TEMA FLUJOS DE DATOS. FICHEROS.

1 FICHEROS

Con las estructuras de almacenamiento vistas hasta ahora sólo se podían guardar los datos durante la ejecución del programa, para que los datos puedan perdurar de unas ejecuciones a otras, la solución es almacenarlos en un fichero en disco.

Fichero: se define como una colección de información, que almacenamos en un soporte físico (disco magnético, CD…), para poder manipularla en cualquier momento, esta información se colecciona como un conjunto de registros, formados por campos de un tipo predefinido ó definidos por el usuario, referentes a un mismo objeto ó sujeto. En **POO** hablaremos de objetos más que de registros y de atributos más que de campos.

2 FLUJOS: determinan la comunicación entre un programa y el origen ó destino de cierta información, es decir, es un objeto que hace de intermediario entre el origen y destino de la información.

3 TIPOS de FICHEROS:

Atendiendo a la forma en que se graban los registros hay 2 tipos: Secuenciales y Aleatorios.

3.1_SECUENCIALES: Pueden almacenar registros de cualquier longitud, incluso de un solo byte, cuando la información se escribe registro a registro, estos son colocados uno a continuación de otro, y cuando se lee, se empieza por el primero y se continúa con el siguiente hasta el final, este tipo se utiliza con ficheros de texto, los cuales no son adecuados para almacenar grandes cantidades de números, porque cada número se almacena como una secuencia de bytes ejemplo: Un nº de 9 dígitos ocuparía 18 bytes.

3.1.2_Tipos de Flujos :

a) Flujos de bytes:

Los datos son escritos ó leídos byte a byte. Los flujos utilizados son:

PARA GRABAR BYTES DESDE MEMORIA AL FICHERO:

FileOutputStream:

Establece un flujo para escribir bytes en un fichero.

CONSTRUCTORES:

1- FileOutputStream (String nombre). Crea el fichero, si ya existiera, borraría su contenido.

2- FileOutputStream (String nombre,boolean añadir). Crea el fichero y en caso de que ya existiera, añade datos a partir del último registro grabado, el identificador añadir, tendrá que estar a "true".

3- FileOutputStream (File fichero).

4- FileOutputStream (File nombre,boolean añadir).

Ejemplo:

1-FileOutputStream fs = new FileOutputStream(“texto.txt”);

Esta instrucción crea el fichero con nombre texto.txt, si ya existiera borraría su contenido.

2-FileOutputStream fs = new FileOutputStream(“texto.txt”,true);

Esta instrucción si no existiera el fichero, lo crea y si ya existe añade datos al final.

3- FileOutputStream fs = new FileOutputStream(File nombre);

La clase File, incorpora nuevos métodos que nos permitirán comprobar si el fichero ya existe antes de crearlo, evitando que se pueda perder la información.

Ejemplo:

```
public class Ejemplo :EscribirBytesFichero
{
   public static void main(String[] args)
      {
        BufferedReader lee = new BufferedReader(new InputStreamReader(System.in));
      FileOutputStream fs=null;
      byte [] buffer = new byte [81];    // Define una matríz BUffer
      int nbytes;

      try{
          fs = new FileOutputStream("Fichero_1.txt");//Para crear y borrar datos anteriores.
            fs = new FileOutputStream("Fichero_1.txt",true)// para crear y añadir al final.
         System.out.println("Escribir texto para almacenar en el fichero"); nbytes=System.in.read(buffer);//leemos una línea de texto desde teclado
         fs.write(buffer, 0, nbytes-1);//grabamos los bytes en fichero.
         }
      catch(IOException e)
         {
        System.out.println("Error."+e.toString());
          }
         finally
                {
                   try
                         {
```

```
                    if (fs!=null)
                     fs.close(); //cerramos fichero
                     }catch(IOException e)
                       {
                        System.out.println("Error."+e.toString());
                        }
                }
   }
}
```

ESTE EJEMPLO ESTÁ HECHO EN CARPETA EJEMPLOS.

PARA LEER BYTES DESDE EL FICHERO Y LLEVARLOS A MEMORIA

FileInputStream:

Permite leer bytes desde un fichero.

