

Programmazione

LAB 03

<http://www.iet.unipi.it/m.cimino/prg/>

Mario G.C.A. Cimino

Dipartimento di Ingegneria dell'Informazione

Terza applicazione - serializzazione su file e socket

2 of 11

1. **Specifiche:** consentire la scrittura e lettura della variabile intera, rimuovendo lo stato verde della moneta.
2. La scrittura su file tramite un solo byte è riduttiva. In generale i *flussi file di byte* `FileOutputStream` e `FileInputStream` sono scomodi da gestire da soli, perché a basso livello e richiederebbero dei cicli 'for' per trasformare il dato in un formato seriale adatto per entrare in un flusso (serializzare).
3. Ci sono i *flussi oggetto*, `ObjectInputStream` e `ObjectOutputStream`, che permettono di serializzare automaticamente un oggetto. Il flusso oggetto può poi essere concatenato ad un flusso file per salvare o caricare lo stato di un oggetto da file.
4. Il file binario non è direttamente leggibile da un utente, né esportabile ad altre applicazioni non Java. È possibile a tale scopo adoperare la classe *Files*. Si adoperino dei metodi che permettono di serializzare automaticamente il dato come una sequenza di caratteri, senza implementare dei cicli for.

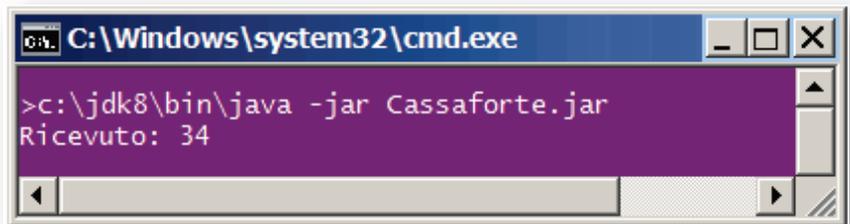
- Implementare l'invio del dato ad un server remoto, creando una applicazione server *Cassaforte* che riceve un valore intero adoperando la classe *ServerSocket*. Per inviare il dato dal lato applicazione, adoperare la classe *Socket*.
- L'applicazione *Cassaforte* andrà avviata prima dell'applicazione *Salvadanaio* in modo che sia pronta a ricevere connessioni remote sulla porta 8080. Per la serializzazione dell'intero, connettere i flussi oggetto alla connessione di rete, e gestire la trasmissione/ricezione in maniera identica a quanto visto precedentemente per i flussi file binari.
- Creare un nuovo progetto di tipo Java Application (senza interfaccia grafica), di nome *Cassaforte*. Prima di cliccare su *Finish* modificare il campo *Create Main Class* a '*Cassaforte*' (quindi senza il prefisso del package).
- Prelevare il file JAR (Java ARchive, un formato zip comodo per distribuire una applicazione composta da molti file class) nella cartella *dist*. Tale cartella viene generata dopo aver eseguito il build.

Esecuzione locale di Server (con console) e Client (console + interfaccia)

The screenshot shows the NetBeans IDE 8.0.1 interface. The main editor displays the source code for `Salvadanaio.java` and `Cassaforte.java`. The `prelevaSalvadanaioTxt` method in `Cassaforte.java` is highlighted, showing a try-catch block that writes to a file and prints an error message. The `Output` window shows the execution of `Cassaforte (run)` and `Salvadanaio (jfxsa-run)`. The `Salvadanaio` application window is visible, featuring a pink piggy bank icon, a yellow coin, and a button labeled "Inserisci" with the value "30".

