Serialización de objetos.

De todos es conocida la importancia de poder salvar en memoria secundaria y permanente determinados estados o valores de una aplicación en un momento determinado para ser restablecidos en algún momento posterior.

Si un programador desea guardar permanentemente el estado de un **objeto** puede utilizar las clases vistas hasta el momento para ir almacenando todos los valores de los atributos como valores char, int, byte, etc.

Existe una forma más cómoda de enviar objetos a través de un *stream* como una **secuencia de bytes** para ser almacenados en disco, y también para reconstruir objetos a partir de *streams* de entrada de bytes.

Esto puede conseguirse mediante la **"serialización"** de objetos.

La **serialización** consiste en la transformación de un objeto Java en una secuencia de bytes para ser enviados a un stream.

Mediante este mecanismo pueden almacenarse objetos en ficheros, o se pueden enviar a través de *sockets*, en aplicaciones cliente/servidor de un equipo a otro, por ejemplo.

Para enviar y recibir objetos serializados a través de un ***stream*** se utilizan las clases **java.io.ObjectOutputStream** para la salida y **java.io.ObjectInputStream** para la entrada.

La clase **ObjectOutputStream** necesita como parámetro en su constructor un objeto de la clase **OutputStream**

public ObjectOutputStream(OutputStream out) throws IOException;

Si, por ejemplo, se utiliza como parámetro un objeto de la clase FileOutputStream, los objetos serializados a través de este *stream* serán **dirigidos a un fichero**.

La clase ObjectInputStream necesita como parámetro en su constructor un objeto de la clase InputStream

public ObjectInputStream(InputStream in) throws IOException, StreamCorruptedException;

Si, por ejemplo, se utiliza como parámetro un objeto de la clase FileInputStream, los objetos se deserializan después de obtenerlos a través de este *stream* que, a su vez, **obtieenen los bytes de un fichero.**

Objetos serializables.

Sólo los objetos de clases que implementen la interface **java.io.Serializable** o aquellos que pertenezcan a subclases de clases serializables pueden ser serializados.

La interface **Serializable** **no posee ningún método**. Sólo sirve para "marcar" las clases que pueden ser serializadas. Cuando un objeto es serializado, también lo son todos los objetos alcanzables desde éste, ignorándose todos los atributos static , transient y los no serializables.

No se almacenan los valores de los atributos static porque éstos pertenecen a la clase (no al objeto), y son compartidos por todos los objetos implementados a partir de ésta.

```
public class Persona implements java.io.Serializable {
//Serializable para poder ser escrita en un stream de objetos

   private String nombre;
   private int edad;

   public Persona(String nombre, int edad) {
      this.nombre = nombre;
      this.edad = edad;
   }

   @Override
   public String toString() {
      return ("nombre: " + nombre + ", edad: " + edad);
   }
}
```

Como puede comprobarse en el ejemplo anterior, en el fichero **Persona.java**, se ha declarado la clase Persona de forma que implemente la interface Serializable para poder ser serializada.

Escritura.

Para serializar objetos y escribirlos a través del stream de salida se llama al **método** writeObject() del objeto ObjectOutputStream creado:

También pueden escribirse, además de objetos, valores de tipos de datos simples mediante cualquiera de los métodos de la interface DataOutput, que implementa la clase ObjectOutputStream

Todos estos métodos generan excepciones de la clase IOException.

Veamos un ejemplo en el que se **crea un *stream* de salida**, en el cual se escribirá un objeto de la clase Persona.

```
import java.io.*;
class SerialEscribe {
   public static void main(String arg[]) {
      try {
// Un OutputStream sobre el que escribir los bytes
         FileOutputStream fichero = new FileOutputStream("prueba.dat");
// El objeto serializador
         ObjectOutputStream objs = new ObjectOutputStream(fichero);
         Persona persona1 = new Persona("Victor", 30);
         Persona persona2 = new Persona("Javier", 30);
         objs.writeObject(persona1);
         objs.writeObject(persona2);
         objs.flush(); // vaciar el buffer
         objs.close();
      } catch (IOException e) {
         System.err.println(e);
      }
   }
}
```

En el caso de los objetos persona se utiliza el **método** writeObject().

Lectura.

Para deserializar objetos después de leerlos a través del stream de entrada se llama al método readObject() del objeto ObjectInputStream creado:

El siguiente ejemplo lee y muestra los objetos del fichero prueba.dat:

```
import java.io.*;
class SerialLee {
   public static void main(String arg[]) {
      try {
// Un InputStream del cuál leer los bytes
         FileInputStream fichero = new FileInputStream("prueba.dat");
// El objeto deserializador
         ObjectInputStream objd = new ObjectInputStream(fichero);
         Persona persona1 = (Persona) objd.readObject();
         Persona persona2 = (Persona) objd.readObject();
         System.out.println(persona1);
         System.out.println(persona2);
         objd.close();
      } catch (IOException e) {
      } catch (ClassNotFoundException e) {
         System.err.println(e);
      }
   }
}
```

En los casos de lectura de objetos, **es necesario realizar una conversión de tipo referencial**, ya que el método readObject() devuelve un objeto de la clase Object. Para ello hay que anteponer el nombre de la clase entre paréntesis a la expresión a convertir.

La salida del programa según los datos almacenados en el ejemplo del punto anterior sería la siguiente:

nombre: Victor, edad: 30

nombre: Javier, edad: 30