|  |
| --- |
| cifp Juan de colonia |
| Acceso a Datos |
| UT1: Manejo de ficheros |
|  |
| **Alexis López Briongos Dam2t** |
| **02/11/2023** |

|  |
| --- |
|  |

*Índice*

[FICHERO 2](#_Toc150358165)

[CLASES JAVA 2](#_Toc150358166)

[FLUJO O STREAM 4](#_Toc150358167)

[ACCESO A FICHEROS 5](#_Toc150358168)

[OPERACIONES SOBRE FICHEROS 6](#_Toc150358169)

[OPERACIONES SOBRE REGISTROS DE UN FICHERO 6](#_Toc150358170)

# FICHERO

¿Qué es un fichero?

* Conjunto de bits (información) almacenados en un dispositivo.

Características:

* No son volátiles.
* Se ubican en directorios o carpetas.
* Dentro de su directorio tienen un nombre (identificador) único.
* Cada fichero suele tener una extensión la cual indica el tipo de datos que contiene.

Compuesto por:

* Registros
  + Campos
* La manera en que se agrupan los datos dentro de un fichero depende del propio diseño del fichero.

# CLASES JAVA

¿Qué es java.io?

* Es la clase que contiene todo lo necesario para tratar ficheros con java

¿Qué es File?

* **File**: es una clase que se utiliza para la manipulación de ficheros.
* **List()**: devuelve un array de strings con los nombres de los ficheros y directorios de un objeto File.
* **ListFiles()** : devuelve un array de files con los ficheros y directorios que existen en ese directorio.
* **getName()**: devuelve el nombre de un fichero o directorio.
* **getPath()**: devuelve la ruta relativa del fichero o directorio que le pasemos.
* **getAbsolutePath()**: devuelve la ruta absoluta del fichero o directorio que le pasemos.
* **exists()**: devuelve un booleano si existe o no un fichero o directorio.
* **canWrite()**: devuelve un booleano si el fichero se puede escribir o no.
* **canRead()**: devuelve un booleano si el fichero se puede leer o no.
* **isFile()**: devuelve un booleano si es fichero o no
* **isDirectory()**: devuelve un booleano si es directorio o no
* **length()**: devuelve un longitud de un fichero en bytes
* **mkdir()**: crea un directorio si no existe y devuelve un booleano si existe o no dicho directorio
* **renameTo(File, “nuevoNombre”)**: renombra el fichero y devuelve booleano si el fichero contiene , o no existe , o no tienes permisos de escritura.
* **delete():** elimina el directorio si está vacío y devuelve un booleano si ha podido hacerlo.
* **createNewFile():** crea un fichero vacío si se puede.
* **getParent():** devuelve el nombre del directorio padre (string) si no devuelve null.

# FLUJO O STREAM

* Para manejar la entrada salida entre una fuente y un destino se utiliza los flujos o streams. Es decir, cualquier programa que necesite obtener información de cualquier fuente necesita abrir un stream y viceversa (enviar).
* La vinculación entre el stream y el dispositivo fuente la realiza el propio sistema de e/s de java (java.io).
* Tipos de streams
  + Streams de bytes (8 bits)
    - Se utilizan para operaciones de e/s de bytes. Es decir, para la lectura y escritura de datos binarios.
    - Todas las clases de streams de bytes descienden de las clases InputStream/OutputStream y cada una de ellas tiene varias subclases dependiendo del dispositivo que se vaya a utilizar.
  + Streams de caracteres (16 bits)
    - Se implementaron porque los streams 8 bits se quedaron cortos.
    - Se utilizan para operaciones de e/s de caracteres y para ello se emplean las clases: 🡺Manejan streams de caracteres UNICODE
      * Reader
      * Writer
* Clases de streams de bytes(8 bits):
  + InputStream: clases que producen entradas de distintas fuentes
    - ByteArrayInputStream permite utilizar un espacio de almacenamiento intermedio (buffer) en memoria.
    - StringBufferInputStream: convierte un string en un InputStream
    - PipedInputStream: almacena en tuberías
    - FilterInputStream: da funcionalidades a otras clases de InputStream
      * LineNumberInputStream
      * DataInputStream
      * BufferedInputStream
      * PushBackInputStream
    - SequenceInputStream: convierte dos o más objetos InputStream en un InputStream único.
    - FileInputStream: realiza un flujo de entrada desde un fichero.
  + OutputStream: clases que producen salidas de distintas fuentes.
    - ByteArrayOutputStream: permite utilizar un espacio de almacenamiento intermedio(buffer) en memoria.
    - PipedOutputStream: se asocia al PipedInputStream y la información que se graba en el PipedOutputStream acaba automáticamente como entrada al PipedInputStream asociado.
    - FilterOutputStream: da funcionalidades a otras clases de OutputStream
      * LineOutputStream
      * BufferedOutputStream
      * PrintStream
    - FileOutputStream: realiza un flujo de salida hacia un fichero.
* Clases de streams de 16 bits
  + Reader
    - BufferedReader (BufferedInputStream)
      * LineNumberReader
    - CharArrayReader (ByteArrayInputStream)
    - InputStreamReader: convierte un InputStream en un Reader
      * FileReader (FileInputStream)
    - FilterReader(FilterInputStream)
      * PushBackReader (PushBackInputStream)
    - PipeReader (PipedInputStream)
    - StringReader (StringBufferedInputStream)
  + Writer
    - BufferedWriter (BufferedOutputStream)
    - CharArrayWriter (ByteArrayOutputStream)
    - OutputStreamWriter
      * FileWriter (FileOutputStream)
    - FilterWriter (FilterOutputStream)
    - PipedWriter (PipedOutputStream)
    - StringWriter (StringOutputStream)
    - PrintWriter (PrintOutputStream)

