MANEJOS DE FICHEROS

Se aprenderá todos los conceptos básicos que hay que tener en cuenta cuando necesitamos tratar con ficheros con lenguaje Java.

1. Introducción

Un fichero es un archivo que contendrá un conjunto de caracteres o bytes que se almacenarán en un **soporte persistente en una ruta y con un nombre concreto**. Es el archivo que usará nuestro programa para almacenar, leer, escribir o gestionar información sobre el proceso que se esté ejecutando.

Existen tipos diferentes de ficheros:

- **Fichero estándar**: es un archivo que contiene todo tipo de datos: caracteres, imagen, audio, vídeo, etcétera. Normalmente son ficheros que contienen información de cualquier tipo.
- **Directorios o carpetas:** son ficheros que albergan más archivos en su interior. Su principal utilidad es mantener un orden o jerarquía en nuestros sistemas.
- **Ficheros especiales:** son todos esos ficheros que usa nuestro sistema operativo y que se utilizan para controlar los dispositivos o periféricos de nuestro ordenador (driver).

Podemos destacar dos tipos de ficheros de datos:

- **Los ficheros de bytes:** también conocidos como ficheros binarios, son archivos que usan los programas para leer o escribir información.
- **Los ficheros de caracteres** también conocidos como ficheros de texto, nos permiten leer o escribir información que contengan.

Un fichero se caracteriza por estar formado por la ruta en la que está almacenado, el nombre y una extensión. La extensión del archivo nos permitirá diferenciar qué programa puede utilizar ese fichero. **Se considera extensión todo lo que podemos encontrar después del punto que ponemos al final del nombre.**

Los archivos que trataremos, contendrán información de texto o caracteres. Cada lengua utiliza un tipo de carácter distinto de otra, por ejemplo, el ruso utiliza un abecedario diferente que el español, por lo que usará caracteres distintos. Los caracteres se almacenan en nuestro ordenador como uno o más bytes. Todos los caracteres están almacenados en ordenadores usando un código especial, es decir, una codificación de caracteres proporciona una clave para descifrar el código. Es un conjunto de asignaciones entre los bytes de los ordenadores y los caracteres en el conjunto de caracteres.

Se denomina encoding al sistema utilizado para transformar los caracteres que usa cada lenguaje en un símbolo que un ordenador pueda interpretar. Ejemplos: Ascii, ISO-8859, Unicode(UTF-8, UTF-16..)

2. Formas de acceso a un fichero.

El acceso a un fichero puede ser de tres formas principales.

- ➢ **Acceso secuencial**. Se leen y escriben los datos del mismo modo que se hace en una antigua cinta de música. Si se quiere acceder a un dato que está hacia la mitad de un fichero, habrá que pasar primero por todos los datos anteriores. Los ficheros de texto son de acceso secuencial. Este es el caso de la lectura del teclado o la escritura en una consola de texto, no se sabe cuándo el operador terminará de escribir.
- ➢ **Concatenación (tuberías o "pipes")**: Muchas veces es útil hacer conexiones entre programas que se ejecutan simultáneamente dentro de una misma máquina, de modo que lo que uno produce se envía por un "tubo" para ser recibido por el otro, que está esperando a la salida del tubo. Las tuberías cumplen esta función.

- ❖ **Acceso aleatorio (random):** Permiten acceder directamente a un dato sin tener que pasar por todos los demás, y pueden acceder a la información en cualquier orden. Tienen la limitación de que los datos están almacenados en unas unidades o bloques que se llaman registros, y que todos los registros que se almacenan en un fichero deben ser del mismo tamaño. Los ficheros de acceso aleatorio son ficheros binarios. Accede a los datos en forma no secuencial, desordenada. Esto implica que el archivo debe estar disponible en su totalidad al momento de ser accedido, algo que no siempre es posible.

- **Acceso binario**. Son como los de acceso aleatorio, pero el acceso no se hace por registros sino por bytes.

3. Clases asociadas a operaciones de gestión de ficheros y directorios: creación, borrado, copia, movimiento, entre otras.

La clase **File** proporciona una representación abstracta de ficheros y directorios.

- Esta clase permite examinar y manipular archivos y directorios, independientemente de la plataforma en la que se esté trabajando: Linux, Windows, etc.

- Las instancias de la clase File representan nombres de archivo, no los archivos en sí mismos.

- El archivo correspondiente a un nombre puede ser que no exista, por esta razón habrá que controlar las posibles excepciones.

