FICHEROS DE ACCESO ALEATORIO

Javier García-Retamero Redondo

¿EN QUÉ CONSISTE?

- Hasta ahora, el acceso a los ficheros se realizaba de manera secuencial. Se escribía el primer byte y a continuación los siguientes.
- En los ficheros de acceso aleatorio se puede acceder a la información contenida en una determinada posición.
- El acceso aleatorio tiene lugar en archivos binarios.

TRABAJANDO CON ARCHIVOS DE ACCESO ALEATORIO

DECLARACIÓN Y MÉTODOS

CONDICIONES

- Para poder trabajar con ficheros aleatorios es imprescindible que:
  - Todos los registros tengan el mismo tamaño
  - Haya un campo clave para poder localizar los registros.

CÁLCULO DE LAS POSICIONES

- Los tamaños de los diferentes tipos vienen determinados en la tabla:

TAMAÑO EN DISCO DE LOS DISTINTOS TIPOS DE DATOS PRIMITIVOS		
byte	ENTERO (-128 a 127)	1 BYTE
short	ENTERO (-32.768 a 32.767)	2 BYTES
int	ENTERO (-2.147.483.648 a 2.147.483.647)	4 BYTES
long	ENTERO (-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807)	8 BYTES
float	REAL	4 BYTES
double	REAL	8 BYTES
char	CARÁCTER	2 BYTES (writeChars) 1 BYTE (writeBytes)

CÁLCULO DE LAS POSICIONES

- Imaginemos que las filas de un fichero queremos que contengan los siguientes campos:

CAMPO	Identificador	Apellido	Departamento	Salario
TIPO	long	10 char	int	double

Apellido sería de 10 bytes si escribiéramos con writeBytes

- Cada registro sería de un tamaño de 36 bytes:

	Identificador	Apellido	Departamento	Salario	POSICIÓN INICIAL	POSICIÓN FINAL	Identificador n
REGISTRO 1	8 bytes	20 bytes	4 bytes	8 bytes	0	39	1
REGISTRO 2	8 bytes	20 bytes	4 bytes	8 bytes	40	79	2
REGISTRO 3	8 bytes	20 bytes	4 bytes	8 bytes	80	119	3
…							
REGISTRO N	8 bytes	20 bytes	4 bytes	8 bytes	(n-1)*40	Posición inicial+40-1	n

DECLARACIÓN

- La clase **RamdomAccessFile** nos permite acceder a un fichero de forma aleatoria.
- Tiene dos constructores:
  - **RamdomAccessFile(** String nombrefichero, **String modoAcceso);**
  - **RamdomAccessFile(** File objetoFile, **String modoAcceso);**
- El modoAcceso puede ser:
  - r: Abre el fichero en modo lectura.
  - rw: Abre el fichero en modo lectura/escritura. Si no existe se crea.

MÉTODOS

- Los métodos más importantes son:

MÉTODO	FUNCIÓN
long **getFilePointer**()	Devuelve la posición actual del puntero del fichero
void **seek**(long posición)	Coloca el puntero del fichero en una posición determinada desde el comienzo.
long **length**()	Devuelve el tamaño del fichero en bytes.
int **skipBytes**(int desplazamiento)	Desplaza el puntero desde la posición actual el numero de bytes indicados.

- Posición inicial cuando se abre un fichero: 0
- Posición final del fichero: length()

TRABAJANDO CON ARCHIVOS DE ACCESO ALEATORIO

ESCRITURA

ESCRITURA

Utilizaremos los mismos métodos de escritura que con **DataOutputStream**:

MÉTODOS PARA ESCRITURA	
void writeBoolean(boolean v);	void writeInt(int v);
void writeByte(int v);	void writeLong(long v);
void writeBytes(String v); (cada carácter 1 byte)	void writeFloat(float v);
void writeShort(int v);	void writeDouble(double v);
void writeChars(String s);	void writeUTF(String str); (cada carácter 1 byte)
void writeChar(int v); (cada carácter **2** bytes)	

ESCRITURA

- **Abrimos el fichero:**

```
File ficheroPrueba = new File("C:\\acceso_a_datos\\tema1\\ejemploprueba.dat");
```

- **Creamos el flujo de escritura:**

```
RamdomAccessFile fichero = new RamdomAccessFile(ficheroPrueba, "rw");
```

- **Escribimos:**

```
fichero.writeInt(33);
StringBuffer buffer = null;
buffer = new StringBuffer("GARCIA");
buffer.setLength(10);
fichero.writeChars(buffer.toString());
fichero.writeDouble(1000.33);
```

OPERACIONES ÚTILES EN LA ESCRITURA

- **Cerrar el stream:**

```
fichero.close();
```

- **Añadir registros a partir del último insertado:**

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

OPERACIONES ÚTILES EN LA ESCRITURA

- **Añadir un registro a partir de su identificador:**

```
……..
StringBuffer buffer = null;
String apellido = "GARCIA";
Double salario = 1300;
int identificador = 15;
int dep = 20;

long posicion = (identificador-1)*40;
fichero.seek(posicion);
fichero.writeInt(identificador);
……..
fichero.close();
```

OPERACIONES ÚTILES EN LA ESCRITURA

- **Modificar un registro a partir de su identificador:**
- Si quisiéramos modificar el departamento y el salario:

CAMPO	Identificador	Apellido	Departamento	Salario
Bytes	8	20	4	8

  - Calculamos la posición de inicio según el identificador:

```
long posicion = (identificador-1)*40;
```

  - Nos saltamos los campos que no hay que modificar:

```
posicion = posicion + 8 +20;
```

  - Saltamos a la posición y modificamos:

```
fichero.seek(posicion);
fichero.writeInt(30);
fichero.writeDouble(2000);
```

TRABAJANDO CON ARCHIVOS DE ACCESO ALEATORIO

LECTURA

LECTURA

Utilizaremos los mismos métodos de lectura que con **DataInputStream**:

MÉTODOS PARA LECTURA	
Boolean readBoolean();	int readInt();
Byte readByte(); (lee un byte)	long readLong();
int readUnsignedByte();	float readFloat();
int readUnsignedShort();	double readDouble()
short readShort();	string readUTF(); (lee un String)
char readChar(); (lee un carácter -2 bytes-)	

LECTURA DE UN REGISTRO CONCRETO

- Para la lectura no necesitamos recorrer todos los registros, necesitaremos:
  - El identificador
  - El tamaño del registro

LECTURA DE UN REGISTRO CONCRETO

- **Obtenemos el identificador:**

```
int identificador = 5;
```

- **Calcular la posición en función del tamaño:**

```
posicion  = (identificador-1)*40;
```

- **Verificar que la posición no sea mayor que la longitud del archivo:**

```
if (posicion >= fichero.length())
  System.out.println("ID: "+ identificador + " no existe empleado…");
```

- **Posicionarnos y leer:**

```
fichero.seek(posicion);
id = fichero.readInt();
```

OPERACIONES ÚTILES EN LA LECTURA

- **Para verificar que hemos llegado al final del archivo:**

```
if (fichero.getFilePointer() == fichero.length())
```

- **Cerrar el stream:**

```
fichero.close();
```