Esecuzione remota di cassaforte (classe server) con console

1. Scaricare sullo host **server** la Java Runtime Environment (jre) e l'applicazione Cassaforte:
<http://www.iet.unipi.it/m.cimino/prg/res/jre8.zip>
<http://www.iet.unipi.it/m.cimino/prg/res/Cassaforte.jar>
2. Supponiamo di estrarre jre8.zip in C, e Cassaforte.jar sul desktop)
 Da shell posizionarsi sul desktop e digitare
 C:\jre8\bin\java -jar "Cassaforte.jar"
3. Sullo host server
 connettersi all'indirizzo
www.myipaddress.com
 per avere l'indirizzo web.
 Supponiamo che tale
 indirizzo pubblico appartenga direttamente allo host server.
4. Sullo host client: modifichiamo in Salvadanaio la prima riga del metodo *inviaSalvadanaioBin*, sostituendo "localhost" con l'indirizzo IP dello host server. Eseguendo Salvadanaio e poi terminando si potrà vedere sulla console dell'applicazione Cassaforte il valore della variabile monete.



```
C:\prg\myapps\Cassaforte\src\Cassaforte.java
```

```

1 import java.net.*;
2 import java.io.*;
3
4 public class Cassaforte { // (00)
5     public static void main(String[] args) { //01
6         int result = 0;
7         try ( ServerSocket servs = new ServerSocket(8080); // (02)
8             Socket s = servs.accept(); // (03)
9             ObjectInputStream oin = new ObjectInputStream(s.getInputStream());
10        ) { result = (int) oin.readObject();
11        } catch (IOException | ClassNotFoundException e) {e.printStackTrace();}
12        System.out.println("Ricevuto: " + result);
13    }
14 }
15
16 /*
17 (00)
18 L'applicazione Salvadanaio va lanciata dopo aver lanciato l'applicazione
19 Cassaforte. In basso a netbeans si vedono due console distinte.
20 Oppure aprire una finestra shell, collocarsi nella
21 cartella dove si trova Cassaforte.class e digitare:
22 c:\prg\jdk8\bin\java Cassaforte
23 quindi lanciare l'applicazione client.
24 Alla chiusura del client apparira' un messaggio sulla console di Cassaforte
25 (01)
26 Un socket e' un canale di comunicazione bidirezionale tra due programmi che
27 girano su host (computer) connessi in rete. Uno di essi (server) sta in ascolto
28 su una certa porta logica (un identificativo), l'altro (client) si connette
29 su quella porta e si stabilisce cosi il canale di comunicazione, che rilascia
30 dei flussi oggetto nel quale far passare i dati.
```

```

31  https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html
32  https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html
33  https://docs.oracle.com/javase/tutorial/networking/sockets/index.html
34
35 (02)
36  Crea un TCP server socket, in grado di ricevere richieste di connessione e
37  restituire un socket.
38
39 (03)
40  L'applicazione si ferma ed attende richieste di connessione, questo metodo
41  e' quindi bloccante. Se la connessione va a buon fine viene restituito un socket
42  che poi restituisce un ObjectInputStream. E' anche possibile avere un
43  ObjectOutputStream tramite getOutputStream() dato che il canale e' bidirezionale
44 */

```

```
C:\prg\myapps\Salvadanaio\src\Salvadanaio.java
```

```

1  import javafx.application.*;
2  import javafx.stage.*;
3  import javafx.scene.*;
4  import javafx.scene.shape.*;
5  import javafx.scene.image.*;
6  import javafx.scene.control.*;
7  import javafx.scene.paint.*;
8  import javafx.event.*;
9  import java.io.*;
10 import java.nio.file.*;
11 import java.net.*;
12
13 public class Salvadanaio extends Application {
14
15     private int monete = 0;
16
17     private Label la;
18     private Circle ci;
19     private ImageView im;
20
21     private TextField tf;
22     private Button bu;
23
24     public void start(Stage stage) {
25
26         la = new Label("Salvadanaio");
27         ci = new Circle(120, 65, 20, Color.GOLD);
28         im = new ImageView("file:../../myfiles/foto.png");
29         tf = new TextField("0");
30         bu = new Button("Inserisci");
31
32         la.setLayoutX(20);
33         la.setStyle("-fx-font-size: 40px; -fx-text-fill: green; -fx-font-weight: bold;");
34         im.setY(80);
35         tf.setMaxWidth(60);
36         tf.setLayoutX(110); tf.setLayoutY(370);
37         tf.setStyle("-fx-font-size: 20px; -fx-text-fill: blue;");
38         bu.setLayoutX(60); bu.setLayoutY(420);
39         bu.setStyle("-fx-font-size: 30px;");
40
41         bu.setOnAction((ActionEvent ev)->{inserisciMonete();});
42         stage.setOnCloseRequest((WindowEvent we) ->
43 {conservaSalvadanaioBin();conservaSalvadanaioTxt();inviaSalvadanaioBin();});
44
45         Group root = new Group(la, ci, im, tf, bu);
46         Scene scene = new Scene(root, 280, 500);
47         stage.setTitle("Applicazione JavaFX");
48         stage.setScene(scene);
49         stage.show();
50
51         prelevaSalvadanaioTxt();
52         prelevaSalvadanaioBin();
53     }
54     private void inserisciMonete() {

