

El paquete java.io.

Manejo de las I/O.

Leo Suarez

leo@javahispano.com

<http://www.javahispano.com>

Julio 2001

```
BufferedOutputStream (OutputStream in);  
BufferedOutputStream (OutputStream in, int bufSize);
```

Combinación de las clases.

A continuación veremos una serie de ejemplos que combinan las clases que tratan directamente sobre los flujos de entrada o de salida con las clases que las envuelven.

Ejemplo 4. Lectura de datos crudos (raw data).

```
/*  
En este ejemplo leemos un fichero que contiene un flujo de bytes que  
representa una imagen  
2D de tamaño 128x128. Cada pixel viene codificado por 2 bytes que indican  
el nivel de gris de dicho pixel. Así  
mismo, escribimos en el fichero salida.txt los niveles de grises con un  
"formato" de array 2D.  
Este programa lee de manera correcta cualquier short, aunque exceda de  
32767.  
*/  
import java.io.*;  
class RawData {  
  
    final static int WIDTH = 128;  
    final static int HEIGHT = 128;  
    public static void main (String args[]) {  
  
        InputStream in = null;  
        DataInputStream dis = null;  
        FileOutputStream fout = null;  
        PrintStream miSalida = null;  
        int shortLeido = 0;  
        try {  
            in = new FileInputStream("Rana_128.035");//Leemos un stream de bytes  
            dis = new DataInputStream(in);//Envolvemos la clase de entrada (in)  
para acceder a las funcionalidades de DataInput  
            fout = new FileOutputStream("salida.txt");  
            miSalida = new PrintStream(fout);//convertimos a PrintStream  
        }  
        catch(IOException e) {  
            System.out.println("Error al abrir el fichero");  
            System.exit(0);  
        }  
        catch (Exception e) {  
            System.out.println(e.getMessage());  
            System.exit(0);  
        }  
  
        int contX = 0, contY = 0;  
        for (contX = 0; contX < HEIGHT; contX++) {  
            miSalida.print("\n");  
            for (contY = 0; contY < WIDTH; contY++) {  
                try {  
                    shortLeido = dis.readUnsignedShort();//Al leer short sin signo  
podemos tratar datos hasta 65535.  
                    miSalida.print("[ " + shortLeido + " ] ");  
                }  
            }  
        }  
    }  
}
```

```
        catch(IOException e) {
            System.out.println("Se leyó todo el fichero " + e);
        }
        catch (Exception e) {
            System.out.println("excepción desconocida: " + e);
            System.exit(0);
        }
    }
} // fin for exterior
try {
    in.close();
    fout.close();
} catch (IOException ioe) {System.out.println("No se pudo cerrar alguno
de los ficheros");}

} // end main

} //Fin RawData
```

Ejemplo 5. Lectura a través de la red. Carga de un fichero desde un applet.

```
/*
En este ejemplo cargamos un fichero a través de la red. Este ejemplo es la
manera que se suele
emplear para cargar un fichero. Como nuestra intención es leer una ristra
de caracteres hacemos
un "wrapping" tal y como explicamos.
*/
/*
<applet code = ReadFromApplet.class width = 400 height = 400>
</applet>
*/
import java.applet.Applet;
import java.io.InputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.net.URL;
import java.net.MalformedURLException;
public class ReadFromApplet extends Applet {

    URL url;
    InputStream in;
    BufferedReader br;
    public void init() {
        try {
            url = new URL(this.getCodeBase() + "Propiedades.java");
        } catch (MalformedURLException mue)
        {System.out.println(mue.getMessage());}

        try {
            in = url.openStream();//abre un flujo de bytes

            //hacemos el wrapping de la clase orientada a byte para leer caracteres
            br = new BufferedReader(new InputStreamReader(in));

            String linea;
```

```
while ((linea = br.readLine()) != null) {  
    System.out.println(linea);  
}  
  
br.close();  
} catch (IOException ioe) {System.out.println(ioe.getMessage());}  
}  
}
```

Las clases orientadas a flujo de caracteres (Character Stream).

Aunque las clases orientadas al flujo de bytes nos proporcionan la suficiente funcionalidad para realizar cualquier tipo de operación de entrada o salida, éstas no pueden trabajar directamente con caracteres Unicode. Es por ello que fue necesario la creación de las clases orientadas al flujo de caracteres para ofrecernos el soporte necesario para el tratamiento de caracteres.

Como ya dijimos al principio, estas clases nacen de las clases abstractas **Reader** y **Writer**. Las clases concretas derivadas de ellas tienen su homónimo en las clases concretas derivadas de **InputStream** y **OutputStream**, por tanto, la explicación hecha para todas ellas tiene validez para las orientadas a carácter salvo quizás algunos matices que podrías completar con las especificaciones por lo que omitiremos la explicación detallada de cada una de ellas.

No obstante, os daré una relación de las clases más importantes. Son las siguientes:

1. Acceso a fichero: **FileReader** y **FileWriter**.
2. Acceso a carácter: **CharArrayReader** y **CharArrayWriter**.
3. Buferización de datos: **BufferedReader** y **BufferedWriter**.

Serialización.

La serialización es el proceso de escribir el estado de un objeto a un flujo de bytes. La utilidad de esta operación se manifiesta cuando queremos salvar el estado de nuestro programa en un sitio de almacenamiento permanente o, en otras palabras, cuando queremos hacer la persistencia de nuestro programa. Así, en un momento posterior dado podemos recuperar estos objetos deserializándolos.

Otra de las situaciones en las que necesitamos recurrir a la serialización es cuando hacemos una implementación RMI. La Invocación de Métodos Remotos consiste en que un objeto Java de una máquina pueda llamar a un método de un objeto Java que está en otra máquina diferente. Entonces, la máquina que lo invoca serializa el objeto y lo transmite mientras que la máquina receptora lo deserializa.

Dado que cuando un objeto se serializa, éste puede tener referencias a otros objetos que a la vez lo tendrán a otros, los métodos para la serialización y deserialización de objetos contemplan esta posibilidad. Así, cuando serializamos un objeto que está en la cima del grafo de objetos, todos los objetos a los que se hace referencia son también serializados. El proceso inverso de recuperación de objetos hará justo lo contrario.