# 10.F. Actividades propuestas.

# Tabla de contenidos

# 1 ActividadUT10-1: Propiedades de fichero.

Escribe un fuente denominado **PropiedadesFichero** que se ejecute de la siguiente forma: **java PropiedadesFichero nomFich1 nomFich2....** (es decir, que maneje parámetros del main).

Crea una **Exception** propia si el número de argumentos no es correcto (será incorrecto cuando no pasemos ningún parámetro, es decir, 0 ficheros, y será correcto, cuando pasemos uno o más). En caso de que el fichero exista, invoca a un método denominado **propiedades()** que haga lo siguiente (esto lo hará para cada uno de los ficheros pasados como parámetros):

Si el fichero es ordinario, lo indica y muestra en pantalla: nombre, .camino absoluto, camino relativo, si se puede leer y si se puede modificar; si no, si es un directorio muestra la lista del los ficheros que están en ese path.

# 2 ActividadUT10-2: Crear fichero de texto carácter a carácter.

Escribe un programa denominado **CrearFicheroTexto** que se ejecute de la siguiente forma:

**C:/> java CrearFicheroTexto texto.txt**

El programa deberá controlar que el número de parámetros pasados a main es correcto y manejará excepciones de tipo **IOException**. Si el número de argumentos es correcto, invocará a un método denominado **crearFichero()**, donde se llevará a cabo la creación del fichero de texto. En caso de que ya exista, le preguntaremos si desea sobrescribirlo, validando la respuesta (s/n). Si el número de parámetros no es correcto, lanzará una excepción propia indicando que los parámetros son incorrectos y la sintaxis correcta para ejecutar el programa. La información se procesará carácter a carácter, leyendo con método **System.in.read()**, y casteando su salida con **(char)**. Utilizar como flujo de salida un **FileWriter**.

# 3 ActividadUT10-3: Crear fichero de texto línea a línea.

Repite el ejercicio anterior procesando la información línea a línea . El fuente se puede llamar **CrearFicheroTextoLineas**.

Para ello crearemos un flujo lector de tipo **BufferedReader** de la forma:

**BufferedReader flujoLector = new BufferedReader(new InputStreamReader(System.*in*));**

Dicho flujo lector leerá líneas mediante método **readLine()**.

El flujo de salida para la escritura en fichero podrá incorporará las líneas pasándolas como parámetros a su método **write()**.

# 4 ActividadUT10-4: Leer fichero de texto carácter a carácter.

Escribe un programa denominado **LeerFicheroTexto** que se ejecute de la siguiente forma:

**C> java LeerFicheroTexto texto.txt**

El programa deberá controlar que el número de parámetros pasados a main es correcto, manejará excepciones de tipo **IOException**, si el número de argumentos es correcto, invocará a un método denominado **mostrarFichero**, donde se llevará a cabo la visualización del fichero de texto. Si el número de parámetros no es correcto lanzará una excepción propia indicando que los parámetros son incorrectos y la sintaxis correcta para ejecutar el programa. La información se procesará carácter a carácter. Utilizar como flujo de entrada un **FileReader**, invocando a su método **read()**, y casteando su salida con **(char)**.

# 5 ActividadUT10-5: Leer fichero de texto línea a línea.

Repite el ejercicio anterior procesando la información línea a línea . El fuente se puede llamar **LeerFicheroTextoLineas**.

Para ello crearemos un flujo lector o de entrada de tipo **BufferedReader** de la forma:

**BufferedReader flujoEntrada = new BufferedReader(new** FileReader(fichero));

Dicho flujo lector leerá líneas mediante método **readLine()**. Iremos leyendo las líneas llamando desde ese flujo de entrada al método **readLine()** sucesivamente hasta que ya no encuentre más líneas, circunstancia que se dará al llegar a la marca **EOF** (End Of File).

# 6 ActividadUT10-7: Mostrar agenda de teléfono.

Para leer el fichero creado por la aplicación anterior, vamos a escribir otra basada en una clase MostrarListaTfnos. Esta clase define dos métodos: mostrarFichero y main. El método mostrarFichero recibe como parámetro un objeto String que almacena el nombre del fichero que se desea leer y realiza las tareas siguientes:

- Crea un objeto File para identificar al fichero.
- Si el fichero especificado existe, crea un flujo desde el mismo que permite leer datos de tipos primitivos utilizando un buffer.
- Lee un grupo de datos nombre, dirección y teléfono desde el fichero y los muestra. Cuando se alcance el final del fichero Java lanzará una excepción del tipo EOFException, instante en el que finalizará la ejecución del método.
- Si durante la ejecución alguno de los métodos invocados lanza una excepción IOException, la pasa para que sea atrapada que le invocó.

