Tema 12: Almacenamiento de objetos en ficheros. Serialización

¿Qué aprenderás?

- Guardar objetos en ficheros como una secuencia de bytes.
- Recuperar esta secuencia de bytes y transformarlas en objetos.

¿Sabías que…?

- Con la serialización podemos crear objetos sin ejecutar el constructor de la clase.
- El concepto marshalling se suele usar como sinónimo de serialización, pero no es lo mismo. Con el marshalling se almacena el estado de un objeto junto con su código.

12. Almacenamiento de objetos en ficheros. Serialización

En lecciones anteriores has visto las herramientas que ofrece Java para trabajar con ficheros. Has visto su utilidad para ofrecer persistencia y poder así guardar los datos con los que trabaja nuestra aplicación. Estos datos acostumbraban a ser cadenas de caracteres y otros tipos de datos simples.

Ahora hemos hecho aplicaciones que trabajan con clases y objetos. ¿Es posible guardar un objeto entero en un fichero? La respuesta es sí. Para ello vamos a ver el concepto de serialización.

12.1. Serialización

La serialización es un proceso mediante el cual un objeto es transformado en una secuencia de bytes que representa el estado de dicho objeto, es decir, el valor de sus atributos, para luego ser guardado en un fichero, ser enviado por la red, etc. Un objeto serializado se puede recomponer luego sin ningún problema.

Para hacer que una clase pueda ser serializable hay que hacer que implemente la interfaz Serializable. Esta interfaz no define ningún método, por lo que no tendremos que implementar nada nuevo en nuestra clase.

```
import java.io.Serializable;
public class Empleado implements Serializable{
    private String nombre;
    private int edad;
    private double sueldo;

    public Empleado(String nombre, int edad, double sueldo) {
            super();
            this.nombre = nombre;
            this.edad = edad;
            this.sueldo = sueldo;
    }
    public String getNombre() {
            return nombre;
    }
    public void setNombre(String nombre) {
            this.nombre = nombre;
    }
    public int getEdad() {
            return edad;
    }
    public void setEdad(int edad) {
            this.edad = edad;
    }
    public double getSueldo() {
            return sueldo;
    }
    public void setSueldo(double sueldo) {
            this.sueldo = sueldo;
    }
    @Override
    public String toString() {
            return "Empleado [nombre=" + nombre + ", edad=" + edad + ", sueldo=" + sueldo + "]";
    }
}
```

Declaración de una clase serializable

Una vez la clase se ha marcado como serializable, ya podemos guardar los objetos de esta clase en un fichero como una secuencia de bytes. Java se encarga de este proceso.

12.1.1. **Clase ObjectOutputStream y clase ObjectInputStream**

La clase ObjectOutputStream es la que usaremos para guardar objetos en un fichero. Por su parte, la clase ObjectInputStream será la encargada de recuperar los datos de los objetos del fichero.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class Principal {
    public static void main(String[] args) throws FileNotFoundException, IOException,
    ClassNotFoundException {
            Empleado e1 = new Empleado("e1", 44, 30000);
            File archivo = new File("Empleados");
            //ESCRIBIR EN EL ARCHIVO
            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(archivo));
            oos.writeObject(e1);
            oos.close();
            //LEER EL ARCHIVO
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(archivo));
            Empleado e2 = (Empleado) ois.readObject();
            ois.close();
            System.out.println(e2);
    }
}
```

Ejemplo de escritura y lectura de un objeto serializable en un fichero

En el código anterior vemos que creamos un objeto de la clase Empleado que se ha implementado antes y que se ha definido como serializable. Es importante destacar este hecho ya que, si intentásemos guardar en un fichero un objeto que no se haya definido como serializable, obtendríamos un error de compilación.

Para guardar el objeto en el fichero "Empleados" se crea un objeto de la clase ObjectOutputStream sobre el fichero, y se llama al método writeObject, pasándole como parámetro el objeto a guardar.

Para leer el objeto del fichero, se crea un objeto de la clase ObjectInputStream, y se obtienen los datos del empleado con el método readObject. Para realizar la asignación se hace un casting indicando entre paréntesis que los datos leídos deben convertirse en un objeto de tipo Empleado.

No hay que olvidarse de cerrar los flujos de salida y entrada de datos una vez ya hemos escrito o leído todo lo que queríamos. Para ello usamos el método close que está implementado tanto en la clase ObjectOutputStream como en la clase ObjectInputStream.

12.1.2. **El modificador transient**

En algunas ocasiones querremos que uno o varios atributos de una clase serializable no se incluyan en la secuencia de bytes que representa el estado del objeto. Es decir, no queremos que se guarde el valor de un atributo. Pensemos, por ejemplo, en el caso de guardar los datos de los usuarios en un fichero. ¿Debemos guardar también sus contraseñas?

Para que el valor de un atributo no se guarde cuando serializamos un objeto usaremos el modificador transient.