- Un objeto de clase File permite examinar el nombre del archivo, descomponerlo en su rama de directorios o crear el archivo si no existe, pasando el objeto de tipo File

a un constructor adecuado como FileWriter(File f), que recibe como parámetro un objeto File.

- Para archivos que existen, a través del objeto File, un programa puede examinar los atributos del archivo, cambiar su nombre, borrarlo o cambiar sus permisos.

- Dado un objeto File, podemos hacer las siguientes operaciones con él:

  ➔ Renombrar el archivo, con el método renameTo(). El objeto File dejará de referirse al archivo renombrado, ya que el String con el nombre del archivo en el objeto File no cambia.
  ➔ Borrar el archivo, con el método delete(). También, con **deleteOnExit()** se borra cuando finaliza la ejecución de la máquina virtual Java.
  ➔ Crear un nuevo fichero con un nombre único. El método estático **createTempFile()** crea un fichero temporal y devuelve un objeto File que apunta a él. Es útil para crear archivos temporales, que luego se borran, asegurándonos tener un nombre de archivo no repetido.
  ➔ Establecer la fecha y la hora de modificación del archivo con **setLastModified()**. Por ejemplo, se podría hacer:

  new File("prueba.txt").setLastModified(new Date().getTime());
  para establecer la fecha actual al fichero que se le pasa como parámetro.

  ➔ Crear un directorio, mediante el método **mkdir()**. También existe mkdirs(), que crea los directorios superiores si no existen.
  ➔ Listar el contenido de un directorio. Los métodos list() y listFiles() listan el contenido de un directorio. list() devuelve un vector de String con los nombres de los archivos, listFiles() devuelve un vector de objetos File.
  ➔ Listar los nombres de archivo de la raíz del sistema de archivos, mediante el método estático listRoots().

- La clase proporciona los siguientes constructores para crear objetos File:

  public File(String nombreFichero|path);
  public File(String path, String nombreFichero|path);
  public File(File path, String nombreFichero|path);

  La ruta o path puede ser absoluta o relativa.

  Ejemplos utilizando el primer constructor:
  File f = new File("personas.dat");

Crea un Objeto File asociado al fichero personas.dat que se encuentra en el directorio de trabajo. En este caso no se indica path. Se supone que el fichero se encuentra en el directorio actual de trabajo.

File f = new File("ficheros/personas.dat");

Crea un Objeto File asociado al fichero personas.dat que se encuentra en el directorio ficheros dentro del directorio actual. En este caso se indica la ruta relativa tomando como base el directorio actual de trabajo. Se supone que el fichero personas.dat se encuentra en el directorio ficheros. A su vez el directorio ficheros se encuentra dentro del directorio actual de trabajo.

File f = new File("c:/ficheros/personas.dat");

Crea un Objeto File asociado al fichero personas.dat dando la ruta absoluta:

Ejemplos utilizando el segundo constructor:
En este caso se crea un objeto File cuya ruta (absoluta o relativa) se indica en el primer String.

File f = new File("ficheros", "personas.dat");

Crea un Objeto File asociado al fichero personas.dat que se encuentra en el directorio ficheros dentro del directorio actual. En este caso se indica la ruta relativa tomando como base el directorio actual de trabajo.

Ejemplos utilizando el tercer constructor:
Este constructor permite crear un objeto File cuya ruta se indica a través de otro objeto File.

```
File ruta = new File("ficheros");
File f = new File(ruta, "personas.dat" );
```

Crea un Objeto File asociado al fichero personas.dat que se encuentra en el directorio ficheros dentro del directorio actual.

Debemos tener en cuenta que crear un objeto File no significa que deba existir el fichero o el directorio o que el path sea correcto. Si no existen no se lanzará ningún tipo de excepción ni tampoco serán creados.

Otros métodos de la clase FILE

boolean canRead()
Devuelve true si se puede leer el fichero
boolean canWrite()
Devuelve true si se puede escribir en el fichero
boolean createNewFile()
Crea el fichero asociado al objeto File. Devuelve true si se ha podido crear. Para poder crearlo el fichero no debe existir. Lanza una excepción del tipo IOException.
boolean delete()
Elimina el fichero o directorio. Si es un directorio debe estar vacío. Devuelve true si se ha podido eliminar.
boolean exists()
Devuelve true si el fichero o directorio existe
String getName()
Devuelve el nombre del fichero o directorio

