Ejercicios Tema 12. Gestión de archivos

*Todos los ejercicios de este boletín deben estar dentro de un **package** llamado **tema12** (Siguiendo las reglas que hemos visto en clase, ej: com.germangascon.tema12).*

Hasta ahora, los ejercicios que hemos realizado a lo largo del curso no suponían ningún peligro para los datos de nuestro equipo en caso de equivocarnos, pero a lo largo de este tema, vamos a trabajar con archivos guardados físicamente en tu disco duro. Por este motivo, para la realización de los siguientes ejercicios debes crearte una carpeta específica donde copiaremos algunos directorios y archivos de tu disco duro (los que tu quieras) a modo de ejemplo. Todas las rutas deben hacer referencia, única y exclusivamente, a archivos ubicados dentro de dicha carpeta.

Un ejemplo de ruta absoluta para la carpeta de los ejemplos podría ser:
/home/usuario/tema12/ficheros

Para evitar problemas no utilices espacios en blanco en la ruta absoluta de la carpeta.

La carpeta debe contener al menos: varios directorios y subdirectorios, varios archivos de texto plano (*.txt) y varios archivos binarios (*.exe).

1. Utilizando la clase File comprueba que la carpeta que has creado para este boletín de ejercicios existe y que se trata de un directorio.

2. Muestra los archivos y directorios que están en la carpeta del ejercicio. Solo los de primer nivel (no recursivo).

3. Obtén las siguientes propiedades de la carpeta del ejercicio:
   a) El nombre de la carpeta
   b) La ruta absoluta
   c) Si se puede leer
   d) Si se puede escribir

4. Obtén las siguientes propiedades de un archivo que esté contenido dentro de la carpeta del ejercicio (el que tú quieras):
   a) El nombre del archivo
   b) La ruta absoluta

c) Si se trata de un archivo oculto

d) Si se puede leer

e) Si se puede escribir

f) La fecha de la última modificación (ayúdate de SimpleDateFormat). Luego cambia la fecha de modificación a la hora actual y vuélvela a comprobar.

g) El tamaño en bytes, en KB y en MB.

5. Crea los siguientes métodos en una clase llamada GestionArchivos.

a) boolean crearArchivo(String directorio, String archivo): creará un archivo con el nombre archivo.

b) void listarDirectorio(String directorio): visualizará el contenido del directorio mostrando el tipo (fichero "f" , directorio "d"), el tamaño y los permisos de lectura y escritura. Ejemplo:

hola.txt f 10 bytes rw

c) void verInfo(String directorio, String archivo): visualizará el nombre, la ruta absoluta, si se puede escribir, si se puede leer, el tamaño, si es un directorio y si es un archivo.

Los parámetros que necesitan los métodos serán solicitados al usuario desde el programa principal.

6. Añade un método a la clase anterior que permita leer y mostrar el contenido de un archivo de texto.

7. Añade un método a la clase anterior que permita ver el contenido de un archivo binario en hexadecimal agrupados en bytes (de dos en dos). Ejemplo de salida:

6d 20 1b 5b 33 37 6d 5b 32 31 3a

3a 5d 0d 0a 1b 5b 31 6d 1b 5b 33

5b 30 30 6d 1b 5b 4b 1b 5b 3f 31

…...

Recuerda que la opción de formato "%X" podemos representar un valor numérico en hexadecimal.

8. Crea un programa que permita leer y mostrar el contenido de un archivo pasado como parámetro al llamar al programa. Para ello debes utilizar el parámetro String[] args de la función main.

9. Haz un programa con dos métodos que invocaremos desde el principal:

a) Un método (insertarAlumnos) para insertar los todos elementos de un array en un archivo. Será un array con el nombre de los alumnos de clase. Se debe insertar cada nombre una línea diferente (utilizar la clase BufferedWriter).

b) Otro método (eliminarAlumno) que solicite al usuario el nombre del alumno que se quiere eliminar del archivo y lo elimine.

10. Haz un método llamado concat que reciba como parámetro 2 archivos y cree un tercer fichero, cuyo contenido, se obtendrá uniendo la información de los 2 archivos indicados. Primero irá el contenido del archivo recibido como primer parámetro y después el contenido del archivo recibido como segundo parámetro.

11. Haz un método llamado concatLines que reciba como parámetro 2 archivos y cree un tercer archivo, el contenido, se obtendrá uniendo la información de los 2 archivos. Cada línea del archivo creado estará formado por la unión de la misma línea de los dos archivos leídos.

