Nell'informatica e in particolare nell'ambito del salvataggio e della trasmissione dei dati, la serializzazione è il processo mediante il quale è possibile convertire una struttura dati e/o un insieme di oggetti in una sequenza di bits/bytes affinché possano essere salvati in un file, in un buffer di memoria oppure possano essere trasmessi attraverso una connessione di rete per essere successivamente recuperati e ricostruiti nell'ambiente software di un qualsiasi computer.

La rilettura della sequenza di bits/bytes risultante, secondo le specifiche della serializzazione, permette di creare un clone semanticamente identico all'oggetto originale.

Pertanto, possiamo affermare che la serializzazione fornisce un metodo per garantire la persistenza degli oggetti che, per certi aspetti, è più conveniente di alcuni metodi tradizionali come, ad esempio, lo scrivere le proprietà degli oggetti in un file su disco per rileggerle e riassemblarle successivamente. Inoltre, essa fornisce un modo per rispondere a "Chiamate di procedure remote" (Remote Procedure Calls o RPC) come ad esempio in SOAP (Simple Object Access Protocol), un modo per creare oggetti distribuiti e, non ultimo, un metodo per rilevare modifiche dei dati.

Ovviamente, affinché queste features possano essere realmente utili è necessario che sia mantenuta una certa indipendenza dall'architettura hardware di una specifica macchina e dal sistema operativo che gira su di essa. Ad esempio, affinché un'applicazione software sia facilmente distribuibile su macchine eterogenee è necessario che i dati serializzati possano essere ricostruiti efficacemente indipendentemente dall'architettura di memoria (16, 32 o 64 bit) e dal modo in cui il sistema operativo la gestisce. La possibilità di serializzare i dati in un formato indipendente dall'architettura porta con se diversi vantaggi come l'assenza di problemi legati ad ordinamenti specifici dei bytes di dati, l'assenza di problemi legati alla configurazione della memoria e, cosa non meno importante, l'assenza di impedimenti derivanti da differenti modi di rappresentare le strutture dati in differenti linguaggi di programmazione. Java realizza gran parte di queste features mediante l'adozione di una Java Virtual Machine specifica per ciascuna macchina. In questo senso la JVM costituisce un layer che si interpone fra i dati ed il sistema operativo.

Il linguaggio di programmazione Java mette a disposizione un meccanismo automatico piuttosto semplice per la serializzazione degli oggetti che richiede, esclusivamente, che le classi da serializzare implementino l'interfaccia "java.io.Serializable".

Pertanto, una classe che implementa la suddetta interfaccia viene "marcata" come serializzabile in modo che java possa gestirne la serializzazione internamente.

Java fornisce agli sviluppatori anche uno strumento per effettuare l'override del processo di serializzazione standard al fine di personalizzarlo. Questo strumento è rappresentato dall'interfaccia "java.io.Externalizable".

Nel corso di questo articolo, tuttavia, vedremo l'uso del meccanismo standard di serializzazione e salveremo su file un "ArrayList" di oggetti "relativamente complessi". La classe che costituirà ciascun elemento dell'ArrayList implementerà, pertanto, l'interfaccia "java.io.Serializable".

Vedremo, in tal senso, due realizzazioni della serializzazione. Nella prima applicheremo la serializzazione in modo semplice, limitandoci a scrivere su file i byte che compongono l'oggetto serializzato. Nella seconda, invece, effettueremo una serializzazione su file codificando il contenuto secondo un algoritmo di codifica a 64 bit.

Per la codifica utilizzeremo il metodo "encode" della classe "sun.misc.BASE64Encoder". L'uso di questa classe è solo esemplificativo. Sun ne sconsiglia l'utilizzo, in quanto non garantisce il supporto nelle future versioni di Java e nelle distribuzioni non ufficiali.

Per chi in ambiente Linux utilizzasse openjdk (ma anche altre distribuzioni Java di terze parti) è possibile sostituire questa classe con la classe "org.apache.commons.codec.binary.Base64" appartenente alla lib "commons-codec-1.4.jar".

