UNIDAD 1

Ficheros de acceso aleatorio

1 FORMAS DE ACCESO A UN FICHERO.

Hay dos formas de acceso a la información almacenada en un fichero: acceso secuencial y acceso directo o aleatorio:

Acceso secuencial	- Los datos o registros se leen y se escriben en orden. - Para acceder a un dato, o a un reg. Hay que leer los anteriores. - Se añaden datos o reg a partir del último. Para acceder a un fichero de forma secuencial utilizaremos las clases: - **FileInputStream y FileOutputStream,** para el acceso binario (byte streams) - **FileReader y FileWriter**, para el acceso a caracteres (texto). Utilizaremos estas clases para tratar los ficheros de texto.
Acceso directo o aleatorio	- Permite acceder directamente a un dato o registro sin necesidad de leer los anteriores. - Los datos se almacenan en reg. de tamaños conocidos. Para el acceso aleatorio se utiliza la clase **RandomAccessFile**

Java dispone de la clase **RandomAccessFile** que dispone de métodos para acceder al contenido de un fichero binario de forma aleatoria y para posicionarnos en una posición concreta del mismo.

Disponemos de dos constructores para crear el fichero de acceso aleatorio, estos pueden lanzar la excecepción ***FileNotFoundException***:

- ***RandomAccessFile(String nombrefichero, String modoAcceso):*** Escribiendo el nombre del fichero incluido el path.
- ***RandomAccessFile(File objetoFile, String modoAcceso):*** Con un objeto **File** asociado a un fichero.

El argumento *modoAcceso* puede tener dos valores:

```
File fichero = new File(".\\AleatorioEmple.dat");
//declara el fichero de acceso aleatorio
RandomAccessFile file = new RandomAccessFile(fichero, "rw");
```

Modo de acceso	Significado
r	Abre el fichero en modo de solo lectura. El fichero debe existir. Una operación de escritura en este fichero lanzará la excepción ***IOException***.
rw	Abre el fichero en modo lectura y escritura. Si el fichero no existe se crea.

Una vez abierto el fichero pueden usarse los métodos *read()* y *write()* de las clases **DataInputStream** y **DataOutputStream**.

Métodos para lectura DataInputStream *Tipos primitivos*	**Métodos para escritura DataOutputStream** *Tipos primitivos*

`boolean readBoolean();`
`byte readByte();`
`int readUnsignedByte();`
`int readUnsignedShort();`
`short readShort();`
`char readChar();`
`int readInt();`
`long readLong();`
`float readFloat();`
`double readDouble();`
`String readUTF();`
	`void writeBoolean(boolean v);`
	`void writeByte(int v);`
	`void writeBytes(String s);`
	`void writeShort(int v);`
	`void writeChars(String s);`
	`void writeChar(int v);`
	`void writeInt(int v);`
	`void writeLong(long v);`
	`void writeFloat(float v);`
	`void writeDouble(double v);`
	`void writeUTF(String str);`

La clase **RandomAccessFile** maneja un *puntero* que indica la posición actual en el fichero.
Cuando el fichero se crea el puntero al fichero se coloca en 0, apuntando al principio del mismo.
Las sucesivas llamadas a los métodos *read()* y *write()* ajustan el puntero según la cantidad de bytes leídos o escritos.

Los métodos más importantes son:

Método	Función
long getFilePointer()	Devuelve la posición actual del puntero del fichero.
void seek(long posicion)	Coloca el puntero del fichero en una posición determinada desde el comienzo del mismo.
long length()	Devuelve el tamaño del fichero en bytes. La posición *length()* marca el final del fichero.
int skipBytes(int desplazamiento)	Desplaza el puntero desde la posición actual el número de bytes indicados en *desplazamiento*.

El ejemplo que se muestra a continuación inserta datos de empleados en un fichero aleatorio.
Los datos a insertar: el apellido, departamento y salario, se obtienen de varios arrays que se llenan en el programa, los datos se van introduciendo de forma secuencial (no se usará el método *seek()*).
Por cada empleado también se insertará un identificador que coincidirá con el índice +1 con el que se recorren los arrays. La longitud del registro de cada empleado es la misma (36 bytes) y los tipos que se insertan y su tamaño en bytes es el siguiente:

- Se inserta en primer lugar un entero, que es el identificador, ocupa 4 bytes.
- A continuación, una cadena de 10 caracteres, es el apellido, cada carácter Unicode ocupa 2 bytes, por tanto, el apellido ocupa 20 bytes.
- Un tipo entero, que es el departamento, ocupa 4 bytes.
- Un tipo *Double* que es el salario, ocupa 8 bytes.

