Lectura/Escritura en ficheros binarios utilizando memoria intermedia (*buffered*)

Para leer y escribir bytes con memoria intermedia:

```
FileInputStream fis;
fis = new FileInputStream("imagen.jpg")
BufferedInputStream bis= new BufferedInputStream(fis);

FileOutputStream fos;
fos = new FileOutputStream("copia_imagen.jpg");
BufferedOutputStream bos = new
BufferedOutputStream(fos);
```

Lectura/Escritura en ficheros de caracteres haciendo lectura por líneas

```
BufferedReader flujoDeEntrada = null;
PrintWriter flujoDeSalida = null;
try {
        flujoDeEntrada =  new BufferedReader(new FileReader(“libro.txt"));
        flujoDeSalida =  new PrintWriter(new FileWriter("copia_libro.txt"));
        String linea;
        while ((linea = flujoDeEntrada.readLine()) != null) {
                flujoDeSalida.print(linea);
        }
 }  finally {
    if (flujoDeEntrada != null) {
        flujoDeEntrada.close();
    }
    if (flujoDeSalida != null) {
        flujoDeSalida.close();
    }
```

Lectura/Escritura de datos de tipos primitivos

- Se utilizan las clases DataInputStream y DataOutputStream. Estas clases, además de los métodos read() y write(), proporcionan métodos específicos para cada tipo de dato primitivo.
- Ejemplos:
  - File fich= new File(ruta);
  - FileInputStream fis= new FileInputStream(fich);
  - DataInputStream dis= new DataInputStream(fis);

  - File fich= new File(ruta);
  - FileOutputStream fos= new FileOutputStream(fich);
  - DataOutputStream dos= new DataOutputStream(fos);

Lectura/Escritura de datos de tipos primitivos (*buffered*)

- Se pueden utilizar con memoria intermedia. Por ejemplo:

```
archivo_salida = new DataOutputStream(
                    new BufferedOutputStream(
                       new FileOutputStream(ruta)));

archivo_entrada = new DataInputStream(
                     new BufferedInputStream(
                        new FileInputStream(ruta)));
```

donde ruta denota un ejemplar de *File* o un ejemplar de *String*.

Lectura/Escritura de Objetos

Con algunas limitaciones, los objetos se pueden leer y escribir en flujos de forma similar a los tipos primitivos. Concretamente, las clases ObjectInputStream y ObjectOutputStream poseen las mismas funcionalidades que DataInputStream y DataOutputStream (y otras específicas de objetos), lo cual da lugar a que un flujo de objetos pueda contener tanto tipos primitivos como objetos. Los métodos que más interés tienen son

```
Object readObject()
void writeObject(Object obj)
```

que leen/escriben el próximo objeto presente en el flujo.

Acceso aleatorio

- Java dispone de la clase RandomAccessFile que permite acceder al contenido de un fichero binario de forma aleatoria.
- Para crear objetos de esta clase, al constructor se le puede pasar o la ruta mediante un String o mediante un File. En cualquiera de los dos casos hay que pasar un segundo parámetro con el modo de acceso. Ejemplos:
  - Fich= new RandomAccessFile(String nombre, String modoAcceso)
  - Fich = new RandomAccessFile(File fic, String modoAcceso)
  - El modo de acceso puede ser "r", "rw"

Acceso aleatorio

- Los métodos más importantes son:
  - long getFilePointer()
    - Devuelve la posición del puntero en el fichero.
  - void seek(long pos)
    - Pone el puntero en la posición dada por pos, tomada desde el inicio.
  - long length()
    - Devuelve el tamaño del fichero en bytes.
  - Int skipBytes (int desplaz)
    - Mueve el puntero desde su posición el número de bytes indicado por desplaz.

Acceso aleatorio

- Ejemplo, vamos a guardar los datos de una serie de empleados. Interesa que los datos de cada persona tengan la misma longitud para poder saltar e ir a una persona, para ello limitamos (fijamos) el tamaño de los String antes de escribirlos:
  - buffer = new StringBuffer("GARCIA");
  - buffer.setLength(10);
- El resto de campos tienen una longitud fija, porque son enteros y double.
  - int identif;
  - int departamento;
  - float salario;

Acceso aleatorio

```
public class EscribirFichAleatorio {
public static void main(String[] args) throws IOException {
        // TODO code application logic here
        File fich = new File("Empl_aleat.dat");
        RandomAccessFile file = new RandomAccessFile(fich, "rw");
        String[] apellidos = {"MARTÍN", "SORIA", "FERNÁNDEZ", "LUNA", "GARCÍA", "PÉREZ",
"RODRÍGUEZ", "MARTÍNEZ"};
        int[] dept = {10, 20, 30, 20, 10, 40, 30, 40};
        Double[] salario = {850.65, 12035.36, 2156.36, 1500.32, 989.23, 1566.32, 1866.88, 2356.78};
        StringBuffer buffer = null;
        for (int i = 0; i < apellidos.length; i++) {
            file.writeInt(i + 1); //identificador de empleado
            buffer = new StringBuffer(apellidos[i]);
            buffer.setLength(10);
            file.writeChars(buffer.toString());
            file.writeInt(dept[i]);
            file.writeDouble(salario[i]);
        }
```

Formatos en ficheros de texto

- Con longitud de campos fija o encolumnados.
  - Cada campo se almacena con una longitud fija para todos los elementos.
  - Es necesario conocer estos valores de longitudes para poder leer y escribir.
  - Supongamos que queremos guardar datos de empleados (dni (9 caracteres), edad (2 dígitos), nombre (20 caracteres) y salario mensual (5 dígitos y 2 decimales). Podríamos escribir con un formato %9s%2d%20s%7.2f\n
  - Ejemplo:
    - 14567432F32 Ana Martín Pérez 1654.43
    - 4322224M28 Manuel Rodríguez Mar 998.32
    - 657897H45 Jesús López Castro 10234.23