Universita di Pisa
CdL in Ingegneria Informatica

Programmazione

LAB 04

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

Mario G.C.A. Cimino
Dipartimento di Ingegneria dell'Informazione

Quarta applicazione - server ciclico monoprocesso

1. Modificare il server in modo che sia ciclico, ossia in grado di accettare più richieste in sequenza senza terminare.

2. Modificare il server in modo che compia un'operazione di qualche secondo ogniqualvolta si crea una connessione.

3. Modificare il client in modo che faccia molte richieste al server, ad intervalli di qualche decimo di secondo.

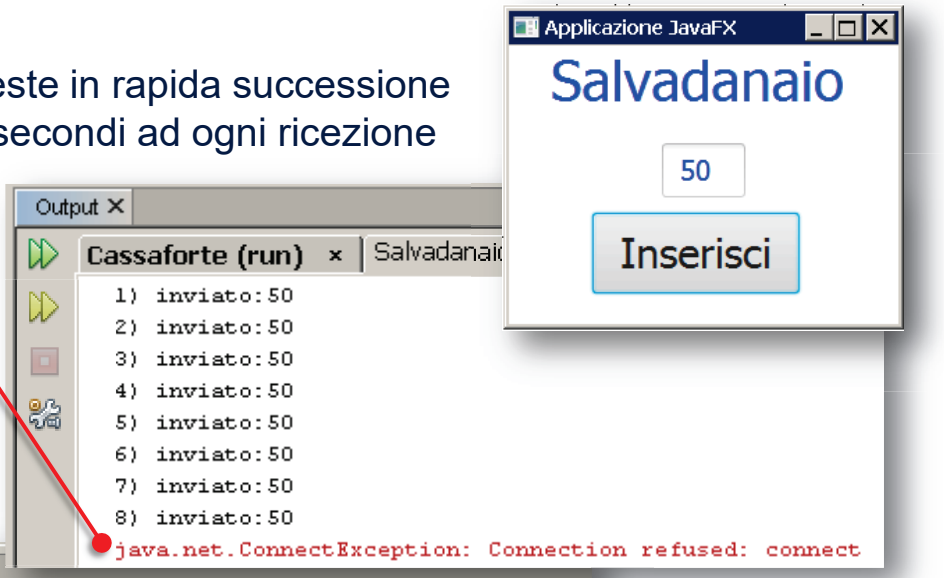
4. Per gestire richieste ravvicinate sulla porta di un server c'è la coda di backlog (arretrato) la cui dimensione indica il numero massimo di richieste TCP che possono essere automaticamente accettate ed accodate senza che la accept() sia ancora stata invocata.

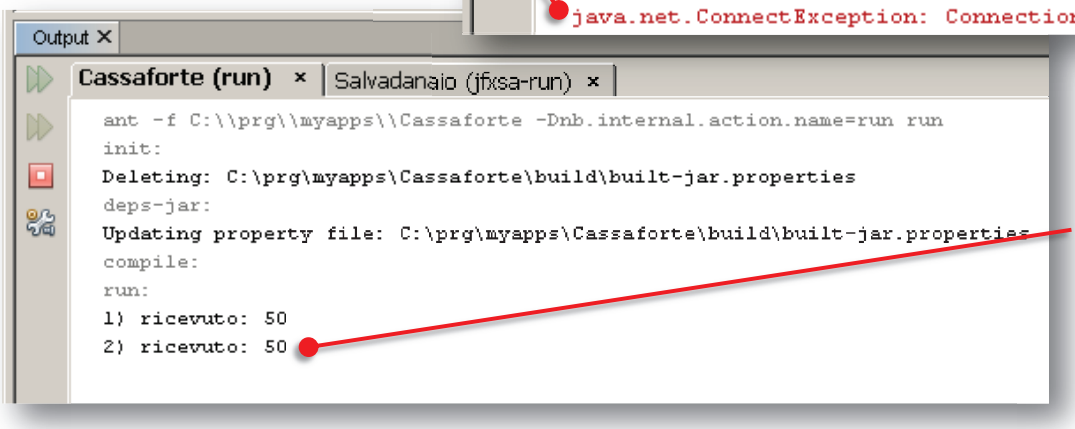
5. In Java esiste il costruttore ServerSocket(int port, **int backlog**) per configurare le dimensioni della coda di backlog. Settare la coda a 7.

