Persistencia y Serialización en Java con Archivos

Agustín J. González
ELO-329

Persistencia en Java

- Un objeto se dice persistente cuando es almacenado en un archivo u otro medio permanente. Un programa puede grabar objetos persistentes y luego recuperarlos en un tiempo posterior.
- A diferencia de C++ que sólo soporta persistencia a través de bibliotecas propietarias por lo cual su portabilidad y generalidad es limitada, Java se provee un mecanismo de serialización para almacenar objetos en disco.
- La serialización se obtiene llamando al método writeObject de la clase ObjectOutputStream para grabar el objeto, para recuperarlo llamamos al método readObject de la clase ObjectInputStream.
- La serialización además de persistencia, se puede usar para transferir objetos desde una máquina a otra a través de un socket (ELO330).

Interfaz Serializable

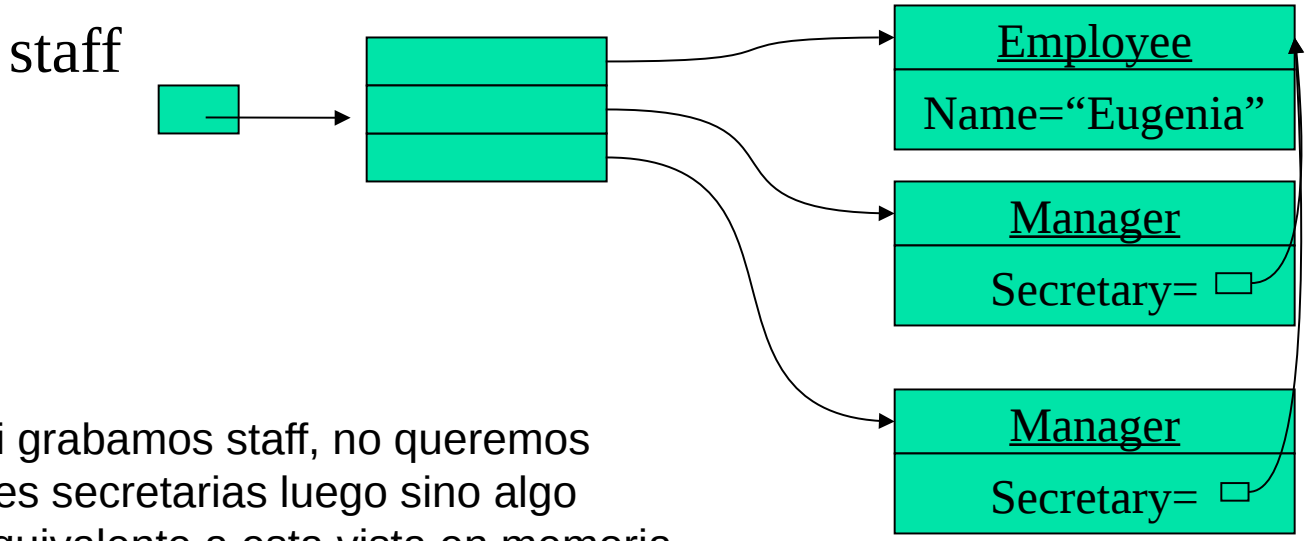
- Sólo objetos que implementen la interfaz Serializable pueden ser escritos a stream. La clase de cada objeto es codificada incluyendo el nombre de la clase y la firma de la clase (su prototipo) los valores de los sus campos y arreglos, y la clausura de cualquier otro objeto referenciado desde el objeto inicial.
- Para hacer que un objeto sea serializable, sólo debemos declarar que implementa la interfaz serializable. Nada más. No hay métodos que debamos definir.
- Por razones de seguridad las clases no son serializable por defecto. por ejemplo, no tiene sentido guardar en archivo una referencia a un archivo abierto.
- Hay que tener claro el orden y tipo de los objetos almacenados en disco para recuperarlos en el mismo orden.