Ovviamente, per entrambe le realizzazioni vedremo anche i processi inversi di lettura dei files, deserializzazione e ricostruzione delle informazioni.

Ma passiamo ad esaminare il codice della prima implementazione. Innanzitutto la prima operazione che eseguiamo è la creazione delle classi (bean) che vogliamo serializzare. Ne utilizzeremo due: la classe Persona e la classe Luogo.

La classe Persona dovrebbe contenere i dati anagrafici di ipotetici individui. Tra questi dati saranno presenti anche il luogo di nascita ed il luogo di residenza che sono istanze della classe Luogo.

Useremo gli stessi bean in entrambe le realizzazioni; tuttavia, tengo a precisare che queste classi hanno il solo scopo di mostrare il processo di serializzazione e, quindi, sono ingegnerizzate e definite quel tanto che basta a conseguire il nostro obiettivo.

{codecitation class="brush: java; gutter: false;" width="700px"} /* Inizio classe Persona.java */ package it.battuello.esercizi.serializzazione.standard.bean; import java.io.Serializable; import java.util.Date; /** * @author Enrico Battuello * */ public class Persona implements Serializable { /** * */ private static final long serialVersionUID = -3736556547234648729L; private String nome; private String cognome; private Integer eta; private String sesso; private Date dataNascita; private Luogo luogoNascita; private Luogo luogoResidenza; /** * */ public Persona() { super(); } /** * @param nome * @param cognome * @param eta * @param sesso * @param dataNascita * @param luogoNascita * @param luogoResidenza */ public Persona(String nome, String cognome, Integer eta, String sesso, Date dataNascita, Luogo luogoNascita, Luogo luogoResidenza) { super(); this.nome = nome; this.cognome = cognome; this.eta = eta; this.sesso = sesso; this.dataNascita = dataNascita; this.luogoNascita = luogoNascita; this.luogoResidenza = luogoResidenza; } /** * @param personaCorrente */ public Persona (Persona personaCorrente) { this(personaCorrente.getNome(), personaCorrente.getCognome(), personaCorrente.getEta(), personaCorrente.getSesso(), personaCorrente.getDataNascita(), personaCorrente.getLuogoNascita(), personaCorrente.getLuogoResidenza()); } /** * @return the nome */ public String getNome() { return nome; } /** * @param nome the nome to set */ public void setNome(String nome) { this.nome = nome; } /** * @return the cognome */ public String getCognome() { return cognome; } /** * @param cognome the cognome to set */ public void setCognome(String cognome) { this.cognome = cognome; } /** * @return the eta */ public Integer getEta() { return eta; } /** * @param eta the eta to set */ public void setEta(Integer eta) { this.eta = eta; } /** * @return the sesso */ public String getSesso() { return sesso; } /** * @param sesso the sesso to set */ public void setSesso(String sesso) { this.sesso = sesso; } /** * @return the dataNascita */ public Date getDataNascita() { return dataNascita; } /** * @param dataNascita the dataNascita to set */ public void setDataNascita(Date dataNascita) { this.dataNascita = dataNascita; } /** * @return the luogoNascita */ public Luogo getLuogoNascita() { return luogoNascita; } /** * @param luogoNascita the luogoNascita to set */ public void setLuogoNascita(Luogo luogoNascita) { this.luogoNascita = luogoNascita; } /** * @return the luogoResidenza */ public Luogo getLuogoResidenza() { return luogoResidenza; } /** * @param luogoResidenzA the luogoResidenza to set */

public void setLuogoResidenza(Luogo luogoResidenza) { this.luogoResidenza = luogoResidenza; } } /* Fine classe Persona.java */ {/codecitation}