CONSTRUCTORES:

1- FileInputStream (String nombre)

2- FileIntputStream (File fichero)

Ejemplo:

```
public class  LeerBytesFichero {
   public static void main(String[] args)
 {
 BufferedReader lee = new BufferedReader(new
InputStreamReader(System.in));
```

```
FileInputStream fe=null;
byte [] buffer = new byte [81];
int nbytes;

try{
   fe = new FileInputStream(“Fichero_1.txt"); //Crear flujo desde el fichero
   nbytes=fe.read(buffer, 0, 81);//leer bytes desde el fichero
   String str=new String(buffer, 0, nbytes);//grabar los bytes en la cadena str.
   System.out.println(str);
}catch(IOException e)
 {
  System.out.println("Error."+e.toString());
 }
  finally{
  try{
     if (fe!=null)
        fe.close();
     }catch(IOException e)
       {
     System.out.println("Error."+e.toString());
       }
```

```
      }
   }
}
```

Ejemplo en carpeta Ejemplos.

Clase File:

Un objeto de esta clase, representa el nombre de un fichero ó de un directorio que puede existir en el sistema de ficheros de la máquina.

CONSTRUCTORES

1- Public File(String ruta_completa);

File fichero=new File("proyecto\\texto.txt");

2- Public File(String ruta,String nombre);

File fichero=new File("proyecto","texto.txt");

3- Public File(File ruta,String nombre);

MÉTODOS:

Tiene varios métodos entre los cuales destaca *exists*(), que es muy útil para comprobar si el fichero ya existe, en cuyo caso devuelve el valor "true".

Length()=Devuelve el tamaño del fichero.

Delete()=borra el fichero.

RenameTo()=Renombra el fichero especificado por el objeto File.

toString()=devuelve la ruta especificada cuando se creó el objeto File.

AHORA VAMOS A VER COMO SE TRABAJA CON DATOS PRIMITIVOS

b) FLUJOS DE DATOS:

Flujo de salida:

Se usan para escribir en un fichero datos de tipo primitivo: boolean, byte, double, float, long, int y short, para luego recuperarlos como tal.

Constructores:

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

METODOS:

writeBoolean(), escribe un valor de tipo Boolean.

writeByte (), escribe un valor de tipo byte.

writeInteger(), escribe un valor de tipo int.

……..

Así con todos los tipos de datos.

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

Flujo de entrada:

Constructores:

DataIntputStream dis=new DataIntputStream(new FileIntputStream(fichero));

MÉTODOS:

readBoolean()

readByte()

readInt()

readUTF(): devuelve una cadena de caracteres en formato UTF-8.

Ejemplo en carpeta de ejemplos.

C) Flujos de Objetos.SERIACIÓN DE OBJETOS

En poo, debemos trabajar con objetos y no con campos aislados, es decir trataremos los campos referentes a una misma entidad ó persona, como si fuesen atributos de un determinado objeto que pertenezca a una clase, lo que nos conducirá a leer y a escribir objetos hacia un fichero ó desde un fichero.

La operación de enviar una serie de objetos a un fichero en disco para hacerlos persistentes recibe el nombre de SERIACIÓN y la operación de leer DESERIACIÓN, para hacer estas operaciones de forma automática el paquete java.io proporciona las clases ObjectOutputStream y ObjectInputStream, se trata de convertir el estado de un objeto (los atributos excepto las variables estáticas), incluyendo la clase del objeto y el prototipo de la misma, en una secuencia de bytes y viceversa, por ello:

los flujos ObjectOutputStream y ObjectInputStream deben ser construídos sobre otros flujos (FileOutputStream y FileInputStream) que canalicen esos bytes a y desde el fichero. Es importante saber que para poder seriar los objetos de una clase ésta debe implementar la interfaz Serializable, se trata de una interfaz vacía, sin ningún método, su propósito es identificar clases cuyos objetos se pueden seriar.

Public class Cpersona implements serializable

{//cuerpo};

Ejemplo completo en carpeta ejemplos.

Una vez ya creado el fichero para añadir objetos, debemos crear una clase que implemente el método, writeStreamHeader(), que inactive la

posibilidad de crear una nueva cabecera, cada vez que tengamos que añadir objetos nuevos al fichero.

por ejemplo:

```
public class ObjectOutputStreamSinCabecera extends ObjectOutputStream
        {
   public ObjectOutputStreamSinCabecera(OutputStream out) throws IOException
   {
      super(out);
   }

   @Override
   protected void writeStreamHeader() throws IOException
      {
      this.reset();
       }
}
```

d) SERIAR OBJETOS QUE REFERENCIAN A OTROS OBJETOS

Cuando en un fichero se escribe un objeto, que hace referencia a otros objetos, como por ejemplo un ArrayList, entonces todos los objetos accesibles desde el primero, es decir desde el ArrayList, deben ser escritos en el mismo proceso, para mantener así la relación existente entre todos ellos, este proceso es llevado a cabo automáticamente por el método writeObject, que escribe el objeto especificado (ArrayList), recorriendo sus referencias a otros objetos recursivamente, escribiendo así todos ellos. Este método sobreescribe la dirección del arrayList, por lo cual hay que realizar paso por valor en lo referente a la dirección del array.