```
import java.io.Serializable;
public class Empleado implements Serializable{
    private String nombre;
    private int edad;
    private double sueldo;
    private transient String clave = "12345";
    public Empleado(String nombre, int edad, double sueldo) {
            super();
            this.nombre = nombre;
            this.edad = edad;
            this.sueldo = sueldo;
    }
    …
}
```

Ejemplo de uso del modificador transient

En el código anterior vemos el atributo clave de la clase Empleado declarado como transient. Esto hace que, en caso de que un empleado se serializase, no se guardaría el valor del atributo clave.

12.1.3. **Serialización a medida**

Hemos visto que Java se encarga del proceso de serialización de un objeto, es decir, del proceso de convertirlo en una secuencia de bytes. Pero también tenemos la posibilidad de controlar nosotros esta acción. Para ello disponemos de los métodos readObject y writeObject que tendremos que añadir en la clase serializable e implementar en ellos las acciones que queremos que se realicen en el proceso de serialización del objeto.

Recursos y enlaces

- Documentación oficial sobre la serialización: https://docs.oracle.com/javase/10/docs/api/java/io/Serializable.html

Conceptos clave

- **Serialización:** proceso mediante el cual un objeto es convertido en una secuencia de bytes para poder ser guardado en un fichero, enviado por la red, etc.
- **Serializable**: interface que hace que los objetos de una clase puedan ser serializados.
- **transient**: modificador que se puede añadir a un atributo de una clase serializable que hace que el valor de este no se incluya en la secuencia de bytes del objeto.

Test de autoevaluación

¿Qué clases utilizaremos para el almacenamiento y la recuperación de objetos en ficheros?

a) DataInputStream y DataOutputStream
b) FileInputStream y FileOutputStream
c) FilterInputStream y FilterOutputStream
d) ObjectInputStream y ObjectOutputStream

¿Qué tendremos que añadir en la cabecera de las clases que queremos serializar?

a) extends java.io.Serializable
b) implements java.io.Serializable
c) extends java.io.transient
d) implements java.io.transient

¿Qué modificador se utiliza para hacer que un atributo de una clase no sea serializable?

a) protected serializable
b) static-serializable
c) no-serializable
d) transient

Ponlo en práctica

Actividad 1

Crea la clase Contacto con los atributos nombre (cadena de caracteres) y teléfono (long). Implementa el constructor, los métodos getters y setters, y el método toString. Esta clase debe ser serializable.

En el programa principal, crea una ArrayList y almacena en él 4 contactos. A continuación, deberás crear el archivo “Agenda” y recorrer el ArrayList, guardando los objetos que contiene dentro del archivo.

Por último, lee el archivo para recuperar los contactos, y muestra por pantalla el total de contactos recuperados y sus datos.

SOLUCIONARIOS

Test de autoevaluación

¿Qué clases utilizaremos para el almacenamiento y la recuperación de objetos en ficheros?

a) DataInputStream y DataOutputStream
b) FileInputStream y FileOutputStream
c) FilterInputStream y FilterOutputStream
d) ObjectInputStream y ObjectOutputStream

¿Qué tendremos que añadir en la cabecera de las clases que queremos serializar?

e) extends java.io.Serializable
f) implements java.io.Serializable
g) extends java.io.transient
h) implements java.io.transient

¿Qué modificador se utiliza para hacer que un atributo de una clase no sea serializable?

i) protected serializable
j) static-serializable
k) no-serializable
l) transient

Ponlo en práctica

Actividad 1

Crea la clase Contacto con los atributos nombre (cadena de caracteres) y teléfono (long). Implementa el constructor, los métodos getters y setters, y el método toString. Esta clase debe ser serializable.

En el programa principal, crea una ArrayList y almacena en él 4 contactos. A continuación, deberás crear el archivo “Agenda” y recorrer el ArrayList, guardando los objetos que contiene dentro del archivo.

Por último, lee el archivo para recuperar los contactos, y muestra por pantalla el total de contactos recuperados y sus datos.

El solucionario está disponible en la versión interactiva del aula.