{codecitation class="brush: java; gutter: false;" width="700px"} /* Inizio classe Luogo.java */ package it.battuello.esercizi.serializzazione.standard.bean; import java.io.Serializable; /** * @author Enrico Battuello * */ public class Luogo implements Serializable { /** * */ private static final long serialVersionUID = -7176050266479335730L; private String tipoIndirizzo; private String indirizzo; private String numeroCivico; private String cap; private String nomeComune; private String provincia; /** * */ public Luogo() { super(); } /** * @param tipoIndirizzo * @param indirizzo * @param numeroCivico * @param cap * @param nomeComune * @param provincia */ public Luogo(String tipoIndirizzo, String indirizzo, String numeroCivico, String cap, String nomeComune, String provincia) { super(); this.tipoIndirizzo = tipoIndirizzo; this.indirizzo = indirizzo; this.numeroCivico = numeroCivico; this.cap = cap; this.nomeComune = nomeComune; this.provincia = provincia; } /** * @param luogoCorrente */ public Luogo (Luogo luogoCorrente) { this(luogoCorrente.getTipoIndirizzo(), luogoCorrente.getIndirizzo(), luogoCorrente.getNumeroCivico(), luogoCorrente.getCap(), luogoCorrente.getNomeComune(), luogoCorrente.getProvincia()); } /** * @return the tipoIndirizzo */ public String getTipoIndirizzo() { return tipoIndirizzo; } /** * @param tipoIndirizzo the tipoIndirizzo to set */ public void setTipoIndirizzo(String tipoIndirizzo) { this.tipoIndirizzo = tipoIndirizzo; } /** * @return the indirizzo */ public String getIndirizzo() { return indirizzo; } /** * @param indirizzo the indirizzo to set */ public void setIndirizzo(String indirizzo) { this.indirizzo = indirizzo; } /** * @return the numeroCivico */ public String getNumeroCivico() { return numeroCivico; } /** * @param numeroCivico the numeroCivico to set */ public void setNumeroCivico(String numeroCivico) { this.numeroCivico = numeroCivico; } /** * @return the cap */ public String getCap() { return cap; } /** * @param cap the cap to set */ public void setCap(String cap) { this.cap = cap; } /** * @return the nomeComune */ public String getNomeComune() { return nomeComune; } /** * @param nomeComune the nomeComune to set */ public void setNomeComune(String nomeComune) { this.nomeComune = nomeComune; } /** * @return the provincia */ public String getProvincia() { return provincia; } /** * @param provincia the provincia to set */ public void setProvincia(String provincia) { this.provincia = provincia; } } /* Fine classe Luogo.java */ {/codecitation}

Come possiamo vedere dal codice Java, entrambi i bean implementano l'interfaccia “java.io.Serializable”. Inoltre, in ciascuno di essi è presente una costante “serialVersionUID” inizializzata con uno specifico valore long. Questa costante serve nella fase di deserializzazione per verificare che il sender (il mittente) e il receiver (il destinatario) dell'oggetto serializzato possiedono la medesima versione della classe. Se il “serialVersionUID” del destinatario è diverso da quello del mittente, il tentativo di deserializzazione si conclude con un'eccezione di

tipo “InvalidClassException”.

Tipicamente, quando si apportano modifiche consistenti ad una classe serializzabile (ad esempio, si cambia il tipo di uno o più attributi) il valore di questa costante deve essere ricalcolato/cambiato, ad indicare che la classe è diversa da quella originaria. Dopo la modifica, il tentativo di leggere dati serializzati con la “vecchia versione” della classe genera eccezioni di tipo “InvalidClassException”.

In assenza di un valore esplicito per la costante “serialVersionUID”, la JVM utilizza un algoritmo interno per determinarne uno di default. La computazione di questo valore di default è altamente sensibile ai dettagli implementativi della classe e può variare in funzione dell'implementazione della JVM. Questo significa che a parità di classe, differenti JVM possono computare un valore diverso per la costante “serialVersionUID”. Un caso tipico può verificarsi, ad esempio, in un ambiente client/server in cui il client usa la JVM di Sun su un sistema operativo Windows mentre il server adotta la JVM di JRockit su un sistema operativo Linux. E' per questi motivi che è fortemente consigliato assegnare un valore esplicito alla suddetta costante. Questo valore può essere assegnato a mano oppure può essere fatto computare alla JVM durante la scrittura del codice sorgente. L'importante è che sia integrato in quest'ultimo.

Torniamo, quindi, all'analisi del codice.

