

GESTIÓN DE ARCHIVOS

BINARIOS CON

LIBRERÍA PACK UTILS

**MARIO JIMÉNEZ MARSET**

**ÍNDICE**

[1. ENUNCIADO – OBJETIVOS 3](#_Toc116746560)

[2. LIBRERÍA PACKUTILS 3](#_Toc116746561)

[3. CONTENIDO DEL PROGRAMA 7](#_Toc116746562)

[4. VISUALIZACIÓN HXD 11](#_Toc116746563)

# ENUNCIADO – OBJETIVOS

En esta práctica se pretendía conocer y profundizar más sobre el tema de la gestión de archivos binarios y conversiones de tipos Java a bytes.

Primero, se debía comentar la librería PackUtils, para terminar de entender cuál es su funcionamiento y su implantación en el código.

Se crean seis objetos personas, que van a ser escritos y leídos posteriormente.

Mediante la clase RandomAccessFile, se almacenan esas seis personas y se hacen operaciones de escritura y lectura. Estas operaciones hacen ver que dentro de este fichero se ha escrito y luego se ha leído por consola.

De igual forma pero con las clases FileOutputStream y FileInputStream se almacenan los mismos objetos personas y se hacen otras operaciones de escritura y lectura, con el mismo objetivo que las operaciones de la clase anterior.

Finalmente, se comprueba mediante el programa HxD (Hexadecimal) que los bytes de los ficheros creados corresponden con la información contenida y orden de guardado en los objetos personas almacenados.

# LIBRERÍA PACKUTILS

En primer lugar se va a plasmar el contenido de la librería PackUtils con sus correspondientes explicaciones.

En el primer método de la librería se va a empaquetar los valores de tipo boolean,

pasando por parámetro un boolean, un array de bytes que va a servir para leer o

escribir los datos y una variable de tipo int, la cual indicara la posición inicial a partir de la que leeremos o escribiremos.

Dentro del método, hay un condicional que dice que, si b es igual a 1, el array de bytes

sea igual a 1(con casteo incluido); si no, será igual a 0. Si es igual a 1, es true; si es igual a 0, es false.

public static void packBoolean(boolean b, byte[] buffer, int offset) {

if(b) {

buffer[offset]=(byte)1;

}

else {

buffer[offset]=(byte)0;

}

}

Sin embargo, en el siguiente método se desempaquetan estos valores de tipo boolean; en este caso, solo se pasan por parámetro el array de bytes y la variable int que indicaba la posición inicial a partir de la que se lee o escribe.

Es un método que devuelve que, el array buffer es igual a 1, lo cual es true, lo que hace que se desempaqueten los bytes correspondientes

public static boolean unpackBoolean(byte[] buffer, int offset) {

return buffer[offset]==(byte)1;

}

En este método, se empaquetan los valores de tipo char. Se pasa por parámetros lo mismo que en el primer método, pero cambiando el boolean por el char; se dice que el array buffer (casteando byte) desplaza 8 bits a la derecha los valores de tipo char. Si se añade 1 al offset, no se desplaza.

public static void packChar(char c, byte[] buffer, int offset) {

buffer[offset]=(byte)(0xFF & (c >> 8));

buffer[offset +1]=(byte)(0xFF & c);

}

En este método se desempaquetan los valores char; se pasa por parámetro lo mismo que al desempaquetar boolean.

Se devuelve (casteando char) el buffer desplazado 8 bits a la izquierda; con el offset sumándole 1, no se desplaza nada.

public static char unpackChar(byte[] buffer, int offset) {

return (char) ((buffer[offset] << 8) | (buffer[offset+1] & 0xFF));

}

En este método se empaquetan valores de tipo String. Se pasa por parámetros lo mismo que en los anteriores métodos, cambiando que se pasa ahora un String y un entero que marca la máxima longitud.

Se hace un bucle for con esta última variable; dentro de él, un if que dice que, si el índice del for es menor a la longitud del String, se llama al método packChar, pasando por parámetro como char a cada posición del for, al buffer y al offset; este último con una modificación: se multiplica la posición específica y se le multiplica por 2; esto se le suma al offset.

El else hace lo mismo, exceptuando que no coge cada posición del for.

public static void packLimitedString(String str, int maxLength, byte[] buffer, int offset) {

for(int i=0;i<maxLength;i++) {

if(i<str.length()) {

packChar(str.charAt(i),buffer,offset+2\*i);

}

else {

packChar('\0', buffer, offset+2\*i);

break;

}

}

}

En este método se desempaquetan los valores String, pasando los mismos valores por parámetro que en el anterior método, pero quitando el String.

Se hace un for (con la misma filosofía que el anterior); dentro, la variable char llama al método de desempaquetar los char, pasando por parámetros el buffer y el offset con la modificación del anterior método.

Después, se hace un if que dice que, si el char es igual a \0, al String result que se acaba de crear se le suma esta c. Finalmente se retorna el String result.

public static String unpacklimitedString(int maxLength, byte[] buffer, int offset) {

String result="";

for(int i=0;i<maxLength;i++) {

char c=unpackChar(buffer, offset+2\*i);

if(c!='\0') {

result+=c;

}

else {

break;

}

}

return result;

}

En este método se empaquetan valores de tipo int, pasando como parámetros un entero, el buffer y el offset.