```

```

55     String x = tf.getText();
56     if (x.startsWith("+"))
57         monete = monete + Integer.parseInt(x);
58     else
59         monete++;
60     tf.setText(Integer.toString(monete));
61 }
62
63 private void conservaSalvadanaioBin() {
64     try ( FileOutputStream fout = new FileOutputStream("./myfiles/salvadanaio.bin");
65           ObjectOutputStream oout = new ObjectOutputStream(fout); ) { //01
66         oout.writeObject(monete);
67     } catch (IOException ex) {
68         ci.setFill(Color.RED);
69         System.out.println("errore: impossibile conservare il salvadanaio!");
70     }
71 }
72
73 private void prelevaSalvadanaioBin() {
74     try ( FileInputStream fin = new FileInputStream("./myfiles/salvadanaio.bin");
75           ObjectInputStream oin = new ObjectInputStream(fin); ) { //01
76         monete = (int) oin.readObject();
77     } catch (IOException | ClassNotFoundException ex) {
78         ci.setFill(Color.RED);
79         System.out.println("errore: impossibile prelevare il salvadanaio!");
80     }
81     tf.setText(Integer.toString(monete));
82 }
83
84 private void conservaSalvadanaioTxt() {
85     String x = Integer.toString(monete);
86     try { //02
87         Files.write(Paths.get("./myfiles/salvadanaio.txt"), x.getBytes());
88     } catch (IOException ex) {
89         ci.setFill(Color.RED);
90         System.out.println("errore: impossibile conservare il salvadanaio!");
91     }
92 }
93
94 private void prelevaSalvadanaioTxt() {
95     try { //02
96         String x = new String(Files.readAllBytes(Paths.get("./myfiles/salvadanaio.txt")));
97         monete = Integer.parseInt(x);
98         tf.setText(x);
99     } catch (IOException ex) {
100         ci.setFill(Color.RED);
101         System.out.println("errore: impossibile prelevare il salvadanaio!");
102     }
103 }
104
105 private void inviaSalvadanaioBin() { // (4)
106     try ( Socket s = new Socket("localhost", 8080); // (5)
107           ObjectOutputStream oout = new ObjectOutputStream(s.getOutputStream());
108           ) { oout.writeObject(monete);
109     } catch (IOException e) { e.printStackTrace(); }
110     System.out.println("invia stato");
111 }
112 }
113
114 /*
115 Note:
116 (01)
117 I flussi file di byte sono scomodi da gestire. Ci sono i flussi oggetto che
118 permettono di serializzare nel flusso gli oggetti e i tipi primitivi:
119 ObjectInputStream e ObjectOutputStream. Il flusso oggetto puo' essere agganciato
120 ad un flusso file per salvare o caricare lo stato di un oggetto.
121 https://docs.oracle.com/javase/7/docs/api/java/io/ObjectOutputStream.html
122
123 (02)
124 La classe Files consiste esclusivamente di metodi statici operanti su file,
125 https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html
126

```

127 (04)
128 L'applicazione Salvadanaio va lanciata dopo aver lanciato l'applicazione Cassaforte.
129 Un socket e' un canale di comunicazione bidirezionale tra due programmi che
130 girano su host (computer) connessi in rete. Uno di essi (server) sta in ascolto
131 su una certa porta logica (un identificativo), l'altro (client) si connette
132 su quella porta e si stabilisce cosi' il canale di comunicazione, che rilascia
133 dei flussi oggetto nel quale far passare i dati.
134 <https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html>
135 <https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html>
136
137 <https://docs.oracle.com/javase/tutorial/networking/sockets/index.html>
138
139 (05)
140 Crea un socket e cerca di connetterlo ad una porta di un certo host.
141 Se la connessione va a buon fine viene restituito un ObjectOutputStream.
142 E' anche possibile avere un ObjectInputStream tramite getInputStream()
143 dato che il canale e' bidirezionale.
144
145 Se il server Cassaforte e' avviato su un host diverso, occorre fornire
146 l'indirizzo IP di tale host invece di "localhost".
147 */
148