Come potete osservare i due bean sono estremamente semplici. Essi sono dotati di costruttori e di metodi get e set per leggere e scrivere i valori degli attributi privati.

A conti fatti su di essi non c'è molto altro da dire.

Pertanto passiamo ad esaminare il codice della classe “SerializzaSuFile” che istanzia alcuni oggetti di tipo Persona e di tipo Luogo e li fa persistere su file sfruttando il meccanismo della serializzazione.

{codecitation class="brush: java; gutter: false;" width="700px"} /* Inizio classe SerializzaSuFile.java */ package it.battuello.esercizi.serializzazione.standard; import

java.io.FileOutputStream; import java.io.IOException; import java.io.ObjectOutputStream; import java.util.ArrayList; import java.util.Calendar; import java.util.GregorianCalendar; import java.util.List; import it.battuello.esercizi.serializzazione.standard.bean.Luogo; import it.battuello.esercizi.serializzazione.standard.bean.Persona; /** * @author Enrico Battuello * */ public class SerializzaSuFile { public static void main(String[] args) { String fileName = "varie/elenco.txt"; Calendar dataNascita = GregorianCalendar.getInstance(); dataNascita.set(Calendar.YEAR, 1971); dataNascita.set(Calendar.MONTH, 2); dataNascita.set(Calendar.DAY_OF_MONTH, 24); Luogo luogoNascita = new Luogo ("Via", "Raffaele Baldi", "22", "84013", "Cava de' Tirreni", "SA"); Luogo luogoResidenza = new Luogo ("Via", "Carso", "6", "13048", "Santhia'", "VC"); Persona person1 = new Persona("Enrico", "Battuello", 39, "M", new java.util.Date (dataNascita.getTimeInMillis()), luogoNascita, luogoResidenza); dataNascita.set(Calendar.YEAR, 1998); dataNascita.set(Calendar.MONTH, 5); dataNascita.set(Calendar.DAY_OF_MONTH, 2); luogoNascita = new Luogo ("Via", "Primo Carnera", "5", "13124", "Cossato", "BI"); luogoResidenza = new Luogo ("Via", "Carso", "10", "13033", "Cigliano", "VC"); Persona person2 = new Persona("Antonio", "Gigliotti", 12, "M", new java.util.Date(dataNascita.getTimeInMillis()), luogoNascita, luogoResidenza); List list = new ArrayList(); list.add(person1); list.add(person2); FileOutputStream fileOutputStream = null; ObjectOutputStream objectOutputStream = null; try { fileOutputStream = new FileOutputStream(fileName); objectOutputStream = new ObjectOutputStream(fileOutputStream); objectOutputStream.writeObject(list); objectOutputStream.close(); fileOutputStream.close(); System.out.println("Oggetto correttamente salvato su file."); } catch (IOException ex) { ex.printStackTrace(); } } } /* Fine classe SerializzaSuFile.java */ {/codecitation}

Come possiamo vedere dal codice anche questa classe è molto semplice. Essa possiede un unico metodo main in cui vengono eseguite sia le operazioni che istanziano gli oggetti sia le operazioni che fanno persistere questi oggetti su file. Indubbiamente, da un punto di vista tecnico, questo non è molto bello ma per gli scopi di questo articolo la cosa non ha alcuna importanza. Inoltre, proprio perché dotata del metodo “main”, questa classe è eseguibile localmente come un'applicazione java.

Pertanto analizziamo più in dettaglio cosa accade in questo metodo.

Dopo aver istanziato due oggetti di tipo Persona:

Persona person1 = new Persona("Enrico", "Battuello", 39, "M", new java.util.Date(dataNascita.getTimeInMillis()), luogoNascita, luogoResidenza);

Persona person2 = new Persona("Antonio", "Gigliotti", 12, "M", new java.util.Date(dataNascita.getTimeInMillis()), luogoNascita, luogoResidenza);

viene creato un ArrayList di oggetti di tipo Persona in cui vengono inserite le due istanze appena create:

```
List list = new ArrayList();
list.add(person1);
list.add(person2);
```

Quindi, viene istanziato un oggetto di tipo “FileOutputStream” per poter scrivere uno stream di output su file. Il nome ed il percorso del file vengono passati al costruttore della classe “FileOutputStream” mediante la variabile “fileName”:

```
String fileName = "varie/elenco.txt";
FileOutputStream fileOutputStream = null;
fileOutputStream = new FileOutputStream(fileName);
```

In questo caso, poiché l'esempio è tratto da un mini-progetto eclipse, la cartella “varie” è nella root del progetto “serializzazione”.