6. Catturare l'eccezione generata quando la coda di backlog è piena, e far terminare il client all'occorrenza di tale eccezione.

Idea di massima:
- Il client invia 5000 richieste in rapida successione
- Il server si ferma per 3 secondi ad ogni ricezione

Risultato: dopo 8 invii il client riceve un rifiuto di connessione, in quanto il server ha saturato la coda di backlog.





A quel punto il client termina, ma il server ogni 3 secondi preleva un dato conservato nella coda di backlog.

7. Si riprogetti il server con i **thread** per assegnare, ad ogni richiesta di servizio, un flusso sequenziale di controllo indipendente. Verificare sperimentalmente che in tal caso non si crea backlog.

8. In generale, possono esserci altri problemi di scalabilità sul server: es. dopo molte richieste si arriva al numero massimo di thread istanziabili da un processo, se questi non vengono completati in un tempo opportuno. Riprodurre sperimentalmente questo fenomeno.

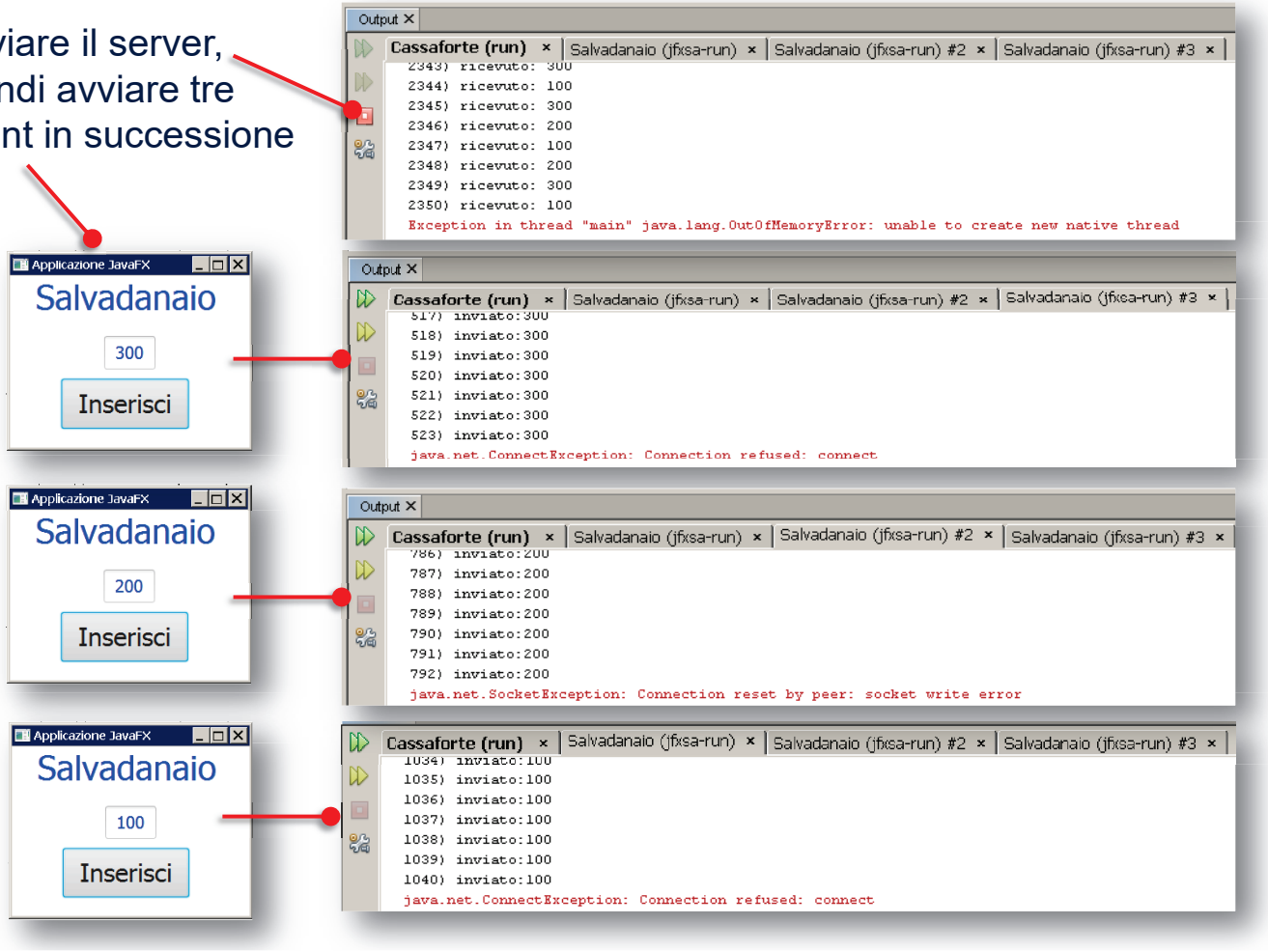
9. Altro problema di scalabilità: se un thread usa una quantità modesta di memoria RAM per un tempo considerevole, un numero elevato di essi può saturare la memoria RAM disponibile per il processo.