String getAbsolutePath()
Devuelve la ruta absoluta asociada al objeto File.
String getCanonicalPath()
Devuelve la ruta única absoluta asociada al objeto File. Puede haber varias rutas absolutas asociadas a un File pero solo una única ruta canónica. Lanza una excepción del tipo IOException.
String getPath()
Devuelve la ruta con la que se creó el objeto File. Puede ser relativa o no.
String getParent()
Devuelve un String conteniendo el directorio padre del File. Devuelve null si no tiene directorio padre.
File getParentFile()
Devuelve un objeto File conteniendo el directorio padre del File. Devuelve null si no tiene directorio padre.
boolean isAbsolute()
Devuelve true si es una ruta absoluta
boolean isDirectory()
Devuelve true si es un directorio válido
boolean isFile()
Devuelve true si es un fichero válido
long lastModified()
Devuelve un valor en milisegundos que representa la última vez que se ha modificado (medido desde las 00:00:00 GMT, del 1 de Enero de 1970). Devuelve 0 si el fichero no existe o ha ocurrido un error.
long length()
Devuelve el tamaño en bytes del fichero. Devuelve 0 si no existe. Devuelve un valor indeterminado si es un directorio.
String[] list()
Devuelve un array de String con el nombre de los archivos y directorios que contiene el directorio indicado en el objeto File. Si no es un directorio devuelve null. Si el directorio está vacío devuelve un array vacío.
String[] list(FilenameFilter filtro)
Similar al anterior. Devuelve un array de String con el nombre de los archivos y directorios que contiene el directorio indicado en el objeto File que cumplen con el filtro indicado.
boolean mkdir()
Crea el directorio. Devuelve true si se ha podido crear.
boolean mkdirs()
Crea el directorio incluyendo los directorios no existentes especificados en la ruta padre del directorio a crear. Devuelve true si se ha creado el directorio y los directorios no existentes de la ruta padre.
boolean renameTo(File dest)
Cambia el nombre del fichero por el indicado en el parámetro dest. Devuelve true si se ha realizado el cambio.

4. Existencia y listado de ficheros y carpetas.

Mediante la clase File, podemos ver si un fichero cualquiera, digamos por ejemplo texto.txt, existe o no. Para ello, nos valemos del método **exists()** de la clase File. Hacemos referencia a ese fichero en concreto con el código siguiente:

```
File f = new File("C:\texto.txt");
if (fichero.exists ())
{
 System.out.println(“el fichero existe”);

}
else
{

  System.out.println(“El fichero no existe”);

}
```

5. Creación y eliminación de ficheros y directorios.

```
try {
// Creamos el objeto que encapsula el fichero
File fichero = new File("c:\\prueba\\miFichero.txt");

// A partir del objeto File creamos el fichero físicamente

if (fichero.createNewFile())
   System.out.println("El fichero se ha creado correctamente");
else
System.out.println("No ha podido ser creado el fichero");
} catch (Exception ioe) {
ioe.getMessage();
}
```

Para borrar un fichero podemos usar la clase File, comprobando previamente si existe el fichero, del siguiente modo:

```
File fichero = new File("c:\\prueba\\miFichero.txt");
if (fichero.exists ())
fichero.delete();
```

Para crear un directorio, lo realizaremos del siguiente modo:

```
Try{
// Declaración de variables
      String directorio = "C:\\prueba";
      String varios = "carpeta1/carpeta2/carpeta3";
// Crear un directorio
      boolean exito = (new File(directorio)).mkdir();
        if (exito)
            System.out.println("Directorio: " + directorio + " creado");
```

```
// Crear varios directorios
        exito = (new File(varios)).mkdirs();
            if (exito)
            System.out.println("Directorios: " + varios + " creados");
         }catch (Exception e){
    System.err.println("Error: " + e.getMessage());
}

//otro ejemplo

public static void main(String args[]){
       File directorio = new File("/ruta/directorio_nuevo");
       if (!directorio.exists()) {
          if (directorio.mkdirs()) {
             System.out.println("Directorio creado");
          } else {
             System.out.println("Error al crear directorio");
          }
       }
   }
```

Para borrar un directorio con Java, tendremos que borrar cada uno de los ficheros y directorios que este contenga. Al poder almacenar otros directorios, se podría recorrer recursivamente el directorio para ir borrando todos los ficheros.

Podemos listar el contenido de un directorio e ir borrando con:

File [] ficheros = directorio.listFiles ();
Si el elemento es un directorio, lo sabemos mediante el método isDirectory ().