Successivamente viene definito un oggetto di tipo “ObjectOutputStream” che costituirà lo stream vero e proprio che sarà scritto sul “FileOutputStream” precedentemente definito:

```
ObjectOutputStream objectOutputStream = null;
objectOutputStream = new ObjectOutputStream(fileOutputStream);
```

Quindi viene chiamato il metodo writeObject dell'ObjectOutputStream appena istanziato e

mediante esso viene eseguita la scrittura su file dell'ArrayList di oggetti Persona precedentemente istanziato:

```
objectOutputStream.writeObject(list);
```

Infine vengono chiusi lo stream di output ed il suo handle al file su disco per effettuare il “flush” del buffer di scrittura e liberare le risorse di sistema:

```
objectOutputStream.close();
fileOutputStream.close();
```

Se l'esecuzione di questa piccola applicazione java si conclude correttamente, nella cartella “varie” del nostro progetto eclipse troveremo il file “elenco.txt” contenente l'ArrayList di oggetti Persona, precedentemente definito, serializzato in stream di byte. Si osservi che “elenco.txt”, nonostante l'estensione, non è un file di testo bensì uno stream di byte.

Vediamo, ora, un altro piccolo programma java che effettua l'operazione inversa ovvero legge il file “elenco.txt” e ricostruisce l'ArrayList con gli oggetti contenuti. Il programma si chiama “DeserializzaDaFile”. Il suo codice è molto semplice e, in sostanza, ripete all'inverso i passi fatti dal programma di serializzazione. L'unica differenza è che, mentre prima, per la scrittura, venivano utilizzati “FileOutputStream” e “ObjectOutputStream” ora, invece, per la lettura, vengono impiegati “FileInputStream” ed “ObjectInputStream”. Per questo motivo eviteremo di commentare il codice.

```
{codecitation class="brush: java; gutter: false;" width="700px"}  /* Inizio classe DeserializzaDaFile.java */    package it.battuello.esercizi.serializzazione.standard;    import it.battuello.esercizi.serializzazione.standard.bean.Persona;    import java.io.FileInputStream;  import java.io.IOException;  import java.io.ObjectInputStream;  import java.util.ArrayList;  import java.util.Iterator;  import java.util.List;    /**   * @author Enrico Battuello   *   */  public class DeserializzaDaFile {       public static void main(String[] args) {        String fileName = "varie/elenco.txt";        List elencoPersone = null;        FileInputStream fileInputStream = null;        Iterator iteraPersone = null;        ObjectInputStream objectInputStream = null;        Persona personaCorrente = null;        try {            fileInputStream = new FileInputStream(fileName);  objectInputStream = new ObjectInputStream(fileInputStream);            elencoPersone =
```

(ArrayList) objectInputStream.readObject(); objectInputStream.close(); fileInputStream.close(); } catch (IOException ex) { ex.printStackTrace(); } catch (ClassNotFoundException ex) { ex.printStackTrace(); } System.out.println("Dimensioni elenco: " + elencoPersone.size()); System.out.println(); iteraPersone = elencoPersone.iterator(); while (iteraPersone.hasNext()) { personaCorrente = (Persona)iteraPersone.next(); System.out.println("Nome: " + personaCorrente.getNome()); System.out.println("Cognome: " + personaCorrente.getCognome()); System.out.println("Sesso: " + personaCorrente.getSesso()); System.out.println("Eta': " + personaCorrente.getEta()); System.out.println("Nato a: " + personaCorrente.getLuogoNascita().getNomeComune()); System.out.println("In via: " + personaCorrente.getLuogoNascita().getIndirizzo() + " " + personaCorrente.getLuogoNascita().getNumeroCivico()); System.out.println("Residente a: " + personaCorrente.getLuogoResidenza().getNomeComune()); System.out.println("In via: " + personaCorrente.getLuogoResidenza().getIndirizzo() + " " + personaCorrente.getLuogoResidenza().getNumeroCivico()); System.out.println(); } } } /* Fine classe DeserializzaDaFile.java */ {/codecitation}