El buffer (casteando byte) será igual al entero n desplazado 24 bits a la derecha. Se va sumando al offset un número y reduciendo los bits a desplazar hasta que se queda sin bits

public static void packInt(int n, byte[] buffer, int offset) {

buffer[offset]=(byte)(n>>24);

buffer[offset+1]=(byte)(n>>16);

buffer[offset+2]=(byte)(n>>8);

buffer[offset+3]=(byte)n;

}

En este método se desempaquetan los valores de tipo int. Este devuelve lo mismo que el anterior método, pero con un cambio: en vez de desplazar los bits a la derecha, se desplazan a la izquierda. Además, se pasaba por parámetros lo mismo, exceptuando

el entero

public static int unpackInt(byte[] buffer, int offset) {

return ((buffer[offset])<<24)|

((buffer[offset+1]&0xFF)<<16)|

((buffer[offset+2]&0xFF)<<8)|

((buffer[offset+3]&0xFF));

}

Este método de empaquetamiento de variables de tipo long es exactamente igual al de empaquetamiento de variables de tipo int, cambiando simplemente el número de bits a desplazar (que obviamente es más grande) y la variable pasada por parámetro, que es long en vez de int.

public static void packLong(long n, byte[] buffer, int offset) {

buffer[offset]=(byte)(n>>56);

buffer[offset+1]=(byte)(n>>48);

buffer[offset+2]=(byte)(n>>40);

buffer[offset+3]=(byte)(n>>32);

buffer[offset+4]=(byte)(n>>24);

buffer[offset+5]=(byte)(n>>16);

buffer[offset+6]=(byte)(n>>8);

buffer[offset+7]=(byte)n;

}

Este método de desempaquetan las variables de tipo long. Es igual que el anterior método, lo único que cambia son los bits desplazados (que se desplazan a la izquierda) y que no se pasa la variable long por parámetro

public static long unpackLong(byte[] buffer, int offset) {

return ((long)(buffer[offset])<<56)|

((long)(buffer[offset+1]&0xFF)<<48)|

((long)(buffer[offset+2]&0xFF)<<40)|

((long)(buffer[offset+3]&0xFF)<<32)|

((long)(buffer[offset+4]&0xFF)<<24)|

((long)(buffer[offset+5]&0xFF)<<16)|

((long)(buffer[offset+6]&0xFF)<<8)|

((long)(buffer[offset+7]&0xFF));

}

Este método de empaquetamiento de variables double pasa por parámetro un double, el buffer y el offset. Se crea un long de bits que llama al método de la clase Double "doubleToRawLongBits", que pasa por parámetro la variable double anterior.

Este método devuelve los bits representados en esta variable. Luego, se empaquetan esos bits llamando al método packLong

public static void packDouble(double n, byte[] buffer, int offset) {

long bits=Double.doubleToRawLongBits(n);

packLong(bits, buffer, offset);

}

Este método desempaqueta las variables double. Se pasa por parámetro solo el buffer y el offset. La variable bits llama al método unpackLong (pasando por parámetros el buffer y el offset) y retornando los valores double de los bits que estaban antes empaquetados en el método packLong

public static double unpackDouble(byte[] buffer, int offset) {

long bits=unpackLong(buffer, offset);

return Double.longBitsToDouble(bits);

}

# CONTENIDO DEL PROGRAMA

En este apartado se muestra el código del programa, donde se crean los objetos personas y se hacen las operaciones de escritura y lectura mediante ficheros de acceso aleatorio y acceso secuencial.

package archivos\_binarios;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.RandomAccessFile;

public class Main{

private RandomAccessFile raf;

private File file;

private FileOutputStream fos;

private FileInputStream fis;

//Parte RAF

private void writePerson(long num, Person person) throws IOException{

this.raf.seek(num\*Person.SIZE);

byte[] record=person.toBytes();

this.raf.write(record);

}

private Person readPerson(long num) throws IOException{

this.raf.seek(num\*Person.SIZE);

byte[] record=new byte[Person.SIZE];

this.raf.read(record);

return Person.fromBytes(record);

}

//Parte SECUENCIAL

private void writePerson(Person person, boolean salida,long num) throws IOException {

fos = new FileOutputStream(file, salida);

byte[] record = person.toBytes();

for(int i=0;i<record.length;i++) {

fos.write(record[i]);

}

fos.close();

}

private Person readPerson(Person person, boolean salida) throws

IOException {

fis = new FileInputStream(file);

byte[] record = person.toBytes();

for(int i=0;i<record.length;i++) {

fis.readAllBytes();

}

fis.close();

return Person.fromBytes(record);

}

public void run() {

try {

Person p1=new Person(1000, "Ignacio", 40,false);

Person p2=new Person(2000, "Ezequiel", 63, true);

Person p3=new Person(3000, "Zacarias", 18, false);

Person p4=new Person(4000, "Ditario", 24, true);

Person p5=new Person(5000, "Hilario", 43, true);

Person p6=new Person(6000, "Jose Luis", 18, false);

