la escribe al fina, empleando PrintWriter.
    6. ***writeFromInputStream()***: rue recoge un flujo de tipo *InputStream* (para, por ejemplo, *System.in*) y escribe lo recogido por el flujo en el fichero.
    7. ***writeFromKeyword()***: escribe en el archivo lo que se escriba en el teclado.
    8. ***getFile()***: devuelve el objeto archivo.
    9. ***toString()***: devuelve la ruta absoluta/canónica al archivo.

AppEditor.java recoge el nombre por línea de órdenes. Si existe, muestra el contenido (llama al método readFile()) si no existe pide que introduzcas por teclado. Para acabar de introducir datos debe escribir una línea que sólo contiene un ".".

EJERCICIO 4. LECTURA DE TECLADO

Se realice una clase de utilidad ***Teclado*** con métodos y atributos estáticos para leer desde teclado, que tenga un atributo estático privado ***LECTOR*** de tipo ***BufferedReader*** (lector de caracteres con *buffer* que permite leer línea la línea). La clase debe ter los siguientes métodos: *lerString*, *lerChar*, *lerInt*, *lerLong*, *lerBoolean*, *lerFloat*, *lerDouble*, *lerByte*, *lerShort*, para cada tipo de dato básico.
Haz un pequeño programa que haga uso esta clase.

*Ayuda: empleade el atributo estático **System.in** (de tipo java.io.**InputStream**), así como la clase correspondiente que permita pasar un flujo de tipo Byte a un flujo de tipo Carácter.*

Como sabéis, Java ya incorpora clases para facilitar la lectura desde teclado: java.io.Console (java 1.6 y sup.) e java.util.Scanner (java 1.5 e sup.)

EJERCICIO 5. GESTIÓN DE EQUIPOS DE FÚTBOL (PROPUESTA)

3

Haga un programa de gestión de la clasificación de la liga de fútbol. Declare una clase ***Equipo*** con los atributos mínimos necesarios: ***nome***, ***ganhados***, ***perdidos***, ***empatados***, ***golesFavor***, ***golesContra***. Para poder ordenar os equipos debe implantar a interface ***Comparable***, y para poder guardarse con el método ***writeObject*** de ***ObjectOutputStream*** debe implantar ***Serializable***. Sobrescribe el método ***equals*** para que dos Equipos sean iguales si tienen el mismo nombre (implanta hashCode())

Los equipos deben guardarse en un fichero "***clasificacion.dat***".

El programa debe tener un menú con las siguientes opciones: **cargar equipos, añadir equipo, guardar equipos, mostrar clasificación, modificar equipo**.

Una vez cargados emplee un objeto de tipo ***TreeSet*** *para que los ordene correctamente*

EJERCICIO 5 (II). XESTIÓN DE EQUIPOS DE BALONCESTO

Haga un programa para la **gestión y clasificación de la liga de baloncesto**. La clasificación de los equipos se guarda en un archivo llamado ***clasificacion.dat***.

a) Declare una clase ***Equipo*** con los atributos mínimos necesarios: ***nombre***, ***victorias***, ***derrotas***, ***puntos a favor***, ***puntos en contra***. Puedes añadir los atributos que te interesen, como ciudad, etc. Tienes libertad para hacerlo, pues, además, te puede servir como práctica.

   Tenga en cuenta que los atributos ***puntos***, ***partidos jugados*** y ***diferencia de puntos*** son atributos **derivados** que se calculan a partir de los partidos ganados, perdidos, puntos a favor y puntos en contra.

   Cree los **métodos** que considere oportunos, pero tome decisiones sobre los métodos get/set necesarios. Así, cree un método que devuelva los puntos, ***getPuntos***, un método ***getPartidosJugados*** que devuelva el número de partidos jugados y un método ***getDiferenciaDePuntos***, que devuelva la diferencia de

puntos. Obviamente, por ser atributos/propiedades derivados/as, no tiene sentido métodos de tipo "set" para ellos.

Debe tener, al menos, un constructor para la clase equipo que recoja el nombre y otro que recoja todas las propiedades. No debe existir un constructor por defecto.

Para poder ordenar los equipos debe **implantar la interface *Comparable<Equipo>***, del mismo modo que se ha hecho con los Naipes. Piense que debe ordenar por puntos y, a igualdad de puntos, por diferencia de puntos encestados. Además, para poder guardar los objetos (***writeObject*** de ***ObjectOutputStream***) y/o recuperarlos (***readObject*** de ***ObjectInputStream***) **debe implantar la interface *Serializable***. Lo mismo con la clase siguiente, ***Clasificacion***, que debe implementar la interface Serializable.

Sobrescribe el método ***equals*** para que se considere que **dos Equipos son iguales si tienen el mismo nombre** (sin distinguir mayúsculas de minúsculas).

b) Declare una clase ***Clasificacion***, con un atributo ***equipos*** de tipo array de 18 equipos (por defecto), aunque debe existir un constructor que permita crear una clasificación con los equipos que se desee.