Lo mismo ocurre en caso de tener que recuperar un objeto ya grabado en un fichero, que contenga las referencias a otros objetos, el método readObject, recorrerá sus referencias a otros objetos recursivamente, para recuperar todos ellos, manteniendo la relación que existía entre ellos cuando fueron escritos, debiendo implementar la interfaz serializable.

Ejemplo completo en carpeta Ejemplos.

3.2_ALEATORIOS:

Este tipo de ficheros se caracterizan por tener un índice ó una clave que le permite acceder directamente a la posición en que está el registro deseado,sin tener que pasar por los diferentes registros que se encuentran situados antes .Para que esto pueda realizarse el fichero tiene que estar estructurado de una forma determinada.

Java proporciona la clase **RandomAccessFile,** que permite que este tipo de ficheros puedan realizar un acceso directo al registro buscado,los ficheros que maneja son **binarios**, tienen un **índice,** por medio del cual se puede acceder directamente a distintos puntos del fichero y **la longitud de todos los registros** del mismo fichero tiene que ser igual. Esta clase **no permite seriar objetos** los datos deben guardarse uno a uno.

Esta clase tiene dos **constructores**:

RandomAccessFile fichero=new RandomAccessFile(String nombre,String modo)

RandomAccessFile fichero=new RandomAccessFile(File objeto,String modo)

Ej:RandomAccessFile fichero=new RandomAccessFile("Alumnos.txt","rw");

MODOS DE APERTURA:

Solo lectura_____r

LecturaEscritura_____rw

Ejemplo de escritura de un objeto:

```
RandomAccessFile raf=new RandomAccessFile(fichero,"rw");


        raf.seek(pos*Alumno.maxlongitud);


        raf.writeInt(al.getClave());
        raf.writeUTF(al.getCurso());
        raf.writeUTF(al.getNombre());
        raf.writeFloat(al.getNota());


        System.out.println("Se ha grabado el registro");


    raf.close();
  }
```

Esta clase implementa las **interfaces** **DataInputStream** **y** **DataOutputStream** que le permiten hacer uso de los métodos **read() y write()**, pero además tiene otros métodos que le permiten moverse por el fichero:

MÉTODOS:

1- **getFilePointer():** Este método **devuelve la posición actual en bytes** del puntero L/E, su prototipo es **: public long getFilePointer() throws** IOException y marca siempre la posición donde se iniciará la siguiente operación lectura ó de escritura sobre el fichero.

2- **Length()**: este método **devuelve la longitud** del fichero en bytes.Su prototio es: **public long length() throws IOException.**

3- **Seek():** este método **mueve el puntero L/E** a una nueva localización desplazada por pos bytes desde el principio del fichero,su prototipo es

Public void seek(long pos) throws IOException.

PROCEDIMIENTO:

En este tipo de ficheros necesitaremos crear siempre dentro **de la clase** del **objeto a tratar un método que calcule el tamaño real de cada registro**, lo cual realizaremos multiplicando por dos el tamaño de cada String y sumando los bytes respectivos a los distintos campos que sean numéricos

ej:

8 bytes si es un long,4 si es int etc….

Ej**:tamaño()** .

Además es necesario fijar un tamaño máximo para cada registro.

Ej**:tamañoReg=140**

con el fín de controlar que el tamaño real de cada uno no exceda del tamaño máximo fijado, y así tb lograremos que todos los registros tengan la misma longitud, lo cual es imprescindible para la localización directa de los distintos registros, por lo cual en todos los métodos que sean: **escribirRegistro(),añadirRegistro(),leerRegistro(),buscarRegistro**(),etc..necesitaremos colocar el puntero L/E donde sea necesario para realizar la operación correspondiente, lo cual haremos siempre con la instrucción: **fes.seek (i * tamañoReg);**

También es necesario declarar una variable de tipo int ó long que contendrá el nº de registros del fichero, nregs, y se podrá calcular por medio de la siguiente fórmula:

Int nregs=(int) Math.ceil (fes.length()/tamañoReg).

Finalmente para borrar fisicamente los registros que se desean dar de baja:

- hay que crear un fichero temporal para grabar en él los registros que siguen de alta,
- luego se borra con el método delete() el fichero inicial
- y a continuación se renombra con el método rename() el fichero temporal actualizado.

Ejemplo completo en carpeta Ejemplos.