REGISTROS Y ARCHIVOS DE ACCESO SECUENCIAL

Los programas necesitan comunicarse con su entorno, tanto para recoger datos e información que deben procesar, como para devolver los resultados obtenidos.

Los archivos son el medio que facilita un lenguaje para almacenar en forma permanente los datos que se generan dentro de un programa, ya sea por captura, proceso, etc.

La manera de representar estas entradas y salidas en Java es a base de *streams* (flujos de datos). Un *stream* es una conexión entre el programa y la fuente o destino de datos. La información se traslada en serie (un caracter a continuación de otro) a través de esta conexión.

El paquete ***java.io*** contiene las clases predefinidas necesarias para la comunicación del programa con el exterior. Son las jerarquías de clases InputStream y OutputStream, con varias clases derivadas y un conjunto de métodos que permiten manipular los datos.

1 . Lectura de caracteres desde el teclado

Paquete: java.io

```
java.lang.Object
  └ java.io.Reader
      └ java.io.BufferedReader
```

Constructor Summary

BufferedReader(Reader in) Create a buffering character-input stream that uses a default-sized input buffer.
BufferedReader(Reader in, int sz) Create a buffering character-input stream that uses an input buffer of the specified size.

Method Summary

int	**read**()	Read a single character.
String	**readLine**()	Read a line of text.
void	**reset**()	Reset the stream to the most recent mark.

Constructor Detail

BufferedReader

```
public BufferedReader(Reader in)
```

Create a buffering character-input stream that uses a default-sized input buffer.

Parameters:

in - A Reader

Method Detail

read

```
public int read()
          throws IOException
```

Read a single character.

Overrides:

read in class Reader

Ejemplo de uso de un objeto de tipo BufferedReader

```
BufferedReader teclado = new BufferedReader(new InputStreamReader(System.in));
```

Si se ingresa por ejemplo un nombre, y se da *enter*, se podrá almacenar este dato en una variable de la siguiente manera:

```
String nombre = teclado.readLine();
```

El método devuelve un string, por lo tanto, en caso de ser otro tipo de dato, se deberá hacer la conversión:

```
double importe = new Double(teclado.readLine()).doubleValue();
```

También se puede obtener un entero, con el método correspondiente:

```
int nota = teclado.read();
```

2 . Escritura en archivos secuenciales

Se utiliza la clase FileOutputStream, que permite crear, manipular y procesar archivos.

java.lang.Object
  └ java.io.OutputStream
    └ **java.io.FileOutputStream**

El constructor recibe dos parámetros:

```
public FileOutputStream(String name, boolean append)
```

Parameters:

name - the system-dependent file name
append - if `true`, then bytes will be written to the end of the file rather than the beginning

El primer parámetro es toda la ruta o path donde quedará el archivo (separado con doble diagonal \\).
El segundo parámetro es la palabra "true", lo cual indica que el archivo se abre en modo llamado "APPEND", es decir que cada nuevo registro se graba al final del archivo. Si se omite este parámetro, un nuevo registro se sobrescribe sobre el registro anterior.

Otra clase muy utilizada es DataOutputStream, clase derivada de la anterior. Su constructor es:

```
public DataOutputStream(OutputStream salida)
```

Como parámetro se le puede pasar un FileOutputStream, que deriva de OutputStream

La diferencia es que FileOutputStream graba/lee bytes y DataOutputStream graba/lee datos formateados (int, float, double, etc). Algunos de sus métodos son:

Retorno	Nombre-Parámetro	
void	writeChar(int v)	graba un dato de tipo char.
void	writeDouble(double v)	graba un dato de tipo doble.
void	writeInt(int v)	graba un dato de tipo entero.
void	writeUTF(String str)	graba un dato de tipo string.
void	writeLong(long l)	graba un dato de tipo long.
void	close()	cierra el archivo y libera los recursos de sistema asociados

3 . Lectura en archivos secuenciales

Se utilizan las clases FileInputStream y DataInputStream. El constructor de FileInputStream tiene un solo parámetro, que es el path donde se encuentra el archivo a leer. El ciclo de lectura se realiza con un while cuya condición es que hayan bytes disponibles para leer (**método available()).**

java.lang.Object
  └ java.io.InputStream
    └ **java.io.FileInputStream**

Los constructores son:

```
public FileInputStream(String name)

public DataInputStream(InputStream in)
```

Algunos métodos de DataInputStream son:

Retorno	Nombre-Parámetro	
double	readDouble()	lee un dato de tipo doble.
float	readFloat()	lee un dato de tipo float.
int	readInt()	lee un dato de tipo entero.
String	readUTF()	lee un dato de tipo string.
long	readLong()	lee un dato de tipo long.
int	available()	devuelve el numero de bytes que pueden ser leídos del corriente "file input stream"

