**Leer y Escribir Objetos Java en un Fichero**

Vamos con un ejemplo tonto de cómo escribir objetos en un fichero y leerlos luego.

Usaremos un **ObjetOutputStream** y al [final](http://www.chuidiang.com/java/ficheros/ObjetosFichero.php#problemas) aprovecharé para comentar un par de cosas raras que he visto en esta clase y que algo tan sencillo como leer y escribir de un fichero, nos pueda traer de cabeza durante un buen rato.

**Las clases de datos**

Primero definimos las clases de datos que vamos a escribir y leer en el fichero. Estas clases deben implementar la interface **Serializable**. También todos los atributos de estas clases deben ser tipos primitivos (**int**, **double**, **float**, etc) o bien clases que a su vez implementen la interface **Serializable**.

Implementar esta interface es sencillo. Simplemente ponemos que la implementa y ya está, no es necesario implementar ningún método.

Como clases para el ejemplo vamos a usar una clase **Persona** con una serie de datos y que a su vez, dentro, tiene una clase **Mascota**, también con una serie de datos.

Estas son las clases:

Persona.java

public class Persona implements Serializable  
{  
    public String nombre;  
    public String apellido;  
    public Mascota mascota=new Mascota();  
    public int edad;  
      
    /\*\* Método para que al meter esta clase en un System.out.println() salga  
     \* algo legible.  
     \*/  
    public String toString()  
    {  
        return nombre+" "+  
        apellido+" de "+  
        edad+" años tiene como mascota a "+  
        mascota.nombre+" de "+  
        mascota.numeroPatas+" patas.";  
    }  
}

Aunque por simplicidad no los he puesto, si descargas [los fuentes](http://www.chuidiang.com/java/ficheros/ejemplo.zip) verás que he puesto un constructor y un método **setPersona()** para rellenar fácilmente los campos. Lo importante de esta clase es que implementa **Serializable** y que todos sus atributos (incluido **Mascota**), también.

Mascota.java

public class Mascota implements Serializable  
{  
    public String nombre;  
    public int numeroPatas;  
}

**Escribir en el fichero**

Para escribir en el fichero, simplemente hay que crear un **ObjectOutputStream** sobre el fichero

ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fichero));

Y ahora hay que ir instanciando datos y metiéndolos en el **ObjectOutputStream**. Ojo, si no quieres problemas raros, haz un **new** por cada objeto que quieras meter, no reaproveches la misma instancia cambiándole los datos.

for (int i = 0; i <5; i++)  
{  
    // ojo, se hace un new por cada Persona. El new dentro del bucle.  
    Persona p = new Persona(i);  
    oos.writeObject(p);  
}  
oos.close();  // Se cierra al terminar.

**Lee el fichero**

Para leer del fichero, creamos un **ObjectInputStream** sobre el fichero

ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fichero));

y nos dedicamos a ir leyendo objetos.

// Se lee el primer objeto  
Object aux = ois.readObject();  
              
// Mientras haya objetos  
while (aux!=null)  
{  
    if (aux instanceof Persona)  
        System.out.println(aux);  // Se escribe en pantalla el objeto  
    aux = ois.readObject();  
}  
ois.close();

**Primer problema**

Hay una cosa curiosa con el **ObjectOutputStream**. Supongo que por hacerlo eficiente, cuando le damos un objeto para escribir, es como si guardara el array de bytes en el interior. Si cambiamos los valores de los atributos de ese objeto y volvemos a escribirlo ... el **ObjetOutputStream** lo escribe nuevamente, pero con los datos antiguos. Da la impresión de que no se entera del cambio y no recalcula los bytes que va a escribir en el fichero. Si escribimos así, con un solo new

Persona p = new Persona(0);  // Un único new fuera del bucle  
for (int i = 0; i <5; i++)  
{  
     p.setPersona(i);  // cambiamos los datos de p, pero no hacemos new.  
    oos.writeObject(p);  
}  
oos.close();  // Se cierra al terminar.

Cuando leamos, obtendremos cinco veces la primera persona.