Ejemplo: Empleados serializables

- Class Employee implements Seralizable {....}
  Employee staff = new Employee[3];
  ....
  out.writeObject(staff);
- Luego podemos recuperar el objeto haciendo:
  Employee[ ] newStaff=(Employee[ ])in.readObject();
- Sólo objetos pueden ser serializados con writeObject().
- Veamos el ejemplo: ObjectFileTest.java

Tratamiento de referencia a objetos

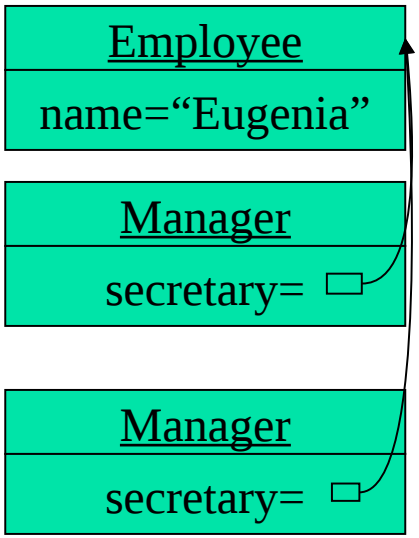
- Múltiples referencias a un único objeto son codificadas usando un mecanismo de referencias compartidas de modo que el “grafo” de objetos puede ser restaurado con la misma forma original.
- Los métodos writeObject y readObject se encargan de crear y almacenar un número de “serie” para cada objeto. De este modo objetos ya almacenados no son grabados nuevamente.
- Supongamos que dada manager tiene una secretaria. Dos manager podrían compartir la secretaria, en este caso tendríamos algo como:



- Si grabamos staff, no queremos tres secretarias luego sino algo equivalente a esta vista en memoria.

Tratamiento de referencia a objetos

- Para ello Java utiliza el siguiente algoritmo para serializar (poner número de series).
- A todos los objetos grabados a discos se les asigna un número de serie.
- Antes de grabar un objeto a disco se ve si ya ha sido grabado.
- Si ya ha sido grabado, se graba “lo mismo que el objeto con número de series xxx”
- Sino, se almacena el objeto.



En disco:

Serial number=1
type=Employee
name=“Eugenia”

Serial number=2
type=Manager
secretary=objeto 1

Serial number=3
type=Manager
secretary=objeto 1

Mezcla de objetos serializables y datos básicos

- Podemos hacerlo por medio de los métodos writeInt, readInt, etc dado que ObjectOutputStream implementa la interfaz DataOutput. Análogo para la entrada. De datos.
- Ejemplo: para escribir un objeto,

```
FileOutputStream ostream = new FileOutputStream("t.tmp");
    ObjectOutputStream p = new ObjectOutputStream(ostream);
    p.writeInt(12345);
    p.writeObject("Today");
    p.writeObject(new Date());
    p.flush();
    ostream.close();
```

- La lectura se hace en forma análoga.

```
FileInputStream istream = new FileInputStream("t.tmp");
    ObjectInputStream p = new ObjectInputStream(istream);
    int i = p.readInt();
    String today = (String)p.readObject();
    Date date = (Date)p.readObject();
    istream.close();
```

Cuando hay objetos no serializables

- Clases que requieren manejos especiales durante el proceso de serialización o deserialización deben implementar los métodos:

```
private void readObject(java.io.ObjectInputStream stream)
    throws IOException, ClassNotFoundException;
private void writeObject(java.io.ObjectOutputStream stream)
    throws IOException
```

- Se aplica en casos que tengamos objetos que no sean serializables (aquellos que tienen algún dato no serializable)
- Por ejemplo Point2D.Double no es serializable en Java.
- Para que no reclame el compilador, definimos nuestro dato Point2D.Double como transiente (transient) y luego definimos los métodos indicados.

Cuando hay objetos no serializables

- Ejemplo:

```
public class LabelPoint {
        ....
Private String label;
private transient Point2D.Double point;
}
```

- Luego implementamos:

```
private void writeObject(ObjectOutputStream out) throws IOException  {
  out.defaultWriteObject();
  out.writeDouble(point.getX());
  out.writeDouble(point.getY());
}
private void readObject(ObjectInutStream in) throws IOException {
  in.defaultReadObject();
  double x=in.readDouble();
  double y=in.writeDouble();
  point =new Point2D.Double(x,y);
}
```