Longitud de los tipos primitivos de datos

int (4 bytes) short (2 bytes) byte y Boolean (1 byte)	long (8 bytes) char (2 bytes) float (4 bytes) double (8 bytes)

OPERACIONES

EJEMPLO CREAR REGISTROS:

El fichero se abre en modo **"rw"** para lectura y escritura.
Para la prueba los datos a insertar los creamos en arrays, insertamos 7 registros.
Se insertará de forma secuencial, el primer registro será FERNANDEZ con el número de empleado 1y el último REY con el número de empleado 7. Este será luego el identificador de registro.
Para esta inserción no se utiliza el método de acceso aleatorio.
El código es el siguiente:

```
import java.io.*;
public class EscribirFichAleatorio {
  public static void main(String[] args) throws IOException {
    File fichero = new File(".\\AleatorioEmple.dat");

//declara el fichero de acceso aleatorio
    RandomAccessFile file = new RandomAccessFile(fichero, "rw");

    //arrays con los datos
    String apellido[] = {"FERNANDEZ","GIL","LOPEZ","RAMOS",
                         "SEVILLA","CASILLA", "REY"}; //apellidos
    int dep[] = {10, 20, 10, 10, 30, 30, 20};       //departamentos
    Double salario[]={1000.45, 2400.60, 3000.0, 1500.56,
                      2200.0, 1435.87, 2000.0};//salarios

    StringBuffer buffer = null;//buffer para almacenar apellido

 int n=apellido.length;//numero de elementos del array
```

```
   for (int i=0;i<n; i++){ //recorro los arrays
      file.writeInt(i+1); //uso i+1 para identificar empleado

      buffer = new StringBuffer( apellido[i] );
      buffer.setLength(10); //10 caracteres para el apellido
      file.writeChars(buffer.toString());//insertar apellido

      file.writeInt(dep[i]);       //insertar departamento
      file.writeDouble(salario[i]);//insertar salario
   }
   file.close();  //cerrar fichero
   }
}
```

Habremos creado un fichero así: 36 Bytes por registro (4 + 20 + 4 + 8)
Formula = (código identificador -1) * long registro

(1-1) * 36 = 0
(4-1) * 36 = 108

Identificador (4bytes)	Apellido(20 bytes)	Departamento (4bytes)	Salario (8 bytes)
POSICION 0 1	POSICION 4 FERNANDEZ	POSICION 24 10	POSICION 28 1000.45
POSICION 36 2	POSICION 40 GIL	POSICION 60 20	POSICION 64 2400.60
POSICION 72 3	POSICION 76 LOPEZ	POSICION 96 10	POSICION 100 3000.0
POSICION 108 4	RAMOS	10	POSICION 136 1500.56
POSICION 144 5	SEVILLA	30	POSICION 172 1435.87
POSICION 180 6	CASILLA	30	POSICION 208 2200.0
POSICION 216 7	REY	20	POSICION 244 2000.0

POSICION 252
Fin de fichero

EJEMPLO VISUALIZAR, RECORRER EL FICHERO

El siguiente ejemplo toma el fichero anterior y visualiza todos los registros.
Utilizamos el posicionamiento, aunque la lectura podría ser secuencial, ya que estos los hemos grabado de forma secuencial y sabemos los que hay.
El posicionamiento para empezar a recorrer los registros empieza en 0, para recuperar los siguientes registros hay que sumar 36 (tamaño del registro) a la variable utilizada para el posicionamiento:

```
import java.io.*;
public class LeerFichAleatorio {
  public static void main(String[] args) throws IOException {
   File fichero = new File(".\\AleatorioEmple.dat");
   //declara el fichero de acceso aleatorio
   RandomAccessFile file = new RandomAccessFile(fichero, "r");
   //
   int  id, dep, posicion;
   Double salario;
   char apellido[] = new char[10], aux;

   posicion=0;  //para situarnos al principio
```

```
   for(;;){  //recorro el fichero
      file.seek(posicion); //nos posicionamos en posicion
      id=file.readInt();   // obtengo id de empleado
      for (int i = 0; i < apellido.length; i++) {
         aux = file.readChar();//recorro uno a uno los caracteres del apellido
         apellido[i] = aux;    //los voy guardando en el array
      }
      String apellidoS= new String(apellido);//convierto a String el array
      dep=file.readInt();//obtengo dep
      salario=file.readDouble();  //obtengo salario

      System.out.println("ID: " + id + ", Apellido: "+  apellidoS +
                         ", Departamento: "+dep + ", Salario: " + salario);
      posicion= posicion + 36; // me posiciono para el sig empleado
                               //Cada empleado ocupa 36 bytes (4+20+4+8)
      //Si he recorrido todos los bytes salgo del for
      if (file.getFilePointer()==file.length())break;

   }//fin bucle for
   file.close();  //cerrar fichero
   }
}
```