Después de vaciar el directorio, este, se puede borrar importando la librería FileUtils y después escribir FileUtils.deleteDirectory(newFile(destination));

6. Interface FilenameFilter.

Hemos visto cómo obtener la lista de ficheros de una carpeta o directorio. A veces, nos interesa ver los archivos que **encajan con un determinado criterio**.

Por ejemplo, nos puede interesar un filtro para ver los ficheros modificados después de una fecha, o los que tienen un tamaño mayor del que el que indiquemos, etc.

La interfaz **FilenameFilter** se puede usar para crear filtros que establezcan criterios de filtrado relativos al nombre de los ficheros. **Una clase que lo implemente debe definir e implementar el método:**

boolean accept(File dir, String nombre)

Este método devolverá verdadero en el caso de que el fichero cuyo nombre se indica en el parámetro nombre aparezca en la lista de los ficheros del directorio indicado por el parámetro dir.

En el siguiente ejemplo vemos cómo se listan los ficheros de la carpeta c:\datos que tengan la extensión .txt. Usamos try y catch para capturar las posibles excepciones, como que no exista dicha carpeta.

```
import java.io.File;
import java.io.FilenameFilter;

public class Filtrar implements FilenameFilter {

String extension;
  // Constructor
  Filtrar(String extension)
   {
         this.extension = extension;
   }
  public boolean accept(File dir, String name)
  {
    return name.endsWith(extension);
  }

public static void main(String[] args) {
try {
    // Obtendremos el listado de los archivos de ese directorio
    File fichero=new File("c:\\datos\\.");
    String[] listadeArchivos = fichero.list();
     // Filtraremos por los de extension .txt
     listadeArchivos = fichero.list(new Filtrar(".txt"));
     // Comprobamos el número de archivos en el listado
     int numarchivos = listadeArchivos.length ;
     // Si no hay ninguno lo avisamos por consola
     if (numarchivos < 1)
        System.out.println("No hay archivos que listar");
        // Y si hay, escribimos su nombre por consola.
        else
      {
           for(int conta = 0; conta < listadeArchivos.length; conta++)
           System.out.println(listadeArchivos[conta]);
    }
}
catch (Exception ex) {
System.out.println("Error al buscar en la ruta indicada"); }
```

7. Ruta de los ficheros

Un fichero se caracteriza por estar formado por la ruta en la que está almacenado, el nombre y una extensión, siguiendo este orden. Además, tenemos que tener en cuenta que un fichero es un archivo que contendrá un conjunto de caracteres o bytes que se almacenarán en el dispositivo en una ruta y con un nombre concreto.
No podrán existir ficheros con el mismo nombre, ruta y extensión. Para que sean únicos, el nombre o la extensión en la misma ruta deben ser distintos.
Para tener acceso a un fichero determinado, se utiliza una ruta (o también la podemos nombrar path) que indica la ubicación de ese fichero en nuestro sistema. La ruta está compuesta por diferentes niveles jerárquicos (carpetas) separado por un símbolo barra /, Aunque en Windows, para separar los niveles jerárquicos, se utiliza la contrabarra o \. En cambio, en Unix el separador será /.
Si queremos definir la ruta independientemente del sistema operativo, podemos realizarlo de este modo:

```
//Ejemplo con la ruta directa al string
        File archivoNoseguro = new File("carpeta/ejemplo.txt");
        //Ruta que asegura el separador correcto segun plataforma
        File archivo = new File("carpeta"+File.separator+"ejemplo.txt")
```

8. Flujos: basados en bytes y basados en caracteres

Una de las acciones más importantes y útiles de los ficheros es la posibilidad de leer el contenido y escribir en él. Para el tratamiento de los flujos de bytes, hemos dicho que Java tiene dos clases abstractas que son **InputStream y OutputStream**.

Los archivos binarios guardan una representación de los datos en el archivo, es decir, cuando guardamos texto no guardan el texto en sí, sino que guardan su representación en un código llamado UTF-8.
UTF-8 es un formato de codificación de caracteres Unicode. Es el responsable de que tu navegador o tu cliente de correo te muestre el contenido del texto correctamente decodificado, sin errores ni caracteres extraños.

Las clases principales que heredan de OutputStream, para la escritura de ficheros binarios son:

FileOutputStream: escribe bytes en un fichero. Si el archivo existe, cuando vayamos a escribir sobre él, se borrará. Por tanto, si queremos añadir los datos al final de éste, habrá que usar el constructor **FileOutputStream(**String filePath, boolean append), poniendo append a true.
ObjectOutputStream: convierte objetos y variables en vectores de bytes que pueden ser escritos en un OutputStream.
DataOutputStream, que da formato a los tipos primitivos y objetos String, convirtiéndolos en un flujo de forma que cualquier DataInputStream, de cualquier máquina, los pueda leer. Todos los métodos empiezan por "write", como writeByte(), writefloat(), etc.