Idea di massima:
- Il client invia 5000 richieste in rapida successione (come prima)
- Il server crea un thread che si ferma per 5 minuti ad ogni ricezione

Risultato: la coda di backlog non si saturerà mai, ma dopo aver creato circa 2350 thread (Windows) il server lancerà una eccezione e si fermerà. A quel punto tutti i thread non potranno più inviare dati e si fermeranno.

Quinta applicazione

Avviare il server, quindi avviare tre client in successione



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

```
 1 import javafx.application.*;
 2 import javafx.stage.*;
 3 import javafx.scene.*;
 4 import javafx.scene.control.*;
 5 import javafx.event.*;
 6 import java.io.*;
 7 import java.net.*;
 8
 9 public class Salvadanaio extends Application {
10
11     private int monete = 0;
12
13     private  Label la;
14     private  TextField tf;
15     private  Button bu;
16
17     public void start(Stage stage) {
18
19         la = new Label("Salvadanaio");
20         tf = new TextField("0");
21         bu = new Button("Inserisci");
22
23         la.setLayoutX(30);
24         la.setStyle("-fx-font-size: 40px; -fx-text-fill: blue;");
25         tf.setMaxWidth(60);
26         tf.setLayoutX(110); tf.setLayoutY(70);
27         tf.setStyle("-fx-font-size: 20px; -fx-text-fill: blue;");
28         bu.setLayoutX(60); bu.setLayoutY(120);
29         bu.setStyle("-fx-font-size: 30px;");
30
31         bu.setOnAction((ActionEvent ev)->{
32           inserisciMonete();
33           inviaSalvadanaioBinCiclico();
```

```
34         });
35
36         Group root = new Group(la, tf, bu);
37         Scene scene = new Scene(root, 280, 200);
38         stage.setTitle("Applicazione JavaFX");
39         stage.setScene(scene);
40         stage.show();
41     }
42
43     private void inserisciMonete() {
44       monete = monete + Integer.parseInt(tf.getText());
45       tf.setText(Integer.toString(monete));
46     }
47
48     private void inviaSalvadanaioBinCiclico() {
49       for (int n=1; n<5000; n++) {
50         try ( Socket s = new Socket("localhost", 8080);
51               ObjectOutputStream oout = new ObjectOutputStream (s.getOutputStream());
52         ){ oout.writeObject(monete);
53           System.out.println(n + ") inviato:" + monete);
54         } catch (Exception e) {e.printStackTrace(); System.exit(1);}
55       }
56     }
57 }
58
```

Server mono-processo

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

```
1 import java.net.*;
 2 import java.io.*;
 3
 4 public class Cassaforte {
 5   public static void main(String[] args) {
 6     int n=0;
 7     try ( ServerSocket servs = new ServerSocket(8080, 7) ){ //0
 8       while(true) {
 9         try ( Socket s = servs.accept();
10               ObjectInputStream oin = new ObjectInputStream(s.getInputStream());
11         ){ System.out.println(++n + ") ricevuto: " + (int) oin.readObject()); }
12         Thread.sleep(3000); //1
13       }
14     } catch (Exception e) {e.printStackTrace();}
15   }
16 }
17
18 /*
19 (0) coda di backlog lunga 7 (di default e' circa 50 sui sistemi Windows)
20 (1) simulazione di una elaborazione; il server è in grado di accettare più
21    richieste in sequenza fino alla dimensione della coda di backlog
22 */
23
```