Defina los métodos para añadir equipos a la clasificación, ***addEquipo***, así como los métodos para eliminar equipo, ***removeEquipo***, y sobrescriba el método ***toString*** que devuelva la cadena de la clasificación.

Crea los métodos estáticos: ***loadClasificacion***, que cargue la clasificación del archivo y la devuelva, y el método ***saveClasificacion***, que guarde la clasificación en el archivo.

*Una vez cargados se podría emplear un objeto de tipo **TreeSet** para que ordene correctamente la clasificación (lo veremos en unidades posteriores)*

c) El programa debe tener un menú con las siguientes opciones:
   a. **añadir equipo** (*pide el nombre del equipo y los valores de los atributos no derivados, añadiendo el equipo a la clasificación*)
   b. **mostrar clasificación** (*muestra la clasificación ordenada de los equipos que están cargados en memoria*)
   c. **guardar clasificación** (que guarda la clasificación en el archivo ***clasificacion.dat***)
   d. **cargar clasificación** (que carga la clasificación del archivo ***clasificacion.dat***)
   e. **salir** (sale del programa, debiendo preguntar antes).

Utilice la clase Scanner para leer de teclado.

Java NIO. Uso de Path, Paths y Files

A partir de Java 7 se **añade una interface denominada *Path*** que forma parte de Java NIO (No-blocking IO). La interface ***Path*** se localiza en el paquete **java.nio.file** (***java.nio.file.Path***), introducido en Java 7. NIO se introdujo en Jva 1.4 y NIO.2 en Java 7.

Una instancia de la interface *Path* representa una **ruta (*path*) en el sistema de archivos**. Un path puede apuntar tanto a un **archivo como a un directorio**, tanto de manera **absoluta como** de manera **relativa**. Proporciona capacidades para tareas con el sistema de ficheros (trabajo con enlaces simbólicos, carga más rápida en buffer…)

java.nio.file.Path es una interface **similar a la clase *java.io.File***, pero con algunas diferencias. En muchos casos se puede sustituir la clase File por la interface Path.

1. Creación de una instancia de Path

Para crear una instancia de ***Path*** hay que usar un método estático de la clase ***Paths*** (***java.nio.file.Paths***) llamado *Paths.get(….)*:

```
import java.nio.file.Path;
import java.nio.file.Paths;

public class EjemploPath {

    public static void main(String[] args) {

        Path path = Paths.get("c:\\datos\\miarchivo.txt");

    }
}
```

Nota: el método *get*, en términos de patrones de diseño, se dice que es un *factory method* para crear instancias de Path. Fíjate que es necesario escapar el carácter \, pues es el carácter de escape de java (\n, \t, \\…)

2. Creación de un Path relativo

Un Path relativo es un path que referenciado desde una ruta base a un archivo o directorio. Se puede crea una ruta relativa usando el método:

```
Paths.get(basePath, relativePath)
```

Por ejemplo:

```
Path proyectos = Paths.get("d:\\datos", "proyectos");

Path     archivo                          =     Paths.get("d:\\datos",
"proyectos\\mitarea\\miarchivo.txt");
```


En el primer caso apunta a un directorio. En el segundo apunta a un archivo.
Un modo muy útil para la **ruta a un archivo en la que se ejecuta el programa** es:

***Path** p **= Paths.**get**(System.**getProperty("**user.dir**"), archivo);*

Paquete ***java.nio.file***:

https://docs.oracle.com/javase/8/docs/api/java/nio/file/package-summary.html

EJERCICIO 6. PROPUESTA AVANZADA DE EXERCICIO (FUERADE CATEGORÍA ☺)

Modificación de un archivo BMP.

1. Haga un programa que lea la cabecera de un archivo BMP sin compresión de 24 bits. Emplee un flujo de tipo *DataInputStream*. Defina una clase ***Cabecera*** que recoja el nombre del archivo y tenga los atributos necesarios para guardar la información del mismo.

/
** 2 signature, must be **4D42** hex*
** 4 size of BMP file in bytes (unreliable)*
** 2 reserved, must be zero*
** 2 reserved, must be zero*
** 4 offset to start of image data in bytes*
** 4 size of BITMAPINFOHEADER structure, must be 40*
** 4 image width in pixels*
** 4 image height in pixels*
** 2 number of planes in the image, must be 1*
** 2 number of bits per pixel (1, 4, 8, or 24)*
** 4 compression type (0=none, 1=RLE-8, 2=RLE-4)*
** 4 size of image data in bytes (including padding)*
** 4 horizontal resolution in pixels per meter (unreliable)*
** 4 vertical resolution in pixels per meter (unreliable)*
** 4 number of colors in image, or zero*

** 4 number of important colors, or zero*
**/*

1. Diseña e implanta de un programa que lea la cabecera de un *BMP* y permita invertir la imagen, pasarla a escala de grises, añadir ruido, aclarar y oscurecer. La imagen está a continuación de la cabecera. Para pasar la escala de grises hay que establecer los 3 colores del píxel al mismo nivel con la media de los colores.