De InputStream, para lectura de ficheros binarios, destacamos:

FileInputStream: lee bytes de un fichero.

ObjectInputStream: convierte en objetos y variables los vectores de bytes leídos de un InputStream.
DataInputStream define diversos métodos readXXX que son variaciones del método read de la clase base para leer datos de tipo primitivo:

```
boolean readBoolean(); // lee un tipo booleano
byte readByte(); //lee un byte
int readUnsignedByte();
short readShort();
int readUnsignedShort();
char readChar(); //lee un carácter
int readInt(); // lee un entero
long readLong();
float readFloat();
double readDouble();
```

Ejemplo: escritura a fichero.

En el siguiente ejemplo se puede ver cómo se escribe a un archivo binario con DataOutputStream:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class CopiaFicheros {
   public static void main(String args[]) {
      // Copiar ficheros
      File origen = new File("origen.txt");
      File destino = new File("destino.txt");
      try {
         InputStream in = new FileInputStream(origen);
         OutputStream out = new FileOutputStream(destino);
         byte[] buf = new byte[1024];
         int len;
         while ((len = in.read(buf)) > 0) {
           out.write(buf, 0, len);
        }
        in.close();
        out.close();
     } catch (IOException ioe) {
       ioe.printStackTrace();
     }
   }
}
```

```
//Suma de números introduciendo los valores por teclado.

  public void sumaNumerosTeclado()
   {try{
     DataInputStream input = new DataInputStream( System.in );
     int valor, total = 0;
     System.out.print("\n["+total+"] ");
     while((valor = input.readInt()) !=-1){

        total += valor;
        System.out.print("\n["+total+"] ");
     }
     System.out.println("\n["+total+"]\n");



    }catch(IOException e)
    {


    }

}
```

En Java se accede a la E/S estándar a través de campos estáticos de la clase java.lang.System:

1. System.in implementa la entrada estándar
2. System.out implementa la salida estándar
3. System.err implementa la salida de error

9. Clases para gestión de flujos de datos desde/hacia ficheros.

Para los flujos de caracteres, Java dispone de dos clases abstractas: **Reader y Writer**.

Si se usan sólo FileInputStream, FileOuputStream, FileReader o FileWriter, cada vez que se efectúa una lectura o escritura, **se hace físicamente en el disco duro**. Si se leen o escriben pocos caracteres cada vez, el proceso se hace costoso y lento por los muchos accesos a disco duro.

Los BufferedReader, BufferedInputStream, BufferedWriter y BufferedOutputStream **añaden un buffer intermedio**. Cuando se lee o escribe, esta clase controla los accesos a disco. Así, si vamos escribiendo, se guardarán los datos hasta que haya bastantes datos como para hacer una escritura eficiente.

Al leer, la clase leerá más datos de los que se hayan pedido. En las siguientes lecturas nos dará lo que tiene almacenado, hasta que necesite leer otra vez físicamente. Esta

forma de trabajar hace los accesos a disco más eficientes y el programa se ejecuta más rápido.
Además FileReader no contiene métodos que nos permitan leer líneas completas, pero sí BufferedReader.
Afortunadamente, podemos construir un BufferedReader a partir del FileReader de la siguiente manera:

```
File archivo = new File ("C:\\archivo.txt");
FileReader fr = new FileReader (archivo);
BufferedReader br = new BufferedReader(fr);
String linea = br.readLine();
```

Ejemplo: lectura de un fichero
En este ejemplo leemos caracteres hasta llegar al fin del archivo (EOF o End Of File) del flujo.

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
public class LeerFichEOF {
        public static void main (String args[]) throws IOException {
                    BufferedReader br = new BufferedReader(new FileReader("origen.txt")) ;
                    int codigo = br.read();//lee el primer caracter
                    char caracter;
           //mientras el código no sea -1 (EOF) continuo leyendo
                while (codigo != -1)
                {
                      caracter = (char) codigo; //casting
                      System.out.print(caracter);
                      codigo = br.read(); //lee un carácter
                }
      }
}
```

10. Operaciones básicas sobre ficheros de acceso secuencial

- Crear un fichero o abrirlo para grabar datos.
- Leer datos del fichero.
- Borrar información de un fichero.
- Copiar datos de un fichero a otro.
- Búsqueda de información en un fichero.
- Cerrar un fichero.

