CLASE RANDOMACCESSFILE

Son ficheros de acceso directo y no secuenciales como estábamos utilizando hasta ahora.
Esta clase permite leer y escribir en los archivos el registro que nosotros le indiquemos sin tener que leer los registros guardados en posiciones físicas anteriores a él. Al abrir el fichero debemos no solo indicar el nombre del archivo o el FILE sino el modo de apertura.

```
RandomAccessFile archivo = new RandomAccessFile( string, modo);
RandomAccessFile archivo = new RandomAccessFile( FILE, modo);
```

Los modos de apertura son dos y deben ir entre comillas dobles. Son:

- r, modo de lectura.
- rw, modo de lectura y escritura.

Las excepciones que pueden aparecer son:

- ✓ **EOFException**.
- ✓ **FileNotFoundException**.
- ✓ **IOException**.

Algunos de los métodos que tiene esta clase son:

- **seek(long pos)**, permite colocarse en una posición concreta, contada en bytes, en el archivo. Lo que se coloca es el puntero de acceso que es la señal que marca la posición a leer o escribir.
- **long getFilePointer( )**, posición actual del puntero de acceso.
- **long length( )**, devuelve el tamaño del archivo.
- **Int skipbytes(int d)**, desplaza el puntero del fichero las posiciones en bytes que le estamos indicando.

Para leer o escribir utilizamos los métodos de las clases **DataInputStream** y **DataOutputStream**.

- Métodos de lectura: **readBoolean, readByte, readChar, readInt, readDouble, readFloat, readUTF, readLine**.
- Métodos de escritura: **writeBoolean, writeByte, writeBytes, writeChar, writeChars writeInt, writeDouble, writeFloat, writeUTF, writeLine**.

Aunque el fichero esta creado como directo si queremos leer todos los registros podemos leerlo de forma secuencial, con la clase **DataInputStream**.

Tabla con el número de bytes que se utilizar por el tipo de datos simples.

TIPO	BYTE	CHAR	SHORT	INT	LONG	FLOAT	DOUBLE
Nº BYTES	1	2	2	4	8	4	8

Algunas consideraciones a tener en cuenta a la hora de trabajar con los ficheros directos:

1. Los registros deben ser de longitud fija, por lo que debemos leer/escribir siempre el número de bytes correspondientes de cada registro.

2. Podemos tener posiciones de registros en los que no hemos grabado nada.

3. Antes de grabar en el fichero, debemos comprobar que no está ocupada esas posiciones por otro registro ya que perderemos sus datos. Por lo que se posiciona uno, lee lo que hay y vuelva a posicionarse para escribir si está vacío o lo sustituimos por otro valor. Puede darse el caso de colisiones (dos registros tienen la misma ubicación en el archivo), en ese caso deberíamos guardar el registro al final del fichero en posiciones consecutivas (llamada zona de excedentes).

4. Si vamos a procesar todo el fichero antes de realizar la operación deseada sobre el registro, comprobar que tiene información. Ya que puede que no tenga nada grabado en dicha posición.

5. Podemos preparar el fichero antes de grabar los datos correctos. Enviando un registro nulo a todas las posiciones que puede ocupar el fichero. De esta forma sabremos qué valor tenemos en los campos o en alguno de ellos si no ha sido grabado.

6. Para salir del fichero podemos buscar una posición inexistente.

7. El formato writeUTF, guarda dos bytes más para para saber cuántos bytes tiene que leer.

```
import java.io.*;
import java.util.*;
/* Ejemplo de fichero directo. Guardamos los números de la bonoloto y su frecuencia
  Lo grabamos como directo y luego hacemos una lectura directa.
  Pero también podemos hacer una lectura secuencial con DataInputStream.
                                        directo.java
*/
class directo
{
  public static void Generar(int [] array)
  {
    int ind, rep, num;
    for(rep=1;rep<201;rep++)
    {
      num = (int)((Math.random()*49)+1);
     array[num-1] += 1;
    }
  }

  public static void Grabar(int [] array) throws IOException, EOFException
  {
   RandomAccessFile elemento = new RandomAccessFile("bonoloto.dir","rw");
   int i;
   for(i=0;i<49;i++)
   {
     elemento.writeInt(i+1);
     elemento.writeInt(array[i]);
   }
   elemento.close();
  }

  public static void Visualizar() throws IOException, EOFException
  {
   FileInputStream lectura = new FileInputStream("bonoloto.dir");
   DataInputStream elemento = new DataInputStream(lectura);
   boolean salir = true;
   int numero, repeticion, cnt=0;
   try{
       do{
       numero = elemento.readInt();
       repeticion = elemento.readInt();
       System.out.format("%5d.....%5d",numero, repeticion);
       cnt++;
       if(cnt%4==0)
            System.out.print("\r\n");  // salto de línea
    }while(salir);
      }
   catch(IOException e){
        System.out.println("");// para anular el error de fin de fichero                }
   finally  {          elemento.close();       }
  }
  public static void BuscarRepeticion() throws IOException, EOFException
```

```
  {
    Scanner dato = new Scanner(System.in);
    RandomAccessFile elemento = new RandomAccessFile("bonoloto.dir","r");
    int numero, repeticion;
    long  valor;
   try
    {
       do{
            System.out.println("NUMERO A BUSCAR ");
        valor = dato.nextLong();
        elemento.seek((valor-1)*8);// 8 bytes por registro.
        numero = elemento.readInt();
        repeticion = elemento.readInt();
        System.out.format("%5d.....%5d\n",numero, repeticion);
          }while(repeticion !=-1);
    }
   catch( IOException e)
    { elemento.close(); }   }

  public static void main(String [] args) throws IOException
  {
    int [] tabla = new int[49];
      Generar(tabla);
      Grabar(tabla);
      Visualizar();
      BuscarRepeticion();
  }
}
```