Acceso a Datos

UT1. MANEJO DE FICHEROS DE TEXTO Y BINARIOS

FICHEROS BINARIOS Y DE ACCESO ALEATORIO

1. Ficheros binarios en Java

- Los ficheros binarios almacenan secuencias de dígitos binarios que no son legibles por el usuario, a diferencia de lo que ocurría con los ficheros de texto plano.
- Ocupan menos espacio en disco.
- Para trabajar con ficheros binarios usaremos:
  - **FileInputStream** → Para lectura.
  - **FileOutputStream** → Para escritura.

1. Ficheros binarios en Java

FileInputStream

- Esta clase nos permite leer un fichero a través de los siguientes métodos:

Método	Función
int read()	Lee un byte y lo devuelve.
int read(byte[] buf)	Lee hasta buf.length bytes de datos. Los bytes leídos se almacenan en buffer
int read(byte buf, int desplazamiento, int n)	Lee hasta n byte de datos a partir de la posición de desplazamiento.

1. Ficheros binarios en Java

FileOutputStream

- Esta clase nos permite escribir en un fichero a través de los siguientes métodos:

Método	Función
void write(int b)	Escribe un byte
void write(byte[] buf)	Escribe un array de bytes
void write(byte buf, int desplazamiento, int n)	Escribe n bytes de datos a partir de la posición de desplazamiento.

1. Ficheros binarios en Java

FileOutputStream

- Igual que sucede con las clases de escritura de caracteres, FileOutputStream sobrescribre la información ya existente en el fichero.
- Para evitar que esto suceda debemos crear el objeto FileOutputStream en modo append (Añadir).
  - Esto se consigue añadiendo un segundo parámetro al constructor del objeto.

```
FileOutputStream fos = new FileOutputStream(f, true);
```

1. Ficheros binarios en Java

FileInputStream / FileOutputStream

En el siguiente ejemplo podemos ver como creamos un flujo de salida y lo usamos para insertar los 100 primeros números enteros dentro de un fichero.

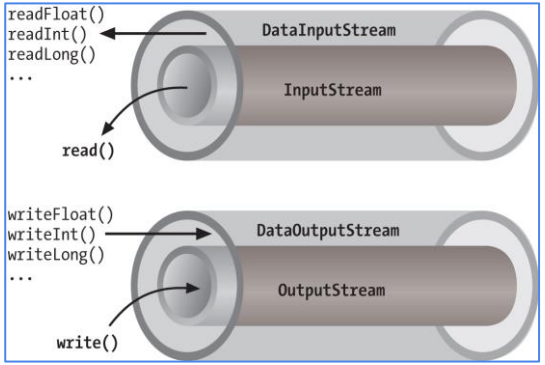
Posteriormente, creamos un flujo de entrada y procedemos a leerlos.

```
File f= new File("./myFiles/Ejemplo1");
try {
    //Creamos un Stream de salida
    FileOutputStream fos = new FileOutputStream(f);
    //Escribimos el el fichero los 100 primeros números
    for(int i=1;i<=100;i++)
        fos.write(i);
    //Cerramos el stream de salida
    fos.close();
    //Creamos un flujo de entrada
    FileInputStream fis= new FileInputStream(f);
    //Visualizamos los datos del fichero.
    int b=0;
    while((b=fis.read())!=-1) {
        System.out.print(b+ " ");
    }
    //Cerramos el Stream de entrada
    fis.close();
} catch (FileNotFoundException e) {

    e.printStackTrace();
} catch (IOException e) {

    e.printStackTrace();
}
```

1. Ficheros binarios en Java

DataInputStream / DataOutputStream

- Con las clases FileInputStream y FileOutputStream podemos leer y escribir bytes respectivamente.
- Sin embargo, dependiendo del tipo de dato que queramos utilizar su codificación binaria será diferente.
- Es por ello que para la escritura de tipos primitivos emplearemos los métodos:
  - **DataInputStream** → Lectura
  - **DataOutputStream** → Escritura
- Estas clases actúan como un wrap de las clases FileInputStream y FileOutputStream dotandolas de la capacidad de trabajar con diferentes tipos de datos (Más allá de poder leer y escribir bit a bit).