Ejemplo 1: Código ejemplo de grabación

```
import java.io.*;

public class GrabaSecuencial{
  public static void main(String[] args){
     int LU = 0;
     String nombre = "";
     int edad = 0;
     char opcion = 'S';

     // creando objeto teclado
     BufferedReader teclado = new BufferedReader(new
                                  InputStreamReader(System.in));

     try {
         // Creando un objeto de tipo archivo secuencial
         FileOutputStream archiFOS = new FileOutputStream(
                     "c:\\carpeta\\Alumno.dat",true);
         DataOutputStream archiDOS = new DataOutputStream(archiFOS);
         while(opcion != 'n' && opcion != 'N'){
            System.out.println("ingrese LU: ");
            LU = Integer.parseInt(teclado.readLine());

            System.out.println("ingrese nombre: ");
            nombre=teclado.readLine();

            System.out.println("ingrese edad: ");
            edad = Integer.parseInt(teclado.readLine());
```

```
            //grabando el archivo
            archiDOS.writeInt(LU);
            archiDOS.writeUTF(nombre);
            archiDOS.writeInt(edad);

            System.out.println("Más datos ? (S/N): ");
            opcion=teclado.readLine().charAt(0);
         }
         archiDOS.close();
     } // cierra try
     catch(FileNotFoundException fnfe) {
           System.out.println("Archivo no encontrado");
     }
     catch(IOException ioe){
          System.out.println("Error al grabar");
     }
  } // cierra main
} // cierra clase
```

Nota: Observar que la grabación lleva un try-catch FileNotFound y IOException, que son obligatorios.

Ejemplo 2: Código ejemplo de lectura

```
import java.io.*;

public class LeeSecuencial{
  public static void main(String[] args) {
     int LU=0;
     String nombre=new String("");
     int edad=0;
     try{
         // Creando un objeto de tipo archivo secuencial para leer
         FileInputStream archiFIS=new FileInputStream("c:\\carpeta\\archi1.dat");
         DataInputStream archiDIS = new DataInputStream(archiFIS);

         //leyendo archivo
         while (archiDIS.available()>0){ // mientras haya bytes para leer ...
             LU     = archiDIS.readInt();
             nombre = archiDIS.readUTF();
             edad   = archiDIS.readInt();

             System.out.println(clave+" "+nombre+" "+edad);
         }
         archiDIS.close();
     } // cierra try
     catch(FileNotFoundException fnfe){
            System.out.println("Archivo no encontrado");
     }
     catch (IOException ioe){
            System.out.println("Error al leer");
     }
   } // cierra main
} // cierra clase
```

Búsqueda en archivos secuenciales

En ocasiones, se desea desplegar un único registro de información, habiendo proporcionado un dato de búsqueda (generalmente la clave del registro). En este caso, se debe abrir el archivo para lectura, hacer un ciclo while(archivo.available()>0), colocar una condición, y solo procesar el registro cuando se cumpla la condición.

Ejemplo: mostrar en pantalla el alumno cuya LU es 3548.

Filtros o condiciones en archivos secuenciales

Los Filtros se usan para obtener información acerca de un subconjunto de renglones del archivo. En este caso, se debe abrir el archivo para lectura, hacer un ciclo while((archivo.available()>0), colocar una condición, y procesar todos aquellos registros que cumplan la condición.

Ejemplo: a partir de un archivo de alumnos de la Licenciatura en Sistemas de Información, generar un archivo secuencial con todos los alumnos de la cátedra "Programación Orientada a Objetos".

Baja o eliminación en archivos secuenciales

Eliminar o dar de baja en un archivo secuencial, implica procesar dos archivos a la vez. El segundo de ellos es un archivo temporal. El algoritmo de eliminación es el siguiente:

1. Abrir el archivo original en modo lectura.
2. Abrir un archivo temporal en modo escritura.
3. Iniciar un ciclo de lectura del archivo original.
   a. Dentro del ciclo leer un registro.
   b. Colocar una condición para determinar si el registro se debe dar de baja. Si no se dará de baja, grabarlo en el archivo temporal.
   c. Fin de ciclo (cerrar el ciclo).
4. Cerrar ambos archivos.
5. Eliminar el archivo original.
6. Renombrar el archivo temporal con el nombre de archivo original.

ARCHIVOS DE ACCESO DIRECTO

Se dice que un archivo es de acceso u organización directa cuando para acceder a un registro ***n*** cualquiera no es necesario pasar por los ***n-1*** registros anteriores. Por esta característica son de acceso mucho más rápidos que los archivos secuenciales.