L'esecuzione di questa piccola applicazione java mostra a video sullo standard output (che in eclipse corrisponde alla finestra della console) l'elenco delle persone che abbiamo salvato nel file “elenco.txt”.

Ciò che abbiamo visto fino ad ora corrisponde al processo di serializzazione standard. Il file generato può essere decodificato nelle informazioni di partenza in modo “abbastanza semplice” da chiunque. Vediamo, pertanto, un'altra implementazione di questa procedura di serializzazione su file che codifica i dati adoperando un algoritmo di codifica a 64bit. Ciò renderà i nostri dati molto meno leggibili almeno per coloro che ignorano il metodo e la classe che ci accingiamo ad utilizzare.

Ovviamente questa “soluzione” ha il solo scopo di ispirare l'ingegno del lettore circa le possibilità che possono aprirsi. Iniziamo, quindi, con il vedere il codice della classe “SerializzaSuFileEncoded64” che effettua la serializzazione e la codifica su file.

{codecitation class="brush: java; gutter: false;" width="700px"} /* Inizio classe SerializzaSuFileEncoded64.java */ package it.battuello.esercizi.serializzazione.standard; import it.battuello.esercizi.serializzazione.standard.bean.Luogo; import it.battuello.esercizi.serializzazione.standard.bean.Persona; import java.io.ByteArrayOutputStream; import java.io.FileOutputStream; import java.io.IOException; import java.io.ObjectOutputStream; import java.util.ArrayList; import java.util.Calendar; import

java.util.GregorianCalendar; import java.util.List; /** * @author Enrico Battuello * */ public class SerializzaSuFileEncoded64 { public static void main(String[] args) { String filename = "varie/elenco64.txt"; Calendar dataNascita = GregorianCalendar.getInstance(); dataNascita.set(Calendar.YEAR, 1971); dataNascita.set(Calendar.MONTH, 2); dataNascita.set(Calendar.DAY_OF_MONTH, 24); Luogo luogoNascita = new Luogo ("Via", "Raffaele Baldi", "22", "84013", "Cava de' Tirreni", "SA"); Luogo luogoResidenza = new Luogo ("Via", "Carso", "6", "13048", "Santhia'", "VC"); Persona person1 = new Persona("Enrico", "Battuello", 39, "M", new java.util.Date(dataNascita.getTimeInMillis()), luogoNascita, luogoResidenza); dataNascita.set(Calendar.YEAR, 1998); dataNascita.set(Calendar.MONTH, 5); dataNascita.set(Calendar.DAY_OF_MONTH, 2); luogoNascita = new Luogo ("Via", "Primo Carnera", "5", "13124", "Cossato", "BI"); luogoResidenza = new Luogo ("Via", "Carso", "10", "13033", "Cigliano", "VC"); Persona person2 = new Persona("Antonio", "Gigliotti", 12, "M", new java.util.Date(dataNascita.getTimeInMillis()), luogoNascita, luogoResidenza); List list = new ArrayList(); list.add(person1); list.add(person2); FileOutputStream fileOutputStream = null; ObjectOutputStream objectOutputStreamEncoded = null; ByteArrayOutputStream byreArrayOutputStream = null; ObjectOutputStream objectOutputStreamToEncode = null; try { fileOutputStream = new FileOutputStream(filename); objectOutputStreamEncoded = new ObjectOutputStream(fileOutputStream); byreArrayOutputStream = new ByteArrayOutputStream(); objectOutputStreamToEncode = new ObjectOutputStream(byreArrayOutputStream); objectOutputStreamToEncode.writeObject(list); objectOutputStreamToEncode.close(); byte[] buf = byteArrayOutputStream.toByteArray(); // String encodedString = new sun.misc.BASE64Encoder().encode(buf); String encodedString = new org.apache.commons.codec.binary.Base64().encodeToString(buf); objectOutputStreamEncoded.writeObject(encodedString); objectOutputStreamEncoded.close(); System.out.println("Oggetto correttamente salvato su file."); } catch (IOException ex) { ex.printStackTrace(); } } } /* Fine classe SerializzaSuFileEncoded64.java */ {/codecitation}