# ACCESO A FICHEROS

* Existen dos maneras para acceder a ficheros
  + Acceso Secuencial en el cual los datos o registros del fichero se leen y se escriben siempre en orden secuencial (de principio a fin) el nuevo registro ira siempre al final del fichero a parir del último fichero.
    - Se suelen utilizar en procesos por lotes (Batch).
    - Clases para acceder de forma secuencial a los ficheros
      * FileOutputStream
      * FileInputStream
      * FileWriter
      * FileReader
    - Ventajas
      * Son rápidos para acceder a los registros.
      * Aprovechan mejor el espacio.
      * Son sencillos de usar.
    - Inconvenientes
      * No se puede acceder directamente a un registro.
      * Son muy complicados de actualizar.
  + Acceso Aleatorio (ficheros aleatorios) se permite acceder directamente a un dato o registro sin haber pasado antes por los anteriores.
    - Los datos tienen que estar almacenados en registros de tamaño fijo.
    - Clases que se utilizan para acceso aleatorio en ficheros.
      * RandomAccessFile.
    - Los ficheros aleatorios utilizan direcciones relativas, no absolutas.
    - Ventajas
      * Rapidez con la que se accede a un registro.
    - Inconvenientes:
      * Establecer posición de cada registro
      * Se desaprovecha el espacio, puede haber huecos.

# OPERACIONES SOBRE FICHEROS

* Independientemente de cuál sea su forma de acceso, las operaciones que se puedan realizar sobre un fichero son:
  + Creación: de un fichero con un nombre único en el directorio que nos permite acceder a él. Es un proceso que solo se debe hacer una vez.
  + Apertura: es la primera operación que hay que hacer para que un programa pueda operar con el fichero.
  + Lectura: transferir información desde el fichero hacia la memoria (programa).
  + Escritura: transferir información desde la memoria hacia el fichero.
  + Cierre: cuando no se vaya a usar más y suele ser siempre la ultima instrucción del programa.
  + Eliminación

# OPERACIONES SOBRE REGISTROS DE UN FICHERO

* Altas: insertar nuevos registros
  + Si el acceso es secuencial el registro ira al final del fichero.
  + Si el acceso aleatorio el registro irá se aplica la función designada.
* Bajas: eliminar registros existentes:
  + Si el acceso es secuencial el borrado será física.
    - Se emplea algún campo que cambia de valor para indicar que ese registro está borrado lógicamente.
    - Se realizan las con un fichero auxiliar de manera que se leen todos los registros y se copian en otro fichero leyendo todos los registros y obviando el que tiene registro modificado, se elimina el fichero original y se renombra el fichero auxiliar con el nombre del fichero original.
  + Si el acceso es aleatorio el borrado será lógica del fichero.
    - Se emplea algún campo que cambia de valor para indicar que ese registro está borrado lógicamente.
* Modificaciones: cambiar valores de los registros.
  + Si el acceso es secuencial la modificación será lógica.
    - Se emplea algún campo que cambia de valor para indicar que ese registro está borrado lógicamente.
    - Se realizan con un fichero auxiliar de manera que se leen todos los registros y se copian en otro fichero leyendo todos los registros y añadiendo el registro que tiene el campo modificado pero con el valor modificado, se elimina el fichero original y se renombra el fichero auxiliar con el nombre del fichero original.
  + Si el acceso es aleatorio la modificación mediante la clave.
    - Con el campo clave buscar el registro que queremos modificar y cambiar el valor del registro.
* Consultas: accedemos y obtenemos información de los registros:
  + Si el acceso es secuencial consulta será desde el primero hasta el registro a encontrar.
  + Si el acceso es aleatorio se recogerá el valor de los registros por la clave introducida.

# CLASES PARA LA GESTIÓN DE STREAM DE DATOS

* Ficheros de texto manejan caracteres alfanuméricos (ASCII, UTF-8).
* En ellos se emplean las clases FileReader y FileWriter.
* Cada vez que se lee o escribe en un fichero de este tipo, hay que hacerlo en un bloque try catch dado que pueden generar excepciones.
  + En el caso de lectura 🡺 FIleNotFoundException
  + En el caso de escritura 🡺 IOException
* FILEREADER métodos: estos métodos devolverán siempre los caracteres leídos o -1 si se ha llegado a final del fichero o si no se puede leer más (no hay más datos, corruptos).
  + Int Read(): lee un carácter y lo devuelve
  + Int Read(char[] buf): leer hasta buf.length caracteres de una matriz de caracteres buf desde el principio.
  + Int Read(char[]buf, int despl, int n): lee hasta n caracteres de la matriz buf, comenzando por buf[despl] indicando el despl (desplazamiento) .
  + Pasos con los que trabajaremos con los ficheros.
    - Creamos fichero, creamos el flujo de entrada con el FileReader, realizamos operaciones de lectura, cerramos el fichero con el método close().
* FILEWRITER métodos:
  + Void write(int c): escribi
  + Void write(char[]buf)
  + Void write(char[]buf, int despl, int n): escribir n caracteres de la matriz buf comenzando desde int despl.
  + Void Write(String str): escribir una cadena de caracteres
  + Void append(Char c): añadir el carácter al final del fichero
* La clase FileReader no tenia métodos para leen líneas completas pero BufferedReader si con el método readLine();
* BUFFEREDREADER métodos:
  + readLine(): lee líneas completas
  + read(): lee caracteres