La interfaz Serializable

La interfaz *Serializable* aporta a las clases que la implementan la capacidad de persistencia de sus objetos, es decir, la posibilidad de que dichos objetos puedan perdurar a lo largo del tiempo con la posibilidad de grabarlos en un dispositivo de almacenamiento o transferirlos a través de la red.

Vamos a crear una clase llamada Alumno con capacidad de persistencia:

```
Import java.io.Serializable;

public class Alumno implements Serializable {

}
```

Si ya has creado la clase, comprobarás que Eclipse te está dando una alerta, la palabra Alumno está subrayada en amarillo y, si sitúas el puntero de ratón durante un rato sobre la palabra Alumno, obtendrás un cuadro con información sobre la alerta y las soluciones propuestas por Eclipse.

La alerta nos dice lo siguiente: "*The serializable class Alumno does not declare a static final serial Version UID fields of type long*". Está claro que nos está pidiendo que declaremos una variable llamada *serialVersionUID*.

¿Qué es serialVersionUID?

La *serialVersionUID* es el número de versión de la clase, y se utiliza para evitar problemas de incompatibilidad de versión en los procesos de serialización y deserialización entre los programas que hacen de emisor y receptor del objeto.

- Tenemos un programa A con una clase Alumno que cuenta con las propiedades nombre, edad y telefono. Dentro de la clase Principal creamos un objeto Alumno y lo guardamos en un archivo llamado datos.dat. El programa A es el que realiza la serialización y por lo tanto el emisor del objeto guardado.
- Tenemos otro programa B, donde hemos copiado la clase Alumno, pero está vez se nos ha ocurrido añadir una propiedad más llamada domicilio. Los programas A y B tienen distinta versión de la clase Alumno.
- Ahora desde la clase Principal del programa B recuperamos el objeto Alumno que previamente guardamos en el archivo datos.dat durante la ejecución del programa A. El programa B debe realizar la deserialización del objeto guardado y por lo tanto será el receptor de dicho objeto.
- El programa B nos arroja una excepción, ya que intenta recuperar un objeto construido a partir de la primera versión de la clase Alumno, sin embargo, el programa B contiene la segunda versión de la clase Alumno, que resulta incompatible.
- Las versiones primera y segunda de la clase Alumno deberían tener distinto *serialVersionUID* para distinguir rápidamente que se trata de versiones distintas de la misma clase. De esta forma, en el proceso de deserialización, la máquina virtual de Java comparará la serialVersionUID del objeto guardado con la serialVersionUID de la clase Alumno que contiene el programa B, arrojando una excepción de tipo InvalidClassException.

Volvamos a poner de nuevo la atención en la clase Alumno.

Vuelve a situar el puntero del ratón sobre el nombre de la clase hasta que salga el cuadro informativo y selecciona la opción "*addgenerated serial version ID*". Verás que se añade automáticamente la variable serialVersionUID con un valor auto calculado.

import java.io.Serializable;

public class Alumno implements Serializable {

private static final long serialVersionUID = 4854486451470258537L;

}

Grabar un objeto en disco

En este apartado veremos cómo grabar un objeto de la clase *Alumno* en un archivo.

Paraempezar,debescompletarelcódigodelaclase*Alumno*copiandoypegandoelsiguientecódigo:

```
import java.io.Serializable;
import java.util.ArrayList;

public class Alumno implements Serializable {
	private static final long serialVersionUID = 4854486451470258537L;
	private String nombre;
	private int edad;
	private ArrayList<Calificacion> calificaciones;

	public Alumno(String nombre, int edad) {
		this.nombre = nombre;
		this.edad = edad;
		this.calificaciones = new ArrayList<Calificacion>();
	}
	public void calificar(String asignatura, int nota) {
		this.calificaciones.add(new Calificacion(asignatura, nota));
	}

	public String getNombre() {
		returnnombre;
	}
	public int getEdad() {
		return edad;
	}
	public ArrayList<Calificacion>getCalificaciones() {
		return calificaciones;
	}
}
```

Como puedes comprobar por el código, un objeto *Alumno* está formado por las propiedades *nombre*, *edad* y una colección de objetos *Calificacion*.

La clase *Calificacion* tiene la siguiente implementación:

```
import java.io.Serializable;

public class Calificacion implements Serializable {
	private static final long serialVersionUID = 3057545624874202352L;

	private String asignatura;
	private int nota; // Sobre 100

	public Calificacion(String asignatura, int nota) {
		this.asignatura = asignatura;
		this.nota = nota;
	}

	@Override
	public String toString() {
		return "Calificación [Asignatura="+asignatura+",Nota="+nota+"]";
	}
}
```

Ya tenemos todo listo para construir un objeto *Alumno* y persistirlo grabándolo en un archivo.

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class Principal {
	public static void main(String args[]) {
		//Crear objeto Alumno
		Alumno alu1=new Alumno("Pedro",25);
		alu1.calificar("Matemáticas",50);
		alu1.calificar("Inglés",75);
		alu1.calificar("Informática",95);
		alu1.calificar("Lengua",60);

		//Abrir fichero para escritura
		FileOutputStream file;
		ObjectOutputStream buffer;
		try {
			file = new FileOutputStream("J:\\alumno.dat");
			buffer = new ObjectOutputStream(file);
		} catch (IOException e) {
			System.out.println("No se ha podido abrir el fichero");
			System.out.println(e.getMessage());
			return;
		}
```

```
                //Guarda objeto en el fichero alumno.dat
                try{
                        buffer.writeObject(alu1);
                        System.out.println("El objeto se ha grabado con éxito");
                }catch(IOExceptione){
                        System.out.println("Error al escribir en el fichero");
                        System.out.println(e.getMessage());
                }