Come si può osservare dal codice questa classe è molto simile alla classe “SerializzaSuFile” vista precedentemente. Le uniche differenze presenti nel codice sono l'introduzione di un ulteriore oggetto di tipo “ObjectOutputStream”, l'introduzione di un oggetto istanza della classe “ByteArrayOutputStream” ed ovviamente l'uso del metodo static “encodeToString” che codifica lo stream serializzato.
Ma vediamo in maggior dettaglio cosa accade. Il primo oggetto di tipo “ObjectOutputStream” non scrive più la serializzazione dell'ArrayList di oggetti “Persona” direttamente su file bensì salva temporaneamente quest'ultima sull'oggetto di tipo “ByteArrayOutputStream”:

ObjectOutputStream objectOutputStreamToEncode = null;
ByteArrayOutputStream byteArrayOutputStream = null;

```
byteArrayOutputStream = new ByteArrayOutputStream();
objectOutputStreamEncoded = new ObjectOutputStream(byteArrayOutputStream);
objectOutputStreamEncoded.writeObject(list);
objectOutputStreamEncoded.close();
```

Quindi lo stream di tipo “ByteArrayOutputStream” ottenuto viene convertito in un array di byte che viene passato al metodo di codifica che lo restituisce codificato in un oggetto stringa che, a sua volta, viene salvato sul file “elenco64.txt” mediante il secondo oggetto “ObjectOutputStream”.

```
String filename = "varie/elenco64.txt";
FileOutputStream fileOutputStream = null;
ObjectOutputStream objectOutputStreamEncoded = null;
fileOutputStream = new FileOutputStream(filename);
objectOutputStreamEncoded = new ObjectOutputStream(fileOutputStream);
byte[] buf = byteArrayOutputStream.toByteArray();
// String encodedString = new sun.misc.BASE64Encoder().encode(buf);
String encodedString = new org.apache.commons.codec.binary.Base64().encodeToString(buf);
objectOutputStreamEncoded.writeObject(encodedString);
objectOutputStreamEncoded.close();
```

Si osservi che anche il file “elenco64.txt”, nonostante l'estensione, è uno stream di raw bytes (un binario).

Per concludere aggiungiamo il codice di una piccola applicazione java che effettua l'operazione inversa di decodifica e deserializzazione del contenuto del file. Anche in questo caso vengono ripetuti, in ordine inverso, i passi fatti dall'algoritmo precedente di serializzazione e codifica.