12. Haz un método que reciba como parámetro un fichero con 20 DNI aleatorios (sin letra) y:
    a) compruebe que todos los DNI tienen una longitud de 8 dígitos, sino es así, deberá rellenar con 0's por la izquierda.
    b) calcule la letra correspondiente a cada DNI y la añada por la derecha.
    c) Guarde el resultado en otro archivo cuyo nombre sea el resultado de concatenar el nombre del archivo original más "_conLetras" en la parte del nombre.

13. Crea un programa que calcule números primos. Cada número primo encontrado será añadido a un archivo. Al iniciar el programa comprobará si existe el archivo, en caso afirmativo, continuará por el último número primo calculado en la sesión anterior, en caso negativo, comenzará por el principio.

14. Crea un método que reciba como parámetros dos archivos, compare su contenido e indique si son iguales o no.

15. Implementa una utilidad para comparar el contenido de dos directorios. Sólo tendrá que hacer la comparación con los archivos, no con los posibles subdirectorios que pudieran tener los directorios a comparar. Implementa la utilidad en una clase llamada DiffFolder.
    La clase tendrá los siguientes métodos con la siguiente sintaxis:

    a) *void setFolders(File folder1, File folder2)*: Este método permite asignar los objetos de la comparación. Es necesario que sean dos carpetas que existan en el sistema. En caso de que alguna de ellas no fuera un directorio, se lanzará una excepción del tipo

GestionArchivosNotFolderException. Una vez asignados, serán los directorios empleados por el método compare.

b) File getFolder1(). Devuelve la carpeta (como instancia de File) asignado en primer lugar en el método *setFolders*().

c) File getFolder2(). Devuelve la carpeta (como instancia de File) asignado en segundo lugar en el método *setFolders*().

d) *Iterator<ResultadoComparacion> compare()*. Realiza la comparación entre los dos directorios asignados por medio del método *setFolders()* y devuelve un iterador de instancias de *ResultadoComparacion.*

e) *ResultadoComparacion* es una clase que permite guardar el nombre de un archivo y el valor indicando si existe el mismo archivo en las dos carpetas o sólo en una de ellas. En caso de que exista en las dos carpetas, indicará si se pueden considerar iguales o no. Según cuál sea el caso, el valor coincidirá con alguno de los nombres simbólicos siguientes:

- IGUALES: significa que en cada carpeta hay un archivo con el nombre indicado y de las mismas características (fecha modificación y tamaño).
- FALTA_EN_1: significa que el archivo está en la segunda carpeta analizada pero no en la primera.
- FALTA_EN_2: significa que el archivo está en la primera carpeta analizada pero no en la segunda.
- MAS_NUEVO_EN_1: significa que en las dos carpetas analizadas existe un archivo con el nombre indicado, pero no son iguales. El de la primera carpeta es más nuevo que el de la segunda.
- MAS_NUEVO_EN_2: significa que en las dos carpetas analizadas existe un archivo con el nombre indicado, pero no son iguales. El de la segunda carpeta es más nuevo que el de la primera.

Las características que deberá tener en cuenta en la comparación son el nombre, la fecha de modificación y el tamaño del archivo.

La clase ResultadoComparacion dispondrá como mínimo de los siguientes métodos:

f) *String getNombreArchivo()*: este método devuelve el nombre relativo del elemento que se intenta determinar si existe en los dos directorios y si son idénticos.

g) *ValorComparacion getValorComparacion()*: ValorComparacion es una enumeración con los valores: IGUALES, FALTA_EN_1, FALTA_EN_2, MAS_NUEVO_EN_1, MAS_NUEVO_EN_2.

16. Diseña una clase para gestionar el estado de una partida del 3 en raya. No es necesario que diseñes todas las clases del juego, sólo la clase para "congelar" el estado de una partida. Es decir, la posición de las fichas, a qué jugador le tocaría jugar, y la puntuación acumulada (número de

manos que ha ganado cada jugador).

Implementar también una clase llamada GameStorage que gestione el guardado en disco del estado de la partida y su posterior recuperación. Realiza esta implementación utilizando la técnica de la serialización propia de Java con ObjectOutputStream y ObjectInputStream.

17. Añade una opción a la práctica de BlackJack desarrollada en el tema anterior, para guardar el estado de una partida, para de esta forma, poder continuarla justo en el punto donde se había quedado. Para ello utiliza GSON, que hemos visto en el Anexo, se trata de una librería desarrollada por Google que permite serializar/desearializar objetos utilizando JSON.