                //Cerrar el fichero
                try{
                        buffer.close();
                        file.close();
                } catch (IOException e) {
                        System.out.println("Error al cerrar el fichero");
                        System.out.println(e.getMessage());
                }

        }
}
```

Recuperar un objeto

```
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.IOException;

public class Principal {
        public static void main(String args[]) throws ClassNotFoundException {

                // Abrir fichero para lectura
                FileInputStream file;
                ObjectInputStream buffer;
                try {
                        file = new FileInputStream("J:\\alumno.dat");
                        buffer = new ObjectInputStream(file);
                } catch (IOException e) {
                        System.out.println("No se ha podido abrir el fichero");
                        System.out.println(e.getMessage());
                        return;
                }

                // Lee el objeto guardado en el archivo alumno.dat
                try {
                        Alumno alu1 = (Alumno) buffer.readObject();
                        System.out.println("Nombre del alumno: " + alu1.getNombre());
```

```
                        System.out.println("Edad: " + alu1.getEdad());
                        for (Calificacion c : alu1.getCalificaciones()) {
                                System.out.println(c);
                        }
                } catch (IOException e) {
                        System.out.println("Error al escribir en el fichero");
                        System.out.println(e.getMessage());
                }

                // Cerrar el fichero
                try {
                        buffer.close();
                        file.close();
                } catch (IOException e) {
                        System.out.println("Error al cerrar el fichero");
                        System.out.println(e.getMessage());
                }
        }
}
```

El modificador transient

El modificador *transient* se utiliza con clases serializables para indicar las propiedades que no queremos que sean serializadas, es decir, las que no deseamos que se guarden. Tiene sentido con algunas propiedades, tales como un *password*.

Grabar y recuperar varios objetos

En este apartado veremos cómo podemos guardar en un fichero varios objetos del mismo tipo y cómo podemos posteriormente leerlos controlando el final del fichero.

Como ejemplo, vamos a crear una agenda de contactos. En primer lugar debes crear un nuevo proyecto en Eclipse y una clase *Contacto*, que representará a cada uno de los contactos que queremos guardar en la agenda.

```
import java.io.Serializable;

public class Contacto implements Serializable {
        private static final long serialVersionUID = -4624046047796483183L;

        private String nombre;
        private String telefono;

        public Contacto(String nombre, String telefono) {
                this.nombre=nombre;
                this.telefono=telefono;
        }

        public String getNombre() {
```

```
		return nombre;
	}
	public String getTelefono() {
		return telefono;
	}

	@Override
	public String toString(){
		return"Contacto["+nombre+"-"+telefono+"]";
	}
}
```

Ahora, en la clase *Principal*, guardarás tres contactos en el fichero *agenda.dat.*

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class Principal {

	public static void main(String[] args) {
		//Abrimos fichero para escritura
		FileOutputStreamfile;
		ObjectOutputStream buffer;
		try {
			file = new FileOutputStream("agenda.dat", true);
			buffer = new ObjectOutputStream(file);
		} catch (IOException e) {
			System.out.println("No se ha podido abrir el fichero");
			System.out.println(e.getMessage());
			return;
		}

		//Creamos tres contactos
		Contacto c1=new Contacto("Amelia","913670542");
		Contacto c2=new Contacto("Federico","6166644422");
		Contacto c3=new Contacto("Carmen","639888777");

		//Guardamos los tres contactos en agenda.dat
		try {
			buffer.writeObject(c1);
			buffer.writeObject(c2);
			buffer.writeObject(c3);
			System.out.println("Los contactos se han guardado con éxito");
		}catch(IOExceptione){
			System.out.println("Error al escribir en el fichero");
			System.out.println(e.getMessage());
```

```
                }

                //Cerrar el fichero
                try{
                        buffer.close();
                        file.close();
                } catch (IOException e) {
                        System.out.println("Error al cerrar el fichero");
                        System.out.println(e.getMessage());
                }
        }

}
```

Leer contactos hasta que sea final de fichero

Ahora vamos a realizar un listado de contactos, para lo cual debes crear un nuevo proyecto con el nombre que desees y copiar la clase Contacto del proyecto anterior. Para leer cada contacto debes utilizar el método readObject(), en el momento en que sea final de fichero, el método readObject() lanzará la excepción EOFException.

```
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.IOException;

public class Principal {
        public static void main(String args[])  {

                // Abrimos fichero agenda.dat para lectura
                FileInputStream file;
                ObjectInputStream buffer;
                try {
                        file = new FileInputStream("agenda.dat");
                        buffer = new ObjectInputStream(file);
                } catch (IOException e) {
                        System.out.println("No se ha podido abrir la agenda de contactos");
                        System.out.println(e.getMessage());
                        return;
                }

                // Leemos la lista de contactos
                boolean eof = false;
                Contacto c;
                while (!eof) {
                        try {
                                c = (Contacto) buffer.readObject();
                                System.out.println(c);
                        } catch (EOFException e1) {
```

```
                                        eof = true;
                                } catch (IOException e2) {
                                        System.out.println("Error al leer los contactos de la agenda");
                                        System.out.println(e2.getMessage());
                                } catch (ClassNotFoundException e3) {
                                        System.out.println("La clase Contacto no está cargada en memoria");
                                        System.out.println(e3.getMessage());
                                }
                        }

                        // Cerramos el fichero
                        try {
                                buffer.close();
                                file.close();
                        } catch (IOException e) {
                                System.out.println("Error al cerrar el fichero");
                                System.out.println(e.getMessage());
                        }

                }
}
```