La serialización de objetos consiste en transformar los atributos de un objeto a datos para un fichero binario, esto nos hace ahorrar tiempo a la hora de programar.

Las clases para realizar una serializacion de objetos son ObjectInputStream para leer y ObjectOutputStream para escribir.

Para crear estos objetos deben pasarle como parámetro un **FileInputStream** para **ObjectInputStream** y **FileOutputStream** para **ObjectOutputStream**.

También debemos hacer que la clase que cree el objeto sea serializable, para ello implementaremos una interfaz llamada **Serializable**, debemos importarla desde el paquete **java.io**.

Vamos a usar la clase Empleado, recordemos esta clase antes, implementando la interfaz que hemos comentado anteriormente:

import java.io.Serializable;

/\*\*

 \* Clase Empleado

 \*

 \* Contiene informacion de cada empleado, es una clase abstracta

 \*

 \* @author Fernando

 \* @version 1.0

 \*/

public class Empleado implements Serializable{

    private static final long serialVersionUID = -2873344211410398459L;

    //Constantes

    /\*\*

     \* Constante SALARIO\_DEF

     \*/

    protected final static double SALARIO\_DEF=600;

    //Atributos

    /\*\*

     \* Nombre del empleado

     \*/

    protected String nombre;

    /\*\*

     \* Apellido del empleado

     \*/

    protected String apellido;

    /\*\*

     \* Edad del empleado

     \*/

    protected int edad;

    /\*\*

     \* Salario del empleado

     \*/

    protected double salario;

    //Metodos publicos

    /\*\*

     \* Devuelve el nombre del empleado

     \* @return nombre del empleado

     \*/

    public String getNombre() {

        return nombre;

    }

    /\*\*

     \* Modifica el nombre de un empleado

     \* @param nombre

     \*/

    public void setNombre(String nombre) {

        this.nombre = nombre;

    }

    /\*\*

     \* Devuelve la edad de un empleado

     \* @return edad del empleado

     \*/

    public int getEdad() {

        return edad;

    }

    /\*\*

     \* Modifica la edad de un empleado

     \* @param edad

     \*/

    public void setEdad(int edad) {

        this.edad = edad;

    }

    /\*\*

     \* Devuelve el salarioBase

     \* @return salarioBse

     \*/

    public double getSalario() {

        return salario;

    }

    /\*\*

     \* Modifica el salarioBase de los empleados

     \* @param salarioBase

     \*/

    public  void setSalario(double salario) {

        this.salario = salario;

    }

    public boolean plus (double sueldoPlus){

        boolean aumento=false;

        if (edad>40){

            salario+=sueldoPlus;

            aumento=true;

        }

        return aumento;

    }

    public boolean equals (Empleado a){

        if(a.getNombre().equals(nombre) && a.getApellido().equals(apellido)){

            return true;

        }else{

            return false;

        }

    }

    public int compareTo(Empleado a){

            int estado=-1;

            if(this.edad==a.getEdad()){

                //Los objetos son iguales

                estado=0;

            }else if(this.edad>a.getEdad()){

                //El objeto 1 es mayor que la pasada por parametro

                estado=1;

            }

            return estado;

    }

    public String toString (){

        String mensaje="El empleado se llama "+nombre+" "+apellido+" con "+edad+" años " +

                "y un salario de "+salario;

        return mensaje;

    }

    public String getApellido() {

        return apellido;

    }

    //Constructores

    /\*\*

     \* Constructor por defecto

     \*/

    public Empleado(){

        this ("", "", 0, SALARIO\_DEF);

    }

    /\*\*

     \* Constructor con 2 parametros

     \* @param nombre nombre del empleado

     \* @param apellido nombre del empleado

     \*/

    public Empleado(String nombre, String apellido){

        this (nombre, apellido, 0, SALARIO\_DEF);

    }

    /\*\*

     \* Constructor con 3 parametros

     \* @param nombre nombre del empleado

     \* @param apellido nombre del empleado

     \* @param edad edad del empleado

     \*/

    public Empleado (String nombre, String apellido, int edad){

        this (nombre, apellido, edad, SALARIO\_DEF);