La clase predefinida para trabajar con este tipo de archivos es la RandomAccessFile. Posee dos constructores:

RandomAccessFile(File file, String modo)
RandomAccessFile(String name, String modo)

donde el **modo** puede ser:

"r" → Lectura
"w" → Escritura
"rw" → Lectura y escritura. Si ya existe, sobrescribe

Esta clase contiene muchos métodos, algunos de las cuales son los siguientes:

Retorno	**Nombre-Parámetro**	
void	Close()	cierra el flujo de datos y libera los recursos de sistema asociados
long	getFilePointer()	retorna la posición actual del puntero interno del archive
long	length()	retorna la longitude en bytes del archive
void	seek(long pos)	coloca el puntero interno en la posición ***pos***, medida desde el principio del archivo, para la próxima lectura o escritura.

Escritura en archivos directos

Algunos métodos de grabación son:

Retorno	**Nombre-Parámetro**	
void	writeChar(int v)	graba un dato de tipo char (2 bytes).
void	writeChars(String s)	graba un string como una secuencia de caracteres (2 bytes para cada caracter)
void	writeInt(int v)	graba un dato de tipo entero (4 bytes).
void	writeDouble(double v)	graba un dato de tipo doble (8 bytes).

Es importante resaltar que **el tamaño de registro debe ser fijo**, por lo que se debe tener en cuenta la cantidad de bytes con que se representa cada tipo de dato (Ej: enteros 4 bytes, char 2 bytes, etc. Ver Anexo I).

Ejemplo 1: Código ejemplo de grabación

```
import java.io.*;

public class GrabaRandom{
  public static void main(String[] args) {
     int clave = 0;
     String nombre = " ";
     int edad = 0;
     char opcion = 'S';
     BufferedReader teclado=new BufferedReader(new InputStreamReader(System.in));

     try {
        // Creando archivo de acceso directo
        RandomAccessFile archi=new RandomAccessFile("c:\\carpeta\\Ran.dat","rw");

        while (opcion != 'n' && opcion != 'N'){
           System.out.print("Ingrese clave: ");
           clave = Integer.parseInt(teclado.readLine());

           System.out.println("Ingrese nombre: ");
           nombre = teclado.readLine();
           //dejando string en 25 caracteres (por tamaño fijo)
           if (nombre.length() < 25){
              for(int i=nombre.length(); i <25; i++)
                 nombre = nombre+" ";
              }else{
                   nombre = nombre.substring(0,25);
              }

          System.out.println("Ingrese edad: ");
          edad = Integer.parseInt(teclado.readLine());

          // grabando al archivo
          if (archi.length()!= 0){
             // se posiciona al final del último registro
             archi.seek( archi.length() );
          }
          archi.writeInt(clave);
          archi.writeChars(nombre);
          archi.writeInt(edad);

          System.out.println("Más datos ? (S/N): ");
          opcion=teclado.readLine().charAt(0);
        }
        archi.close();
     } // cierra try
     catch(FileNotFoundException fnfe){
         System.out.println("Archivo no encontrado");
     }
     catch (IOException ioe){
         System.out.println("Error al escribir");
     }
  } // cierra main
} // cierra clase
```

Se recomienda que las claves sigan la secuencia 0,1,2,3,4,5.....
Recordar que un archivo directo tiene un tamaño de registro fijo, y es importante que dicho tamaño se respete (por ese motivo se ajusta la variable ***nombre***)
También es importante recordar que java grabará cada carácter string usando dos (2) bytes (esto es porque utiliza el código UNICODE, de 16 bits). Un string es una cadena de char. Cada char usa 2 bytes. Es decir que el registro del ejemplo quedará grabado en disco en 58 bytes: 50 para string (25 * 2) y 4 bytes por cada entero. **Es importante conocer el tamaño de registro grabado en disco porque esta información será utilizada en algunas funciones.**

Lectura en archivos directos

Algunos métodos de lectura son:

double	readDouble()	lee un dato de tipo doble.
float	readFloat()	lee un dato de tipo float.
int	readInt()	lee un dato de tipo entero.
char	readChar()	lee un dato tipo char (de a un character).

