ANÁLISIS LISTA DE USUARIOS

Cuando el programa se ejecuta, se le solicita al usuario el nombre del archivo inicial con el que va a trabajar el programa en la creación de usuarios. Este documento está ordenado de la siguiente manera:

- La primera línea contiene el número de usuarios en la red.
- La segunda línea contiene el número de conexiones existentes en la red (que están almacenadas en el archivo)
- Desde la tercera línea en adelante se almacenan los datos de las conexiones siguiendo la siguiente estructura: Número de un usuario, espacio, número de otro usuario (123 456).

Cuando el usuario seleccione el archivo, el programa leerá todo el archivo gracias a un FileReader y a un BufferedReader. Durante la lectura, almacenará las líneas leídas en la propiedad datosArchivo, el cual es un ArrayList, del objeto Análisis. Se utiliza la clase BufferedReader ya que es de las mejores para leer archivos muy extensos en un tiempo bastante bajo.

Una vez finalizada la lectura, se cierra el FileReader y el BufferedReader para evitar consumir recursos de forma innecesaria.

Todo esto se realiza gracias a la función leerArchivo, que devuelve un dato de tipo Boolean para expresar si el archivo se ha leído correctamente devolviendo el valor TRUE o, si en cambio ha ocurrido algún error durante la lectura, devolverá el valor FALSE para poder detener la ejecución. Además, también recibe por parámetro un valor de tipo Boolean llamado tipoFichero. Esto es para poder procesar tanto el fichero principal como el fichero extendido de conexiones con la misma función.

Tras leer el fichero principal, el programa preguntará por el fichero extendido. Si este existe, el usuario deberá introducir su nombre junto con su extensión (“extra.txt”). En cambio, si este fichero no existe, deberá pulsar intro y el programa continuará su ejecución.

Cuando se procesa el fichero extendido, se añaden sus datos (conexiones) al final del ArrayList anteriormente mencionado.

Todos los archivos deben estar guardados en una carpeta llamada “DOCS” en la raíz del proyecto.

A continuación se muestra la función leerArchivo():

```java
public static boolean leerArchivo(boolean tipoFichero) {
    FileReader fileReader = null;
    BufferedReader bufferedReader = null;
    String archivoALeer; // Archivo que se lee en la ejecucion actual

    if (tipoFichero) {
        archivoALeer = elAnalisis.nombreFicheroPrincipal;
    } else {
        archivoALeer = elAnalisis.nombreFicheroNuevasConexiones;
    }

    try {
        // Para la lectura del fichero
        fileReader = new FileReader("DOCS/" + archivoALeer);
        bufferedReader = new BufferedReader(fileReader);

        elAnalisis.tILecturaFichero = hora(); // Guardo el tiempo de inicio de lectura del fichero

        String lineaLeida; // Linea leida
        while ((lineaLeida = bufferedReader.readLine()) != null) {
            elAnalisis.datosArchivo.add(lineaLeida); // Lo agregro al arrayList datosArchivo del objeto Analisis para su posterior procesamiento
        }

        // Intento cerrar el archivo y su lectura
        fileReader.close();
        bufferedReader.close();

        elAnalisis.tFLecturaFichero = hora(); // Guardo el tiempo final de lectura del fichero
    } catch (FileNotFoundException fnfEx) {
        if (tipoFichero) { // El fichero que ha dado la excepción es el principal
            System.err.println("No se ha encontrado el archivo " + elAnalisis.nombreFicheroPrincipal);
        } else { // El fichero que ha dado la excepción es el de nuevas conexiones
            System.err.println("No se ha encontrado el archivo " + elAnalisis.nombreFicheroNuevasConexiones);
        }
        return false;
    } catch (IOException ioEx) {
        System.err.println("Un archivo, o los dos, no se pudo cerrar bien.");
        return false;
    } catch (Exception ex) { // Cualquier otra excepción no controlada
        System.err.println("Error no controlado.");
        return false;
    }

    return true;
}
```

Una vez terminada la lectura de los dos archivos (si es que existiera el archivo de conexiones extendidas), se ejecuta la función procesarConexiones(), la cual recorrerá el ArrayList datosArchivo (del objeto Análisis) desde su segunda posición. En este momento, se recogerá cada conexión y se creará un nuevo objeto Conexión pasando por parámetro al constructor la línea leída del ArrayList. En este momento, el trabajo lo hará el constructor del objeto Conexión.