El método main recibe como parámetro el nombre del fichero que se desea mostrar y realiza las tareas siguientes:

- Verifica si se pasó un argumento cuando se ejecutó la aplicación con el nombre del fichero cuyo contenido, se desea visualizar.
- Si no se pasó un argumento, la aplicación mostrará un mensaje indicando la sintaxis que debe emplear para ejecutar la misma y finalizará. En otro caso invoca al método mostrarFichero pasando como argumento args[0].

Nota importante: De cara a poder probar íntegramente este ejercicio y el anterior, es muy recomendable incorporar los 2 ejercicios en un mismo proyecto.

# 7 ActividadUT10-6: Crear agenda de teléfono.

Para escribir en un fichero datos de tipo primitivos (boolean, double, float, long, int y short) el paquete **java.io** proporciona las clases **DataInputStream** y **DataOutputStream**, la cuales permiten leer y escribir, respectivamente. Un flujo **DataInputStream** solo puede leer datos almacenados en un fichero a través de un flujo **DataOutputStream**. Los flujos de estas clases actúan como filtros; esto es, los datos obtenidos del origen o enviados al destino son transformados mediante alguna operación; en este caso, sufren una conversión a un formato portable (UTF-8:Unicode ligeramente modificado) cuando son almacenados y viceversa cuando son recuperados. El procedimiento para utilizar el filtro y un buffer es:

**// crea un flujo hacia el fichero que permita escribir datos en el fichero**

**dos = new DataOutputStream(new BufferedOutputStream(new FileOutputStream(fichero)));**

**// lee nombre del teclado**

**nombre = Leer.dato();**

**//Almacenar un nombre,**

**dos.writeUTF(nombre);**

**writeUTF** escribe una cadena de caracteres en formato UTF-8, los dos primeros bytes especifican el número de bytes de datos escritos a continuación.

Así mismo para poder leer en el fichero:

**dis = new DataInputStream(new BufferedInputStream( new (FileInputStream(fichero)));**

**// lee un nombre**

**nombre = dis.readUTF();**

**//visualiza el nombre**

**System.out.println(nombre);**

Para practicar esto, vamos a escribir una aplicación denominada **CrearListaTfnos.java**. Nuestra aplicación puede construir una lista de teléfonos de la siguiente forma:

La clase contará con un método **main()**, donde como siempre solicitamos por teclado un nombre de fichero; creamos un objeto de tipo fichero asociado a ese nombre, a continuación verificamos si existe, en caso afirmativo preguntamos si desea sobrescribirlo. Si la respuesta es afirmativa invocamos a un método denominado **crearFichero()**. Este método recibe un objeto de tipo **File** y manejando un flujo de salida de datos hacia el fichero, pedimos por teclado el nombre, dirección y teléfono y almacenamos los datos en el fichero. Cada vez que grabemos un registro le preguntamos si desea escribir otro. Utiliza el manejo de excepciones **IOException** ( **finally** para cerrar el flujo).

# 8 ActividadUT10-8: Crear agenda de teléfono con serialización.

Repetir la actividad "ActividadUT10-6: Crear agenda de teléfono", utilizando el concepto de serialización de objetos.

La clase que utilicemos para implementar el objeto (en nuestro caso **Persona**) debe implementar la interfaz **Serializable**.

# 9 ActividadUT10-9: Mostrar agenda de teléfono con serialización.

Repetir la actividad "ActividadUT10-7: Mostrar agenda de teléfono", utilizando el concepto de serialización de objetos (la clase a utilizar sera **ObjectInputStream**).

Nota importante: De cara a poder probar íntegramente este ejercicio y el anterior, es muy recomendable incorporar los 2 ejercicios en un mismo proyecto.

# 10 ActividadUT10-10: Comando cp

Simular el comando cp de Unix/Linux. Se va a tener que copiar un fichero de texto a otro. La Sintaxis seria: **java Copiar nomFichTexto1 nomFichTexto2**.