Ejemplo 2: Código ejemplo de lectura

```
import java.io.*;

public class LeeRandom{
   public static void main(String[] args) {
      int clave = 0;
      String nombre = "";
      int edad = 0;
      long tamreg = 58;  // tamaño de registro
      long canreg = 0;   // cantidad de registros

      try{
         // Creando y leyendo archivo
         RandomAccessFile archi=new RandomAccessFile("c:\\carpeta\\Ran.dat","r");
         //calculando cantidad de registros
         canreg = archi.length()/tamreg;

         // en este bucle lee todos los registros del archivo
         for (int r=0; r < canreg; r++) {
            clave = archi.readInt();
            //leyendo string (lee de a un carácter y concatena)
            for(int i = 0; i < 25; ++i){
                  nombre = nombre + archi.readChar();
            }
            edad = archi.readInt();

            System.out.println(clave+" "+nombre+" "+edad);
            // limpia string, sino concatena con el siguiente
            nombre = "";
         }
         archi.close();
      } //cierra try
      catch(FileNotFoundException fnfe){
            System.out.println("Archivo no encontrado");
      }
      catch (IOException ioe){
            System.out.println("Error al escribir");
      }
  } // cierra main
} // cierra clase
```

Búsqueda en archivos directos

Cuando se desea leer un único registro, proporcionando un dato de búsqueda (generalmente la clave del registro), se observa la gran diferencia con los archivos secuenciales. No es necesario leer todo el archivo. Y también se ve la importancia de grabar una clave numérica incremental (0, 1, 2,..), dado que por medio del método seek() usando como parámetro el tamaño de registro multiplicado por la clave solicitada, permitirá posicionarse en el número de registro deseado.

Ejemplo 3: Código ejemplo de búsqueda

```
import java.io.*;
public class BuscaRandom {
   public static void main(String[] args) {
      int clave = 0;
      String nombre = "";
      int edad = 0;
      long tamreg = 58;  // tamaño de registro
      BufferedReader teclado=new BufferedReader(new InputStreamReader(System.in));

      try{
        RandomAccessFile archi = new RandomAccessFile("c:\\carpeta\\Ran.dat","r");

        System.out.println("Ingrese clave: ");
        clave = Integer.parseInt(teclado.readLine());

        // se posiciona al principio del registro deseado
        archi.seek(clave * tamreg); // argumento es tipo long


        //lee el registro donde se posicionó
        clave=archi.readInt();
        for(int i = 0; i < 25; ++i) {
            nombre += archi.readChar();
        }
        edad=archi.readInt();

        System.out.println(clave+" "+nombre+" "+edad);
        nombre="";
        archi.close();
      } // cierra try
      catch(FileNotFoundException fnfe){ }
      catch (IOException ioe){ }
   } // cierra main
} // cierra clase
```

Baja o eliminación en archivos directos

En archivos directos no se debe eliminar físicamente registros de los archivos, dado que la clave del registro esta enlazada directamente a la posición que dicho registro tiene en disco, y no sería correcto cambiarle la clave a los registros (el número de matrícula de un alumno, el número de serie de un auto, etc.)

Por otra parte, si se elimina físicamente un registro, no hay manera de recuperar esa información posteriormente.

Es por eso que otra técnica común de eliminación es la "baja lógica". Esta consiste en incluir un campo de estado (status, bandera o semáforo) en el registro, y conforme se va cargando el registro y antes de mandarlo a disco se le agrega a dicho campo el carácter "A" (alta), y cuando se quiere dar de baja, solo se pondría dicho campo en "B", y todos los programas de lectura, búsqueda y filtros deberán revisar este campo antes de hacer algo con el registro.

Operaciones con campos en archivos directos

En este caso no es necesario usar dos archivos, sino que una vez leído el registro y modificado el dato deseado, este se vuelve a grabar. Para ello se utiliza la función getFilePointer().

El algoritmo es el siguiente:

1. Lee cada campo
2. Modifica lo deseado
3. Retrocede la cantidad de bytes del tamaño de registro
4. Graba cada campo, incluyendo el modificado

Ejemplo 4: Código ejemplo de operaciones. Suma 10 años a todas las edades.

```
import java.io.*;

public class ModificaRandom {
   public static void main(String[] args) {
      int clave = 0;
      String nombre = "";
      int edad = 0;
      long tamreg = 58;
      long canreg = 0;

      try{
        RandomAccessFile archi=new RandomAccessFile("c:\\carpeta\\Ran.dat","rw");
        canreg = archi.length() /tamreg;
        for (int r=0; r < canreg; r++){
            clave = archi.readInt();
            for(int i = 0; i < 25; ++i) {
                  nombre += archi.readChar();
            }
            edad = archi.readInt();
            //sumando edad + 10
            edad = edad+10;
            //regresando apuntador y regrabando con cambio
            archi.seek(archi.getFilePointer() - tamreg);
            archi.writeInt(clave);
            archi.writeChars(nombre);
            archi.writeInt(edad);
            nombre="";
        } // cierra for
        archi.close();
      } // cierra try
      catch(FileNotFoundException fnfe) { }
      catch (IOException ioe) { }
  } // cierra main
} // cierra clase
```