El futuro digital es de todos | MinTIC

UNIVERSIDAD EL BOSQUE | MisiónTIC 2022

Ciclo 2

Semana 5

Persistencia en Archivos

Lectura 2 - Persistencia en archivos binarios

Convenio de asociación no. 777 de 2021 suscrito entre el Fondo Único de Tecnologías de la Información y las Comunicaciones y la Universidad el Bosque.

Persistencia en archivos binarios

Cuando ejecutamos una aplicación OO lo normal es crear múltiples instancias de las clases que tengamos definidas en el sistema. Cuando cerramos esta aplicación todos los objetos que tengamos en memoria se pierden.

Para solucionar este problema los lenguajes de POO nos proporcionan unos mecanismos especiales para poder guardar y recuperar el estado de un objeto y de esa manera poder utilizarlo como si no lo hubiéramos eliminado de la memoria. Este tipo de mecanismos se conoce como persistencia de los objetos.

En Java hay que implementar una interfaz y utilizar dos clases:

- Interfaz Serializable (interfaz vacía, no hay que implementar ningún método)
- Streams: ObjectOutputStream y ObjectInputStream.

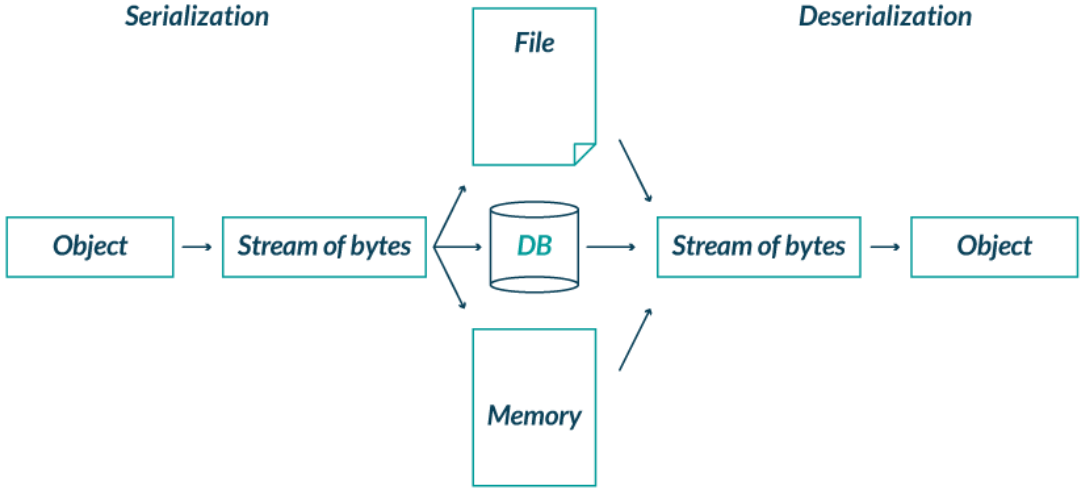
Por ejemplo:

class Clase *implements* Serializable

a partir de esta declaración los objetos que se basen en esta clase pueden ser persistentes.

ObjectOutputStream y ObjectInputStream permiten leer y escribir grafos de objetos, es decir, escribir y leer los bytes que representan al objeto. El proceso de transformación de un objeto en un stream de bytes se denomina serialización.

Los objetos ObjectOutputStream y ObjectInputStream deben ser almacenados en ficheros, para hacerlo utilizaremos los streams de bytes FileOutputStream y FileInputStream, ficheros de acceso secuencial.



Para serializar objetos necesitamos:

- Un objeto FileOutputStream que nos permita escribir bytes en un fichero como, por ejemplo:

  FileOutputStream fos = new FileOutputStream("fichero.dat");

- Un objeto ObjectOutputStream al que le pasamos el objeto anterior de la siguiente forma:

  ObjectOutputStream oos = new ObjectOutputStream(fos);

- Almacenar objetos mediante writeObject() como sigue:

  oos.writeObject(objeto);

- Cuando terminemos, debemos cerrar el fichero escribiendo:

  fos.close();

- Los atributos static no se serializan de forma automática.
- Los atributos que pongan transient no se serializan.

Para recuperar o deserializar los objetos necesitamos:

- Un objeto FileInputStream que nos permita leer bytes de un fichero, como por ejemplo:

  FileInputStream fis = new FileInputStream("fichero.dat");

- Un objeto ObjectInputStream al que le pasamos el objeto anterior de la siguiente forma:

  ObjectInputStream ois = new ObjectInputStream(fis);

- Leer objetos mediante readObject() como sigue:

  (ClaseDestino) ois.readObject();

  Necesitamos realizar una conversión a la "ClaseDestino" debido a que Java solo guarda Objects en el fichero.
- Cuando terminemos, debemos cerrar el fichero escribiendo:

  fis.close();

Ejemplo de serialización y deserialización (Versión 1)

```
public class Persona implements Serializable { … }
class Fecha implements Serializable { … }
/*****************************************************************************************************/
Persona obj1 = new Persona( "06634246S", "Javier", f1, "calle1"); …
Persona obj4 = new Persona( "15664386T", "Carmen", f4, "calle4");
/*****************************************************************************************************/
//Serialización de las personas 1
FileOutputStream fosPer = new FileOutputStream("copiasegPer.dat");
ObjectOutputStream oosPer = new ObjectOutputStream(fosPer);
oosPer.writeObject(obj1); … oosPer.writeObject(obj4);
/*****************************************************************************************************/
//Lectura de los objetos de tipo persona
FileInputStream fisPer  = new FileInputStream("copiasegPer.dat");
ObjectInputStream oisPer = new ObjectInputStream(fisPer);
try {
                while (true) {
                                Persona per = (Persona) oisPer.readObject();
                                System.out.println (per.toString());
                }
} catch (EOFException e) {
        System.out.println ("Lectura de los objetos de tipo Persona finalizada");
}
fisPer.close();
```

Ejemplo de serialización y deserialización (Versión 2)

```
public class Persona implements Serializable { … }
class Fecha implements Serializable { … }
/*****************************************************************************************************/
Persona obj1 = new Persona( "06634246S", "Javier", f1, "calle1”); …
Persona obj4 = new Persona( "15664386T", "Carmen", f4, "calle4”);
//Introducimos los objetos en una tabla hash
HashMap<String, Persona> personas = new HashMap<String, Persona>();
personas.put(obj1.getDni(), obj1); …
personas.put(obj4.getDni(), obj4);
/*****************************************************************************************************/
//Serialización de la tabla hash personas
FileOutputStream fosPer = new FileOutputStream("copiasegPer.dat");
ObjectOutputStream oosPer = new ObjectOutputStream(fosPer);
oosPer.writeObject(personas);
fosPer.close();
/*****************************************************************************************************/
//Lectura de los objetos de tipo persona a través de la tabla hash personas
FileInputStream fisPer = new FileInputStream("copiasegPer.dat");
ObjectInputStream oisPer = new ObjectInputStream(fisPer);
try {
        while (true) {
                personas = (HashMap) oisPer.readObject();
                System.out.println (personas.toString());
        }
} catch (EOFException e) {
        System.out.println ("Lectura de los objetos de tipo Persona finalizada");
}
fisPer.close();
```