La ejecución muestra la siguiente salida:

```
ID: 1, Apellido: FERNANDEZ , Departamento: 10, Salario: 1000.45
ID: 2, Apellido: GIL       , Departamento: 20, Salario: 2400.6
ID: 3, Apellido: LOPEZ     , Departamento: 10, Salario: 3000.0
ID: 4, Apellido: RAMOS     , Departamento: 10, Salario: 1500.56
ID: 5, Apellido: SEVILLA   , Departamento: 30, Salario: 2200.0
ID: 6, Apellido: CASILLA   , Departamento: 30, Salario: 1435.87
ID: 7, Apellido: REY       , Departamento: 20, Salario: 2000.0
```

Utilizando writeUTF y readUTF.

La cadena del apellido no se guarda con longitud fija, con lo que al grabar lo hacemos con el posicionamiento, para que controle la longitud por cada registro.
Y para leer el bucle termina cuando la posición de lectura sobrepase la longitud en bytes del fichero.

```
public static void crearUTF() throws IOException {
       File fichero = new File(".\\AleatorioEmpleUTF.dat");
       RandomAccessFile file = new RandomAccessFile(fichero, "rw");
       String apellido[] = { "FERNANDEZ","GIL","LOPEZ","RAMOS","SEVILLA","CASILLA","REY" };
       int dep[] = { 10, 20, 10, 10, 30, 30, 20 };
       Double salario[] = { 1000.45, 2400.60, 3000.0, 1500.56, 2200.0, 1435.87, 2000.0 };

       int n = apellido.length;
       int posicion = 0;
       for (int i = 0; i < n; i++) {
              file.seek(posicion);
              file.writeInt(i + 1);
              file.writeUTF(apellido[i]);
              file.writeInt(dep[i]);
              file.writeDouble(salario[i]);
              posicion = posicion + 36;
       }
       file.close();
}
```

```
public static void leerUTF() throws IOException {
	File fichero = new File(".\\AleatorioEmpleUTF.dat");
	RandomAccessFile file = new RandomAccessFile(fichero, "r");
	int id, dep, posicion = 0;
	Double salario;

	for (;;) {
		file.seek(posicion);
		id = file.readInt();
		String apellidoS = file.readUTF();
		dep = file.readInt();
		salario = file.readDouble();
		System.out.println("ID: " + id + ", Apellido: " + apellidoS
			+ ", Departamento: " + dep + ", Salario: " + salario);
		System.out.println("Posicion=" + posicion + ", filepointer= "
			+ file.getFilePointer() + ",  file leng: "+ file.length());
		posicion = posicion + 36;
		if (posicion >= file.length())
			break;

	}
	file.close();
}
```

Resultado de la ejecución

```
ID: 1, Apellido: FERNANDEZ, Departamento: 10, Salario: 1000.45
Posicion=0, filepointer= 27,  file leng: 237
ID: 2, Apellido: GIL, Departamento: 20, Salario: 2400.6
Posicion=36, filepointer= 57,  file leng: 237
ID: 3, Apellido: LOPEZ, Departamento: 10, Salario: 3000.0
Posicion=72, filepointer= 95,  file leng: 237
ID: 4, Apellido: RAMOS, Departamento: 10, Salario: 1500.56
Posicion=108, filepointer= 131,  file leng: 237
ID: 5, Apellido: SEVILLA, Departamento: 30, Salario: 2200.0
Posicion=144, filepointer= 169,  file leng: 237
ID: 6, Apellido: CASILLA, Departamento: 30, Salario: 1435.87
Posicion=180, filepointer= 205,  file leng: 237
ID: 7, Apellido: REY, Departamento: 20, Salario: 2000.0
Posicion=216, filepointer= 237,  file leng: 237
```

Los problemas los podemos tener cuando necesitemos modificar datos de un registro, pues como las cadenas no tienen longitud fija, tendremos que utilizar la función .getFilePointer, para posicionar el puntero después de leer la cadena.

EJEMPLO CONSULTAR

Para consultar un empleado determinado no es necesario recorrer todos los registros del fichero. Accedemos por su identificador (el identificador del empleado nos indicaba la posición a ocupar, la 1, la 2, la 3, …..)