Cuando se trabaja con ficheros de texto se recomienda usar las clases Reader, para entrada o lectura de caracteres, y Writer para salida o escritura de caracteres.
Estas dos clases están optimizadas para trabajar con caracteres y con texto en general, debido a que tienen en cuenta que cada carácter **Unicode** está representado por dos bytes.

Las subclases de Writer y Reader que permiten trabajar con ficheros de texto son:

FileReader, para lectura desde un fichero de texto. Crea un flujo de entrada que trabaja con caracteres en vez de con bytes.
FileWriter, para escritura hacia un fichero de texto. Crea un flujo de salida que trabaja con caracteres en vez de con bytes.
También se puede montar un buffer sobre cualquiera de los flujos que definen estas clases:

BufferedWriter se usa para montar un buffer sobre un flujo de salida de tipo FileWriter.
BufferedReader se usa para montar un buffer sobre un flujo de entrada de tipo FileReader.

Ejemplo

```
//leemos un fichero sin usar buffer
public void leerfichero1(String archivo)
   {int i;
   FileReader fichero = null;
   try {
      fichero = new FileReader(archivo);

      while ((i=fichero.read()) != -1)
               System.out.print((char) i);


}   catch (FileNotFoundException e) {
    System.out.println("Se ha producido un error. El fichero no se encuentra");
     e.printStackTrace();
}catch (IOException ex)
{
   System.out.println(ex.getMessage());
}
   }
```

11. Operaciones básicas sobre ficheros de acceso aleatorio.

Java proporciona una clase **RandomAccessFile** para este tipo de entrada/salida.
Esta clase:

- Permite leer y escribir sobre el fichero, no es necesario dos clases diferentes.
- Necesita que le especifiquemos el modo de acceso al construir un objeto de esta clase: sólo lectura o bien lectura y escritura.
- Posee métodos específicos de desplazamiento como **seek(long posicion)** o **skipBytes(int desplazamiento)** para poder movernos de un registro a otro del fichero, o posicionarnos directamente en una posición concreta del fichero.
- Un archivo de acceso directo tiene sus registros de un tamaño fijo o predeterminado de antemano.

La clase posee dos constructores:

RandomAccessFile(File file, String mode).
RandomAccessFile(String name, String mode).
En el primer caso se pasa un objeto File como primer parámetro, mientras que en el segundo caso es un String. El modo es: "r" si se abre en modo lectura o "rw" si se abre en modo lectura y escritura.

A continuación puedes ver una presentación en la que se muestra cómo abrir y escribir en un fichero de acceso aleatorio. También, en el segundo código descargable, se presenta el código correspondiente a la escritura y localización de registros en ficheros de acceso aleatorio.

Ejemplo:

Vamos a ver un pequeño ejemplo, **Log.java**, que añade una cadena a un fichero existente, lo crea en caso de que no exista.

```
import java.io.IOException;
import java.io.RandomAccessFile;
public class RandomEjemplo {
public static void main( String args[] ) throws IOException {
    RandomAccessFile miRAFile;
    String s = "linea a añadir al final del fichero";
    // Abrimos el fichero de acceso aleatorio
    miRAFile = new RandomAccessFile( "java.log","rw" );
    // Nos vamos al final del fichero
    miRAFile.seek( miRAFile.length() );
    // Incorporamos la cadena al fichero
    miRAFile.writeBytes( s );
    // Cerramos el fichero
    miRAFile.close();
    }
}
```

RECURSOS

https://datos.codeandcoke.com/apuntes:ficheros

https://docs.oracle.com/javase/tutorial/essential/environment/sysprop.html

https://docs.oracle.com/javase/7/docs/api/java/io/File.html
http://puntocomnoesunlenguaje.blogspot.com/2013/05/clase-file-java.html

http://www.jtech.ua.es/j2ee/2002-2003/modulos/xml/apuntes/apuntes3.htm

https://datos.codeandcoke.com/apuntes:ficheros

https://docs.oracle.com/javase/tutorial/essential/environment/sysprop.html
https://docs.oracle.com/javase/7/docs/api/java/io/File.html

http://www.jtech.ua.es/j2ee/2002-2003/modulos/xml/apuntes/apuntes3.htm

https://barcelonageeks.com/diferencia-entre-inputstream-y-outputstream-en-java/

https://w3api.com/Java/OutputStream-java-io/write/

https://javadesdecero.es/avanzado/streams-de-bytes/