Esto puede evitarse de tres formas:

* Haciendo un **new** de cada objeto que queramos escribir, sin reaprovechar instancias. Esto es lo que se ha hecho en el código anterior.
* Usar el método **writeUnshared()** en vez de **writeObject()**. Este método funcionará bien si cambiamos **TODOS** los atributos de la clase **Persona**. Si no modificamos uno de los atributos, obtendremos resultados extraños en ese atributo.
* Llamando al método **reset()** de **ObjectOutputStream** después de escribir cada objeto. Aunque funciona, no me parece damasiado fino.

**Segundo problema**

Un segundo problema que he visto en el **ObjectOutputStream** es que al instanciarlo, escribe unos bytes de cabecera en el fichero, antes incluso de que escribamos nada. Como el **ObjectInputStream** lee correctamente estos bytes de cabecera, aparentemente no pasa nada y ni siquiera nos enteramos que existen.

El problema se presenta si escribimos unos datos en el fichero y lo cerramos. Luego volvemos a abrirlo para añadir datos, creando un nuevo **ObjectOutputStream** así

/\* El true del final indica que se abre el fichero para añadir datos al final del fichero.\*/  
ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fichero,true));

Esto escribe una nueva cabecera justo al final del fichero. Luego se irán añadiendo los objetos que vayamos escribiendo. El fichero contendrá lo del dibujo, con dos cabeceras.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Contenido del fichero | | | | | | | | |
| Primera sesión con el fichero | | | | | Segunda sesión con el fichero. Le añadimos datos | | | |
| cabecera | Persona | Persona | Persona | Persona | cabecera | Persona | Persona | Persona |

¿Qué pasa cuando leamos el fichero?. Al crear el **ObjectInputStream**, este lee la cabecera del principio y luego se pone a leer objetos. Cuando llegamos a la segunda cabecera que se añadió al abrir por segunda vez el fichero para añadirle datos, obtendremos un error **StreamCorruptedException** y no podremos leer más objetos.

Una solución es evidente, no usar más que un solo **ObjectOuptutStream** para escribir todo el fichero. Sin embargo, esto no es siermpre posible. Por ejemplo, si nuestro programa es una agenda, un día escribimos tres amigos, cerramos la agenda, apagamos el ordenador y nos vamos de juerga. Al día siguiente, queremos meter a los dos borrachines que conocimos en la juerga anterior o a la chica con la que creemos que hemos ligado, encendemos el ordenador, arrancamos la agenda y por más que buscamos, de nuestro antiguo **ObjectOutputStream** no queda ni rastro. Hay que abrir uno nuevo. No se puede pretender en una agenda que metamos a todos nuestros amigos de una sola vez y que no volvamos a meter a nadie más.

La única solución que he encontrado (que seguramente no es la única) es hacernos nuestro propio **ObjectOutputStream**, heredando del original y redefiniendo el método **writeStreamHeader()** como en la figura, vacío, para que no haga nada.

protected void writeStreamHeader() throws IOException  
{  
// No hacer nada.  
}

El ejemplo

En [ejemplo.zip](http://www.chuidiang.com/java/ficheros/ejemplo.zip) tienes un pequeño programa con lo comentado hasta aquí: Las clases Persona y Mascota. Un MiObjectOutputStream con el método writeStreamHeader() redefinido para que no haga nada y una clase con main() que escribe 10 Personas en un fichero y las lee.

Esta última clase escribe las 5 primeras personas de la forma normal y cierra el fichero. Luego lo vuelve a abrir y escribe otras 5, usando los trucos aquí mencionados: un ObjectOutputStream sin cabecera y el método writeUnshared().

import java.io.Serializable;

/\*\*

\* Clase que contiene el nombre de la mascota y el número de patas.

\*/

public class **Mascota** implements Serializable

{

public String nombre;

public int numeroPatas;

}

import java.io.Serializable;

/\*\*

\* Clase persona con varios datos sobre una persona y su mascota.

\*/

public class **Persona** implements Serializable

{

/\*\*

\* Rellena los campos añadiendo i a unos nombres por defecto

\* @param i Un valor para añadir al final de los nombres

\*/

public Persona (int i)

{

setPersona(i);

}

public String nombre;

public String apellido;

public Mascota mascota=new Mascota();

public int edad;

/\*\* Método para que al meter esta clase en un System.out.println() salga

\* algo legible.