{codecitation class="brush: java; gutter: false;" width="700px"} /* Inizio classe DeserializzaDaFileEncoded64.java */ package it.battuello.esercizi.serializzazione.standard; import it.battuello.esercizi.serializzazione.standard.bean.Persona; import java.io.ByteArrayInputStream; import java.io.FileInputStream; import java.io.IOException; import java.io.ObjectInputStream; import java.util.ArrayList; import java.util.Iterator; import java.util.List; /** * @author Enrico Battuello * */ public class DeserializzaDaFileEncoded64 { public static void main(String[] args) { String filename = "varie/elenco64.txt"; String encodedString = null; List elencoPersone = null;

```
FileInputStream fileInputStream = null;  Iterator iteraPersone = null;  ObjectInputStream objectInputStream = null;  Persona personaCorrente = null;  try {  elencoPersone = new ArrayList();  fileInputStream = new FileInputStream(filename);  objectInputStream = new ObjectInputStream(fileInputStream);  encodedString = (String)objectInputStream.readObject();  objectInputStream.close();  // byte[] bufferDecoded = new sun.misc.BASE64Decoder().decodeBuffer(encodedString);  byte[] bufferDecoded = new org.apache.commons.codec.binary.Base64().decode(encodedString);  if (bufferDecoded != null) {  ObjectInputStream objectIn = new ObjectInputStream(  new ByteArrayInputStream(bufferDecoded));  elencoPersone = (ArrayList) objectIn.readObject();  objectIn.close();  }  } catch (IOException ex) { ex.printStackTrace();  } catch (ClassNotFoundException ex) {  ex.printStackTrace();  }  System.out.println("Dimensioni elenco: " + elencoPersone.size()); System.out.println();  iteraPersone = elencoPersone.iterator();  while (iteraPersone.hasNext()) {  personaCorrente = (Persona)iteraPersone.next(); System.out.println("Nome: " + personaCorrente.getNome()); System.out.println("Cognome: " + personaCorrente.getCognome()); System.out.println("Sesso: " + personaCorrente.getSesso());  System.out.println("Eta': " + personaCorrente.getEta());  System.out.println("Nato a: " + personaCorrente.getLuogoNascita().getNomeComune());  System.out.println("In via: " + personaCorrente.getLuogoNascita().getIndirizzo() +  " " + personaCorrente.getLuogoNascita().getNumeroCivico());  System.out.println("Residente a: " + personaCorrente.getLuogoResidenza().getNomeComune());  System.out.println("In via: " + personaCorrente.getLuogoResidenza().getIndirizzo() +  " " + personaCorrente.getLuogoResidenza().getNumeroCivico());  System.out.println();  } } }  /* Fine classe DeserializzaDaFileEncoded64.java */  {/codecitation}
```

Il codice è molto semplice e non necessita di ulteriori commenti.

Tuttavia, prima di concludere questo articolo, desidero fare alcune osservazioni ulteriori circa la serializzazione in java. Innanzitutto qualcuno potrebbe chiedere il motivo per cui una classe java non è serializzabile di defalut. In realtà ci sono almeno tre ragioni che spiegano questa scelta progettuale:

1. Alcuni oggetti possono contenere informazioni semanticamente inutili in uno stato serializzato. Per esempio un oggetto "Thread" è legato allo stato della JVM corrente. Non esiste un contesto in cui un oggetto "Thread" deserializzato manterrebbe un'informazione semanticamente utile.
2. Lo stato serializzato di un oggetto costituisce una parte del contratto di compatibilità della sua classe. Mantenere la compatibilità tra diverse versioni di classi serializzabili richiede sforzi e considerazioni ulteriori, per cui la scelta di rendere una classe serializzabile deve essere una decisione esplicita di design e non una condizione di default.

3. La serializzazione consente l'accesso ai campi privati non transienti di una classe che normalmente non sarebbero accessibili. Le classi che contengono informazioni sensibili (ad esempio una password) non dovrebbero essere Serializable o Externalizable.

Inoltre, correlato a quanto detto nel punto 3 precedente ma in tono più generale, faccio osservare che la serializzazione rompe l'opacità di un abstract data type, nel senso che potenzialmente può esporre dettagli implementativi privati. Proprio per questo motivo molti produttori di software proprietario mantengono i dettagli implementativi dei formati di serializzazione dei loro programmi come un segreto commerciale. Alcuni arrivano finanche ad offuscare e/o crittografare deliberatamente i dati serializzati.

Infine, come uiltima nota, tengo ad evidenziare il fatto che i componenti dell'interfaccia grafica Swing, seppure serializzabili, non sono portabili tra differenti versioni della Java Virtual Machine. Questo significa che un componente Swing o qualsiasi componente che lo eredita può essere serializzato in un array di bytes ma non è garantito che questa sequenza di bytes venga correttamente interpretata su un'altra macchina

Nei prossimi articoli relativi a questo argomento vedremo come serializzare oggetti complessi in un database oracle/mysql e come creare una serializzazione customizzata.

Viene allegato a questo articolo il progetto eclipse contenente i sorgenti degli esempi presentati (solo per utenti registrati).