Conociendo su identificador, aplicamos la fórmula y podemos acceder a la posición que ocupa dentro del fichero y obtener sus datos.

Por ejemplo, supongamos que se desean obtener los datos del empleado con identificador 5, para calcular la posición hemos de tener en cuenta los bytes que ocupa cada registro (36).

La fórmula es: posición = (identificador - 1) * 36 = (5-1) * 36=144

El primer empleado estaria en la posición 0 del fichero. El segundo en la posición 36. El tercero en la posición 72. El cuarto en la 108, y así sucesivamente.

Para saber si no existe el identificador del empleado preguntaremos si **la posicion >= file.length(),** si es mayor es que no se ha creado el registro.

```
int identificador = 5;
posicion = (identificador - 1 ) * 36; //calculo donde empieza el registro
if(posicion >= file.length())
    System.out.println("ID: " + registro + ", NO EXISTE EMPLEADO...");
 else{
      file.seek(posicion);//nos posicionamos
      id=file.readInt(); // obtengo id de empleado
      //obtener resto de los datos, como en el ejemplo anterior
 }
```

AÑADIR REGISTROS A PARTIR DEL ÚLTIMO

Para añadir un registro al final de forma secuencial, simplemente hemos de posicionar el puntero del fichero al final del mismo, aunque lo normal es hacerlo de forma aleatoria:

```
long posicion= file.length() ;
file.seek(posicion);
```

INSERTAR ALEATORIAMENTE

Para insertar aleatoriamente un nuevo registro aplicamos la función de posicionamiento al identificador para calcular la posición.
El siguiente ejemplo inserta un empleado con identificador 20, se ha de calcular la posición donde irá el registro dentro del fichero (identificador -1) * 36 bytes. Una vez insertado se habrán generado muchos huecos. Prueba a ejecutar ***LeerFichAleatorio*** para ver los huecos generados.

```
StringBuffer buffer = null; //bufer para almacenar apellido
String apellido="GONZALEZ"; //apellido a insertar
Double salario=1230.87; //salario
int id=20;   //id del empleado
int dep=10;  //dep del empleado

long posicion = (id -1 ) * 36; //calculamos la posicion

file.seek(posicion); //nos posicionamos
file.writeInt(id); //se escribe id
buffer = new StringBuffer( apellido);
buffer.setLength(10); //10 caracteres para el apellido
file.writeChars(buffer.toString());//insertar apellido
file.writeInt(dep);       //insertar departamento
file.writeDouble(salario);//insertar salario

file.close();  //cerrar fichero
```

Si se producen huecos, es decir insertamos por ejemplo el 10, y teníamos hasta el 7, en el código de departamento se almacenará un 0 en los registros donde no hay información. Por defecto se almacena 0 en los enteros donde no hay información. Con lo que sabremos que hay hueco cuando el campo identificador sea 0 (es necesario obligar a que los campos identificadores sean mayores que 0, de esta manera, un hueco o un borrado lo podemos identificar con un 0 en el campo identificador del registro)

MODIFICAR

Para modificar un registro determinado, accedemos a su posición y efectuamos las modificaciones, es necesario saber el registro a modificar, es decir el identificador del empleado a modificar.
El fichero debe abrirse en modo “rw”.
Por ejemplo, para cambiar el departamento y salario del empleado con identificador 4 escribo:

```
int registro = 4  ;//id a modificar

long posicion = (registro -1 ) * 36; //(4+20+4+8)  modifico salario y dep
posicion=posicion+4+20; //sumo el tamaño de ID+apellido
file.seek(posicion); //nos posicionamos
file.writeInt(40);   //modif departamento a 40
file.writeDouble(4000.87);//modif salario
```

Hay que tener en cuenta a la hora de posicionarse en el registro, si solo se modifica un campo o varios campos del registro.

MODIFICAR ALEATORIAMENTE EN CADENAS UTF

Se utilizará file.getFilePointer() después de haber leído una cadena (por ejemplo apellidoS = **file.readUTF();**)para acceder al siguiente dato grabado. Las cadenas no se almacenan con longitud fija.
Por ejemplo, este método recibe el número de registo a modificar, y el nuevo salario y departamento.

```
public static void modificar(int registro, Double salario, int dep) throws IOException {
        File fichero = new File(".\\AleatorioEmpleUTF.dat");
        RandomAccessFile file = new RandomAccessFile(fichero, "rw");
        long posicion = (registro - 1) * 36; // Posición inicial
        if (posicion<file.length())
        {
        file.seek(posicion); // nos posicionamos al inicio y leemos
        int id = file.readInt();
        if (id == registro) {
                //leemos la cadena, y nos volvemos a posicionar después de leerla
                String apellidoS = file.readUTF();
                file.seek(file.getFilePointer());
                // ya podemos modoficar
                file.writeInt(dep);
                file.writeDouble(salario);
                System.out.println("MODIFICADO :" +registro + " * "+ dep + " * " + salario);

        } else
                System.out.println("NO MODIFICADO NO LOCALIZADO :" + registro);

        }
        else
                System.out.println("TE HAS PASADO :" + registro);
        file.close();

}
```