1. Ficheros binarios en Java

DataInputStream / DataOutputStream

- La siguiente tabla muestra los principales métodos de lectura y escritura de las clases DataInputStream y DataOutputStream.

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

1. Ficheros binarios en Java

DataInputStream / DataOutputStream

- Para crear un DataInputStream debemos partir de un FileInputStream como muestran los siguientes ejemplos de código.

```
File f= new File("./myFiles/Ejemplo2.dat");
FileInputStream fis= new FileInputStream(f);
DataInputStream dis= new DataInputStream(fis);
```

```
File f= new File("./myFiles/Ejemplo2.dat");
DataInputStream dis= new DataInputStream(new FileInputStream(f));
```

- Del mismo modo, para crear un DataOutputStream debemos partir de un FileOutputStream.

```
File f= new File("./myFiles/Ejemplo2.dat");
FileOutputStream fos = new FileOutputStream(f);
DataOutputStream dos= new DataOutputStream(fos);
```

```
File f= new File("./myFiles/Ejemplo2.dat");
DataOutputStream dos= new DataOutputStream(new FileOutputStream(f));
```

1. Ficheros binarios en Java

DataInputStream / DataOutputStream

- El siguiente ejemplo muestra cómo insertar datos en un fichero binario utilizando DataOutPutStream
  - Como puede verse, dependiendo del tipo de dato que queramos introducir emplearemos un método u otro.
  - Cómo siempre, al terminar la operación **CERRAMOS LOS FLUJOS.**

```
//Creamos el fichero y su flujo de salida
File f= new File("./myFiles/Ejemplo2.dat");
FileOutputStream fos = new FileOutputStream(f);
DataOutputStream dos= new DataOutputStream(fos);
//Definimos dos arrays que escribiremos en el fichero
String[] alumnos= {
"Alberto", "Ana","Josefina","Maribel", "Adriano", "Mariano"
};
int[] calificaciones= {1,2,3,4,5,6};

//Recorremos los arrays y los vamos escribiendo
for (int i = 0; i < alumnos.length; i++) {
    dos.writeUTF(alumnos[i]);
    dos.writeInt(calificaciones[i]);
}
//Creamos el flujo
dos.close();
fos.close();
```

1. Ficheros binarios en Java

DataInputStream / DataOutputStream

- El siguiente ejemplo muestra cómo leer datos de un fichero binario utilizando DataInputStream
  - Como puede verse, dependiendo del tipo de dato que queramos leer emplearemos un método u otro.
  - Cómo siempre, al terminar la operación **CERRAMOS LOS FLUJOS.**

```
//Creamos el flujo de entrada
File f= new File("./myFiles/Ejemplo2.dat");
DataInputStream dis= new DataInputStream(new FileInputStream(f));
//Creamos un try para capturar el EOF
try {
    //Iniciamos un bucle infinito para recorrer todo el fichero
    do {
        System.out.println(dis.readUTF()+ " ha obtenido un "
    +dis.readInt());
    }while(true);
    /*
     * Cuando el fichero llegue al final (EOF) saltará la expceción
     * EOFException. La capturamos para poder cerrar el lujo una vez
     * recorrido todo el fichero.
     */
}catch (EOFException e) {
    //Cerramos el flujo
    dis.close();
}
```

1. Ficheros binarios en Java

Objetos en ficheros binarios

- Hasta ahora hemos visto que en los ficheros binarios es posible almacenar datos primitivos de forma sencilla.
- Sin embargo, una de las grandes virtudes de Java es la orientación a objetos
    - Imaginemos que contamos con un objeto alumno con los atributos nombre, edad, curso y nota.
    - Con los mecanismos estudiados hasta ahora nos veríamos obligados a introducir cada uno de los atributos de forma individual. Lo cual aumentan en varios órdenes de magnitud la complejidad del código.

1. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

- Por suerte, Java permite guardar directamente los objetos en ficheros binarios.
- Para que un objeto pueda ser guardado en un fichero binario, este debe implementar la interface ***serializable.***

```
public class Alumno implements Serializable
```

- Cualquier objeto que implemente esta interface podrá ser leído o escrito a través de
  - **ObjectInputStream** → Lectura de objetos.
  - **ObjectOutputStream** → Escritura de objetos.

- Esto es posible gracias a que cualquier objeto serializable puede ser convertido en una secuencia de bits que posteriormente puede ser restaurada, permitiendo así su recuperación.

1. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

- Para crear un ObjectInputStream debemos partir de un FileInputStream como muestran los siguientes ejemplos de código.

```
File f= new File("./myFiles/EjemploObjetos1.dat");
FileInputStream fis= new FileInputStream(f);
ObjectInputStream ois= new ObjectInputStream(fis);
```

```
File f= new File("./myFiles/EjemploObjetos1.dat");
ObjectInputStream ois= new ObjectInputStream(new FileInputStream(f));
```

- Del mismo modo, para crear un ObjectOutputStream debemos partir de un FileOutputStream.

```
File f= new File("./myFiles/EjemploObjetos1.dat");
FileOutputStream fos= new FileOutputStream(f);
ObjectOutputStream oos= new ObjectOutputStream(fos);
```

```
File f= new File("./myFiles/EjemploObjetos1.dat");
ObjectOutputStream oos= new ObjectOutputStream(new FileOutputStream(f));
```

1. Ficheros binarios en Java

Objetos en ficheros binarios:
ObjectInputStream / ObjectOutputStream

- Para los siguientes ejemplos usaremos la clase **Alumno**
  - Como se puede ver la clase implementa la interface *serializable*.

```
public class Alumno implements Serializable{


    private static final long serialVersionUID = 1L;
    private String nombre;
    private int calificacion;

    public Alumno(String nombre, int calificacion) {
        super();
        this.nombre = nombre;
        this.calificacion = calificacion;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getCalificacion() {
        return calificacion;
    }

    public void setCalificacion(int calificacion) {
        this.calificacion = calificacion;
    }



}
```

1. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

El siguiente código muestra un ejemplo de escritura en fichero utilizando la clase ObjectOutputStream

```
//Creamos e inicializamos el flujo
File f= new File("./myFiles/EjemploObjetos1.dat");
ObjectOutputStream oos= new ObjectOutputStream(new FileOutputStream(f));

//Definimos un array con los objetos que vamos a insertar
Alumno alumnos[]= {new Alumno("Pepe",10), new Alumno("Maria", 5)};

//Insertamos los objetos
for (int i = 0; i < alumnos.length; i++) {
    oos.writeObject(alumnos[i]);
}
//Cerramos el flujo
oos.close();
```

1. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

El siguiente código muestra un ejemplo de escritura en fichero utilizando la clase ObjectOutputStream

```
//Definimos el stream de entrada
File f= new File("./myFiles/EjemploObjetos1.dat");
ObjectInputStream ois= new ObjectInputStream(new FileInputStream(f));
/*
 * Leemos el fichero hasta que se produzca la EOFException, momento en
 * el que cerramos el flujo
 */
try {
    while(true) {
        Alumno al=(Alumno) ois.readObject();
        System.out.println(al.getNombre()+ " "+ al.getCalificacion());
    }
} catch (ClassNotFoundException e) {

}catch (EOFException e) {
    System.out.print("Fin del archivo");
    ois.close();
}
```

1. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

- Problemática con la modificación de ficheros de objetos:
  - Cada vez que se crea un fichero de objetos Java crea una cabecera inicial con metainformación y tras esta se insertan los objetos.
  - Si el fichero se utiliza para añadir nuevos objetos (segunda apertura) se añade una nueva cabecera y se insertan objetos a partir de ella.
  - Esto produce que, cuando vamos a leer el fichero se detecte la segunda cabecera "intercalada" con los objetos y nos salte una StreamCorruptedException.
  - Esta cabecera se agrega cada vez que creamos un nuevo ObjectOutputStream(fichero) por lo que la solución es redefinir esta clase para evitar que la cree.

1. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

- Problemática con la modificación de ficheros de objetos:

```
public class MyObjectOutputStream extends ObjectOutputStream {

    protected MyObjectOutputStream() throws IOException, SecurityException {
        super();
    }
    public MyObjectOutputStream(OutputStream out) throws IOException {
        super(out);
    }

    @Override
    protected void writeStreamHeader() throws IOException{
        //Este es el método encargado de crear la cabecera,
        //Lo dejamos vácio para que no haga nada
    }


}
```

1. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

- Problemática con la modificación de ficheros de objetos:
  - De forma que
    - Si el archivo no existe usaremos la clase ObjectOutputStream
    - Si ya existía, usaremos la nueva clase creada por nosotros.

```
File f= new File("./myFiles/Ejemplo5.dat");
ObjectOutputStream os=null;
if(f.exists())
    os= new MyObjectOutputStream(new FileOutputStream(f));
else
    os= new ObjectOutputStream(new FileOutputStream(f));
```

2. Ficheros de acceso aleatorio en Java

La clase RandomAccessFile

- Hasta ahora, todas las operaciones que hemos realizado sobre ficheros se realizaban de forma secuencial.
  - Comenzábamos la lectura por el primer byte o caracter hasta llegar al final del fichero.
- Java dispone de la clase **RandomAccessFile** que dispone métodos para acceder al contenido de un fichero binario de forma aleatoria.

2. Ficheros de acceso aleatorio en Java

La clase RandomAccessFile

- Para crear una instancia de RandomAccessFile usaremos cualquiera de los siguientes constructores:
  - Indicando la ruta hacia el fichero.

```
RandomAccessFile(String nombreFichero, String modoAcceso)
```

  - Usando un objeto File para hacer referencia al fichero.

```
RandomAccessFile(File objetoFile, String modoAcceso)
```

- En ambos casos debemos indicar el modo de acceso con el que queremos crear el objeto:
  - r → Abre el fichero en modo de solo lectura.
  - rw → Abre el fichero en modo lectura y escritura.

```
RandomAccessFile raf= new RandomAccessFile("./myFiles/miFichero.dat", "rw");
```

```
RandomAccessFile raf= new RandomAccessFile(new File("./myFiles/miFichero.dat"), "r");
```

2. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria.

- Usaremos las clases DataInputStream y DataOutputStream.
- La clase **RandomAccessFile** maneja puntero que indica la posición actual en el fichero.
  - Cuando se crea el fichero el puntero se coloca en la posición 0.
    - Las sucesivas llamadas a .write() y .read() ajustan el puntero según la cantidad de bytes escritos.
  - Los métodos más importantes para el manejo de este puntero son:

Método	Función
long getFilePointer()	Devuelve la posición actual del puntero del fichero
void seek (long posición)	Coloca el puntero en una posición determinada
long length()	Devuelve el tamaño del fichero en bytes
int skipBytes (int desp)	Desplaza el puntero desde la posición actual el número de bytes indicados por desp.

2. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria

- Gracias al puntero de fichero podemos leer y escribir directamente en cualquier punto del fichero.
  - Para saber dónde debemos situar el puntero debemos conocer el tamaño de los registros que estamos almacenando.

Data Type	Size
byte	1 byte
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
boolean	1 bit
char	2 bytes

2. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria

- En el siguiente ejemplo vamos a escribir registros de tipo empleado, formados por:
  - ID → int (4 Bytes)
  - Apellido → 10 caracteres (2 Byte/char)(2*10=20 Bytes)
  - Departamento → int (4 Bytes)
  - Salario → Double (8 Bytes)
    - TOTAL → 36 Bytes

2. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria

Escritura en fichero

```
//Declaramos los arrays a insertar
String[] apellidos= {"Martinez", "Gil", "Ondaruina"};
int[] dpto= {10,20,30};
double[] salario= {300.2, 123.45, 1256.1};

try {

    //Creamos el fichero
    RandomAccessFile raf= new RandomAccessFile(new File("./myFiles/misEmpleados.dat"), "rw");

    for(int i=0;i<apellidos.length;i++) {
        //Insertamos la información
        raf.writeInt(i+1);
        StringBuffer sb= new StringBuffer(apellidos[i]);
        sb.setLength(10);
        raf.writeUTF(sb.toString());
        raf.writeInt(dpto[i]);
        raf.writeDouble(salario[i]);
    }
    //Cerramos el fichero
    raf.close();
} catch (FileNotFoundException e) {

    e.printStackTrace();
} catch (IOException e) {

    e.printStackTrace();
}
```

2. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria

Lectura de fichero

```
//Creamos el fichero
try {
    RandomAccessFile raf= new RandomAccessFile(new File("./myFiles/misEmpleados.dat"), "r");
    //Declaramos las variables necesarias
    int pos=0;
    int id, dep;
    Double salario;
    char[] apellido= new char[10];
    //Recorremos el fichero
    for(pos=0;pos<raf.length();pos+=36) {
        raf.seek(pos);      //Posicionamos el puntero al comienzo del registro
        id=raf.readInt();
        for(int i=0;i<10;i++) {
            apellido[i]=raf.readChar();
        }
        dep=raf.readInt();
        salario=raf.readDouble();
        System.out.printf("ID: %s, Apellido: %s, Dpto: %d Salario: %.2f %n",
                id, String.valueOf(apellido),dep, salario );


    }
    raf.close(); //Cerramos el fichero
```

2. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria

- En los dos ejemplos anteriores (Escritura y lectura) hemos hecho uso de la clase RandomAccessFile para realizar una escritura y una lectura secuenciales.
  - Sin embargo, con este tipo de ficheros no es necesario recorrer todos los registros para poder acceder a uno concreto.
  - En nuestro ejemplo, conocemos el tamaño de cada registro por tanto, si quisiéramos acceder a uno en concreto podríamos sencillamente calcular su posición y ubicar el puntero en ella.
    - Por ejemplo, para leer los datos del empleado 5 deberíamos situar el puntero en (5*36) la posición 180

2. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria

```
//Creamos el fichero
RandomAccessFile raf= new RandomAccessFile(new File("./myFiles/misEmpleados.dat"), "r");
//Declaramos las variables necesarias

int id, dep;
Double salario;
char[] apellido= new char[10];

//Vamos a leer los datos del segundo registro (ID-1)
raf.seek(1*36);//Posicionamos el puntero al comienzo del registro 2
id=raf.readInt();
for(int i=0;i<10;i++) {
    apellido[i]=raf.readChar();
}
dep=raf.readInt();
salario=raf.readDouble();
System.out.printf("ID: %s, Apellido: %s, Dpto: %d Salario: %.2f %n",
        id, String.valueOf(apellido),dep, salario );
raf.close(); //Cerramos el fichero
```

2. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria

- Para mantener la estructura del fichero debemos controlar en qué posición del fichero se insertan cada uno de los registros.
  - Para insertar un nuevo registro aplicamos la función al identificador para cualquier posición.
    - En el siguiente ejemplo se insertará un empleado con identificador 20, debemos calcular donde estará situado (ID-1)*tam → (20-1)*36

```
RandomAccessFile raf= new RandomAccessFile(new File("./myFiles/misEmpleados.dat"), "rw");
StringBuffer sb=null;
String apellido= "Hidalgo";
Double salario=2000.0;
int id=20;
int dpto=10;

long pos= (id-1)*36; //calculamos la posición
raf.seek(pos);//Nos posicionamos
raf.writeInt(id);//Escribimos el ID
sb=new StringBuffer(apellido);
sb.setLength(10);
raf.writeChars(sb.toString()); //Escribimos el apellido
raf.writeInt(dpto);//Escribimos el dpto
raf.writeDouble(salario); //Escribimos el salario
raf.close();//Cerramos el fichero
```

Dudas y preguntas