//Parte RAF

raf=new RandomAccessFile("peopleraf.dat","rw");

this.writePerson(0, p1);

this.writePerson(1, p2);

this.writePerson(2, p3);

this.writePerson(3, p4);

this.writePerson(4, p5);

this.writePerson(5, p6);

Person praf;

praf=this.readPerson(0);

System.out.println("p raf= "+praf);

praf=this.readPerson(1);

System.out.println("p raf= "+praf);

praf=this.readPerson(2);

System.out.println("p raf= "+praf);

praf=this.readPerson(3);

System.out.println("p raf= "+praf);

praf=this.readPerson(4);

System.out.println("p raf= "+praf);

praf=this.readPerson(5);

System.out.println("p raf= "+praf);

System.out.println("/\*\*\*\*\*/\n/\*\*\*\*\*/\n/\*\*\*\*\*/\n/\*\*\*\*\*/");

//Parte SECUENCIAL

file=new File("peoplesec.dat");

this.writePerson(p1, false, 0);

this.writePerson(p2, true, 1);

this.writePerson(p3, true, 2);

this.writePerson(p4, true, 3);

this.writePerson(p5, true, 4);

this.writePerson(p6, true, 5);

Person psec;

psec = this.readPerson(p1,false);

System.out.println("p sec= " + psec);

psec = this.readPerson(p2,true);

System.out.println("p sec= " + psec);

psec = this.readPerson(p3,true);

System.out.println("p sec= " + psec);

psec = this.readPerson(p4,true);

System.out.println("p sec= " + psec);

psec = this.readPerson(p5,true);

System.out.println("p sec= " + psec);

psec = this.readPerson(p6,true);

System.out.println("p sec= " + psec);

}

catch(IOException e) {

System.out.println("Ha ocurrido un error");

}

}

public static void main(String[] args) {

Main objetomain=new Main();

objetomain.run();

}

}

En primer lugar se crean las variables de las clases RandomAccessFile, File, FileOutputStream y FileInputStream, las cuales se utilizarán posteriormente.

Respecto al RandomAccessFile, se crean los métodos que escribirán las personas y las leerán. El método writePerson (void) escribe dentro de un array de bytes todas las personas que posteriormente se crearán. El método readPerson (Person) lee este array de bytes y lo devuelve llamando al método de la clase Person fromBytes (el cual devuelve el nombre, el id, la edad y si están casados)

Respecto al FileOutputStream y FileInputStream, se crean los métodos igual que en el RandomAccessFile, con la diferencia de que los métodos están sobrecargados. En el writePerson se crea el objeto FileOutputStream, pasando por parámetro el file creado al principio y un boolean. En el array de bytes se almacenan las personas y, en este se escribirán gracias a un bucle for que lo hace posible. En el ReadPerson se crea el objeto FileInputStream y, con otro bucle for, se lee todo el FileInputStream con el método readAllBytes.

De las 2 formas se pretende conseguir el mismo objetivo: escribir y leer los mismos datos de personas.

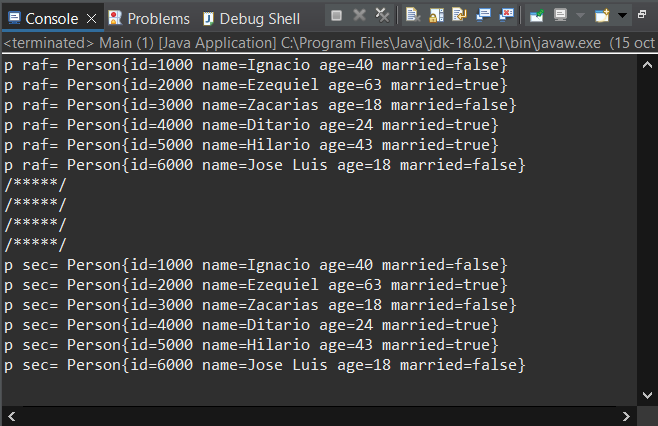
Después, se crea un método de tipo void llamado run, dentro del cual se implementa un try catch, donde se crean 6 objetos persona, dentro de cada uno se ponen datos diferentes (estos 6 objetos son los que hay que leer aleatoria y secuencialmente).

Para hacerlo aleatoriamente, se crea el objeto RandomAccessFile, poniendo como parámetro el fichero donde escribir y leer, además de que este sea de lectura y escritura.

Se llama al método writePerson y se escriben las 6 personas. Luego, se crea un objeto persona y se llama al método readPerson, haciendo que se lean las personas recién escritas.

De igual forma pero creando un file, se llama al método writePerson (el otro sobrecargado) y al método readPerson (también sobrecargado) y se definen las personas en el fichero.

Finalmente, en el main, se crea un objeto de la propia clase, el cual llama al último método creado, consiguiendo por consola visualizar las personas tanto por acceso aleatorio como por acceso secuencial.



Se muestra por consola los resultados: las personas por acceso aleatorio y secuencial.

# VISUALIZACIÓN HXD

Por último, se visualizan los bytes de los dos ficheros .dat creados. Se confirma que la información contenida en los 2 ficheros está bien almacenada.