BORRAR

Para borrra un registro determinado, accedemos a su posición y hacemos un borrado lógico, es decir ponemos a 0 los numéricos y a espacios las cadena. O simplemente ponemos a 0 el campo ientificador, por si se desea recuperar el resto de información. El espacio permanecerá en el fichero como un hueco.
El fichero debe abrirse en modo "rw".
Por ejemplo para borrar el departamento 4 (lo pongo a 0):

```
int registro = 4  ;//id a modificar

long posicion = (registro -1 ) * 36;
file.seek(posicion); //nos posicionamos
file.writeInt(0);   //Pongo a 0 el identificador (se considera hueco o borrado)
```

EJERCICIO

Hacer una clase para crear y hacer operaciones en un fichero directo.
Desde main se llamará al resto de métodos para probarlos, y se debe de visualizar el mensaje que devuelven los métodos.
La clase debe de contener los siguientes métodos:

El fichero contendrá la siguiente información:
COD_DEP int
NOMBRE Cadena de 15 caracteres
LOCALIDAD Cadena de 15 caracteres
NUME_EMPLEADOS int
MEDIA_SALARIO_DEP float

Hacer que el COD_DEP sea el identificador del registro, además de indicar la posición dentro del fichero.

Métodos a crear:

- ***creafichero***. Método para crear el fichero. Debe recibir el nombre de fichero y crearlo. Si ya existe que no lo cree y si no existe que lo cree. El método devuelve un mensaje indicando si EL FICHERO SE HA CREADO CORRECTAMENTE, o si EL FICHERO YA EXISTE.

- ***consultaregistro***. Método que reciba un código de departamento y devuelva true si existe el departamento, y false si no existe.
  Si al leer un número de departamento devuelve 0, quiere decir que es un hueco, con lo que el registro no existe.
  Si no existe debe de visualizar el mensaje DEPARTAMENTO NO EXISTE en main.
  Si existe visualizar el mensaje DEPARTAMENTO SI EXISTE en main

- ***insertaregistro***. Método que reciba un código de departamento, un nombre, una localidad, un número de epleados, y una media de salario. Insertar de forma aleatoria. El código de departamento debe estar entre 1 y 100.
  El método devolverá un mensaje que diga REGISTRO INSERTADO si se inserta correctamente, ERROR REGISTRO NO INSERTADO si ha habido algún error, o ERROR EL DEPARTAMENTO DEBE ESTAR ENTRE 1 Y 100, si el departamento no está en ese intervalo, o ERROR EL DEPARTAMENTO YA EXISTE NO SE PUEDE INSERTAR, caso de que ya exista.
  Hacer uso del método ***consultarregistro*** para saber si el departamento existe o no.

- ***visualizaregistro***. Método que reciba un código de departamento y visualice sus datos. Acceder de forma aleatoria.
  El método devolverá true si existe el departamento, y false si no existe.
  Hacer uso del método ***consultarregistro*** para saber si el departamento existe o no.
  Caso de existir debe de visualuizar los datos del departamento, en el propio método.
  Si no existe debe de visualizar el mensaje DEPARTAMENTO NO EXISTE. NO SE VISUALIZA en main

- ***modificarregistro***. Crear un método que reciba el código de departamento, la localidad y la media de salario. Y actualice la localidad y la media de salario de ese departamento. Acceder de forma aleatoria. El método devuelve el mensaje REGISTRO MODIFICADO si se modifica correctamente, o REGISTRO NO SE PUEDE MODIFICAR, si no existe el departamento.
  Hacer uso del método ***consultarregistro*** para saber si el departamento existe o no.

- ***borrarregistro***. Crear un método que reciba el código de departamento, y lo borre.
  Para borrar el departamento tiene que existir, hacer uso del método ***consultarregistro.***
  Puede ocurrir, que el registro no exista, que sea un hueco o que exista.
  Sabemos que es un hueco si al leer el número de departamento este vale 0.
  El borrado será lógico, pondremos 0 al número de departamento, espacios a las cadenas y 0 al resto de campos numéricos.