    }

    /\*\*

     \* Constructor con 4 parametros

     \* @param nombre nombre del empleado

     \* @param apellido nombre del empleado

     \* @param edad edad del empleado

     \* @param salario salario del empleado

     \*/

    public Empleado(String nombre, String apellido, int edad, double salario){

        this.nombre=nombre;

        this.apellido=apellido;

        this.edad=edad;

        this.salario=salario;

    }

}

Veremos que cuando implementamos la interfaz **Serializable,** el nombre de la clase aparecerá subrayada en amarillo, indicando un warning, si pasamos el ratón por encima veremos una opción llamada **“Add generated serial version ID”** y nos añade un atributo con una serie de números, este atributo nos sirve para evitar problemas de deserializacion en un futuro, te recomiendo que siempre añadas este código.

Ahora escribiremos en un fichero binario, los datos de un objeto de la clase Empleado con ObjectOutputStream, usaremos el método **writeObject(objeto\_serializable)**.

import java.io.\*;

public class SerializacionApp {

    public static void main(String[] args) {

        //Creamos el objeto

        Empleado empleado1=new Empleado("Fernando", "Ureña", 23, 800);

        Empleado empleado2=new Empleado("Antonio", "Lopez", 35, 1000);

        try(ObjectOutputStream oos=new ObjectOutputStream(new FileOutputStream("D:\\empleados.ddr"))){

            //Escribimos en un fichero

            oos.writeObject(empleado1);

            oos.writeObject(empleado2);

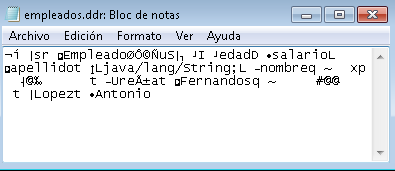
        }catch(IOException e){

        }

    }

}

Si abrimos el bloc de notas, veremos que ha escrito los datos del objeto.

[](http://www.discoduroderoer.es/wp-content/uploads/2013/07/resultado-escritura-objeto-fichero-binario-.png)

Si queremos que no se lea un atributo de una clase, deberemos usar la palabra reservada **transient.**

Ahora vamos a leer los datos del fichero binario con **ObjectInputStream**, usaremos el método **readObject()**, este devolverá un objeto que este leyendo, en nuestro caso, un objeto Empleado, por lo que debemos hacer un casting.

import java.io.\*;

public class SerializacionApp {

    public static void main(String[] args) {

        try(ObjectInputStream ois=new ObjectInputStream(new FileInputStream("D:\\empleados.ddr"))){

            //Cuando no haya mas objetos saltara la excepcion EOFException

            while(true){

                Empleado aux=(Empleado)ois.readObject();

                System.out.println(aux.getNombre());

                System.out.println(aux.getApellido());

                System.out.println(aux.getEdad());

                System.out.println(aux.getSalario());

                System.out.println("");

            }

        }catch(ClassNotFoundException e){

        }catch(EOFException e){

        }catch(IOException e){

        }

    }

}

También debemos controlar la excepción **ClassNotFoundException,** que saltara la excepción.

Debemos tener en cuenta que es posible que experimentemos un problema en el caso de que insertemos un objeto más a un fichero binario que contenga objetos serializados, este no sera leído. La pregunta es **¿Porque?** La respuesta es que cuando escribimos un objeto en un fichero binario **ObjectOutputStream** crea una cabecera al principio y cuando cerramos el fichero y volvemos a añadir un objeto crea de nuevo una cabecera, haciendo que la información a partir de esta no pueda ser leída.

Para conseguir que no cree esta cabecera debemos crear una versión de la clase **ObjectOutputStream**, sobrescribiendo el método que crea esta cabecera, es más fácil de lo que parece. Veamos como seria:

import java.io.\*;

//Esta clase hereda sus propiedades de ObjectOutputStream

public class MiObjectOutputStream extends ObjectOutputStream  {

    //Sobrescribimos el método que crea la cabecera

    protected void writeStreamHeader() throws IOException{

        // No hacer nada.

    }

    //Constructores

    public MiObjectOutputStream () throws IOException{

        super();

    }

    public MiObjectOutputStream(OutputStream out) throws IOException{

                super(out);

        }

}

La idea es que si el fichero binario existe, usaremos la clase original y si existe usamos nuestra clase creada antes. Para indicar si existe o no un fichero, se puede usar el método **exists()** de la clase **File.**