\*/

public String toString()

{

return nombre+" "+

apellido+" de "+

edad+" años tiene como mascota a "+

mascota.nombre+" de "+

mascota.numeroPatas+" patas.";

}

/\*\*

\* Rellena todos los campos de la clase con nombres por defecto a los que

\* añade el número i.

\* @param i

\*/

public void setPersona(int i)

{

nombre="nombre"+i;

apellido="apellido"+i;

mascota.nombre="Fido"+i;

mascota.numeroPatas=i;

edad=i;

}

}

import java.io.IOException;

import java.io.ObjectOutputStream;

import java.io.OutputStream;

/\*\*

\* Redefinicion de la clase ObjectOutputStream para que no escriba una cabecera

\* al principio del Stream.

import java.io.IOException;

import java.io.ObjectOutputStream;

import java.io.OutputStream;

/\*\*

\* Redefinición de la clase ObjectOuputStream para que no escriba una cabecera

\* al inicio del Stream.

\*/

public class **MiObjectOutputStream** extends ObjectOutputStream

{

/\*\* Constructor que recibe OutputStream \*/

public MiObjectOutputStream(OutputStream out) throws IOException

{

super(out);

}

/\*\* Constructor sin parámetros \*/

protected MiObjectOutputStream() throws IOException, SecurityException

{

super();

}

/\*\* Redefinición del método de escribir la cabecera para que no haga nada. \*/

protected void writeStreamHeader() throws IOException

{

}

}

import java.io.EOFException;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

/\*\*

\* Crea y escribe en un fichero 10 objetos de la clase Persona. Luego los lee

\* y los muestra por pantalla.

\*/

public **class EscribirYLeer**

{

/\*\*

\* main del ejemplo. Escribe el fichero y lo lee.

\* @param args se ignoran

\*/

public static void **main**(String[] args)

{

System.out.println(0);

EscribirYLeer eyl = new EscribirYLeer();

eyl.escribeFichero("./mascotas.dat");

System.out.println(1);

eyl.anhadeFichero("./mascotas.dat");

System.out.println(2);

eyl.leeFichero("./mascotas.dat");

System.out.println(3);

}

/\*\*

\* Escribe en el fichero que se le pasa y empezandolo desde cero, 5 objetos

\* de la clase Persona.

\* @param fichero Path completo del fichero que se quiere escribir

\*/

public void **escribeFichero**(String fichero)

{

try

{

ObjectOutputStream oos = new ObjectOutputStream(

new FileOutputStream(fichero));

for (int i = 0; i <5; i++)

{

// ojo, se hace un new por cada Persona. El new dentro

// del bucle.

Persona p = new Persona(i);

oos.writeObject(p);

System.out.println("escrito");

}

oos.close();

} catch (Exception e)

{

e.printStackTrace();

}

}

/\*\*

\* Añade al final del fichero que se le pasa 5 objetos de la clase Persona.

\* @param fichero Path completo del fichero

\*/

public void **anhadeFichero** (String fichero)

{

try

{

// Se usa un ObjectOutputStream que no escriba una cabecera en

// el stream.

MiObjectOutputStream oos = new MiObjectOutputStream(

new FileOutputStream(fichero,true));

// Se hace el new fuera del bucle, sólo hay una instancia de persona.

// Se debe usar entonces writeUnshared().

Persona p = new Persona(5);

for (int i = 5; i < 10; i++)

{

p.setPersona(i); // Se rellenan los datos de Persona.

oos.writeUnshared(p);

System.out.println("añadido");

}

oos.close();

} catch (Exception e)

{

e.printStackTrace();

}

}

/\*\*

\* Se leen todas las Persona en el fichero y se escriben por pantalla.

\* @param fichero El path completo del fichero que contiene las Persona.

\*/

public void **leeFichero**(String fichero)

{

try

{

// Se crea un ObjectInputStream

ObjectInputStream ois = new ObjectInputStream(

new FileInputStream(fichero));

// Se lee el primer objeto

Object aux = ois.readObject();

// Mientras haya objetos

while (aux!=null)

{

if (aux instanceof Persona)

System.out.println(aux);

aux = ois.readObject();

}

ois.close();

}

catch (EOFException e1)

{

System.out.println ("Fin de fichero");

}

catch (Exception e2)

{

e2.printStackTrace();

}

}

}