Cuando se llama al constructor del objeto Conexión pasando por parámetro una línea de conexión del archivo, el constructor separará el primer usuario del segundo de la conexión ya que estos están separados entre sí por un espacio. En este momento, se agregará "a sí mismo", es decir, el objeto que se acaba de crear, al ArrayList red, que pertenece al objeto Análisis. Tras esto, comprobará si el usuario1 y el usuario2 están en el listado de usuarios (ArrayList usr del objeto Análisis). Si no están contenidos, se agregan.

A continuación una captura de la función procesarConexiones():

```java
private static void procesarConexiones() {
    Conexion laConexion; // Instancia del objeto Conexion

    elAnalisis.tIListaUsuarios = hora(); // Guardo el tiempo de inicio del procesamiento de la lista de usuarios

    elAnalisis.numeroUsuarios = Integer.parseInt(elAnalisis.datosArchivo.get(0).toString());
    elAnalisis.numeroConexiones = Integer.parseInt(elAnalisis.datosArchivo.get(1).toString());

    for (int i = 2; i < elAnalisis.datosArchivo.size(); i++) { // Empiezo en 2 porque las posiciones 0 y 1 son el numero de usuarios y las conexiones
        String conexion = elAnalisis.datosArchivo.get(i).toString(); // Recojo la conexion existente

        /**
         * Creo un nuevo objeto Conexion y le mando como paramtero al
         * constructor la conexión recogida. El constructor será el
         * encargado de separar los usuarios de la conexión, crear la nueva
         * conexión y almacenarla
         */
        laConexion = new Conexion(conexion);
    }
    elAnalisis.tFListaUsuarios = hora(); // Guardo el tiempo de finalización del procesamiento de la lista de usuarios
}
```

Y, el constructor del objeto Conexión:

```java
Conexion(String datos) {
    this.usuario1 = Integer.parseInt(datos.split(" ")[0]);
    this.usuario2 = Integer.parseInt(datos.split(" ")[1]);

    elAnalisis.listadoConexiones.add(this);

    if (!elAnalisis.listadoUsuarios.contains(usuario1)) { //Si el usuario no existe
        elAnalisis.listadoUsuarios.add(usuario1);
    }

    if (!elAnalisis.listadoUsuarios.contains(usuario2)) {
        elAnalisis.listadoUsuarios.add(usuario2);
    }
}
```

Resultado final con archivo de pruebas "test20000.txt":

```
Output - CaraLibro (run)
run:
ANÁLISIS DE CARALIBRO
---------------------
Fichero principal: test20000.txt
Lectura fichero: 0,01900 seg.
Fichero de nuevas conexiones (pulse enter si no existe):
20000 usuarios, 19919 conexiones
Porcentaje tamaño mayor grumo: 90
Creación lista usuarios: 0,91600 seg.
Creación lista grumos: 3,39200 seg.
Existen 81 grumos.
Se deben unir los 5 mayores
#1: 4296 usuarios 21.0 %
#2: 4290 usuarios 21.0 %
#3: 4294 usuarios 21.0 %
#4: 4292 usuarios 21.0 %
#5: 2652 usuarios 13.0 %
Nuevas relaciones de amistad (salvadas en extra.txt)
16252944 <-> 48103433
48103433 <-> 79908463
79908463 <-> 85370320
85370320 <-> 75387301
Ordenación y selección de grumos: 0,00200 seg.
BUILD SUCCESSFUL (total time: 18 seconds)
```