Server multi-thread

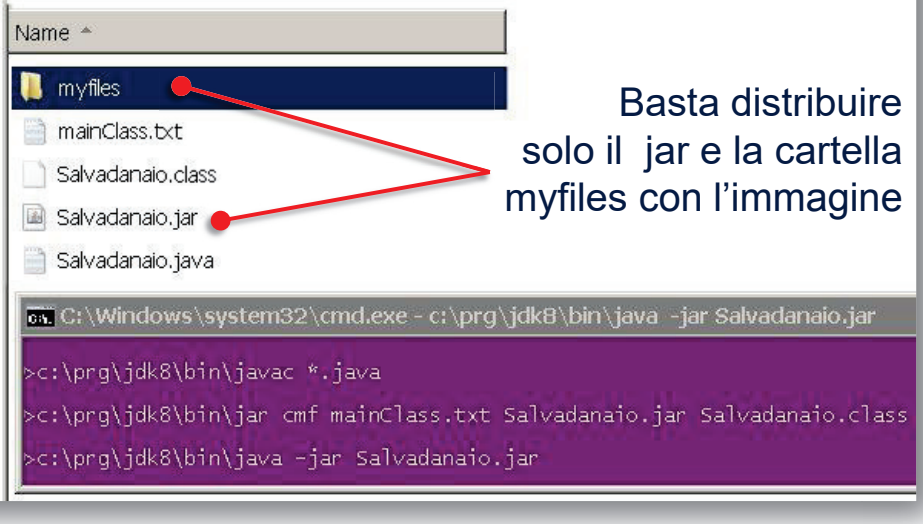
C:\prg\myapps\Cassaforte\src\Cassaforte.java

```
1 import java.net.*;
 2 import java.io.*;
 3
 4 public class Cassaforte {
 5   private static int n = 0;
 6
 7   public static void main(String[] args) {
 8     try ( ServerSocket servs = new ServerSocket(8080, 7) ){ //0
 9       while(true) {
10         Socket s = servs.accept();
11         Thread t = new Thread() { //1
12           public void run() { //2
13             try ( ObjectInputStream oin = new ObjectInputStream(s.getInputStream()) ) {
14                   System.out.println(++n + ") ricevuto: " + (int) oin.readObject());
```

```
15                  Thread.sleep(300000); //3
16              } catch (Exception e) {e.printStackTrace();}
17            }
18          };
19          t.start(); //4
20        }
21      } catch (IOException e) {e.printStackTrace();}
22    }
23 }
24
25 /*
26 Note:
27 (0)  Il parametro 7 del costruttore ServerSocket e' la coda di backlog,
28     il default sotto windows e' 50.
29 (1)  Dopo ogni richiesta viene lanciato un thread per gestirla. Il Thread viene
30     eseguito come classe anonima, in cui si istanzia un costruttore e si
31     ridefinisce il metodo run. Le classi anonime sono comode quando si istanzia
32     un oggetto di classe gia' esistente con modifiche minime. In tal caso non
33     vale la pena di definire nomi di nuove classi, ma basta estendere quella
34     esistente in modo anonimo.
35      https://docs.oracle.com/javase/tutorial/java/javaOO/anonymousclasses.html
36 (2)  Ridefinisco il metodo run() della clase Thread, che contiene il corpo del
37     thread che viene eseguito
38 (3)  Crea una attesa di 5 minuti, sufficiente per creare il massimo numero di
39     thread e far scattare l'eccezione
40 (4)  il metodo start fa partire effettivamente il nuovo thread chiamando run
41
42 Approfondimenti
43 Java Networking
44 https://docs.oracle.com/javase/tutorial/networking/sockets/index.html
45
46 Coda di backlog
47 https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html#ServerSocket-int-int-
48
49 */
```

Creazione di un JAR da riga di comando

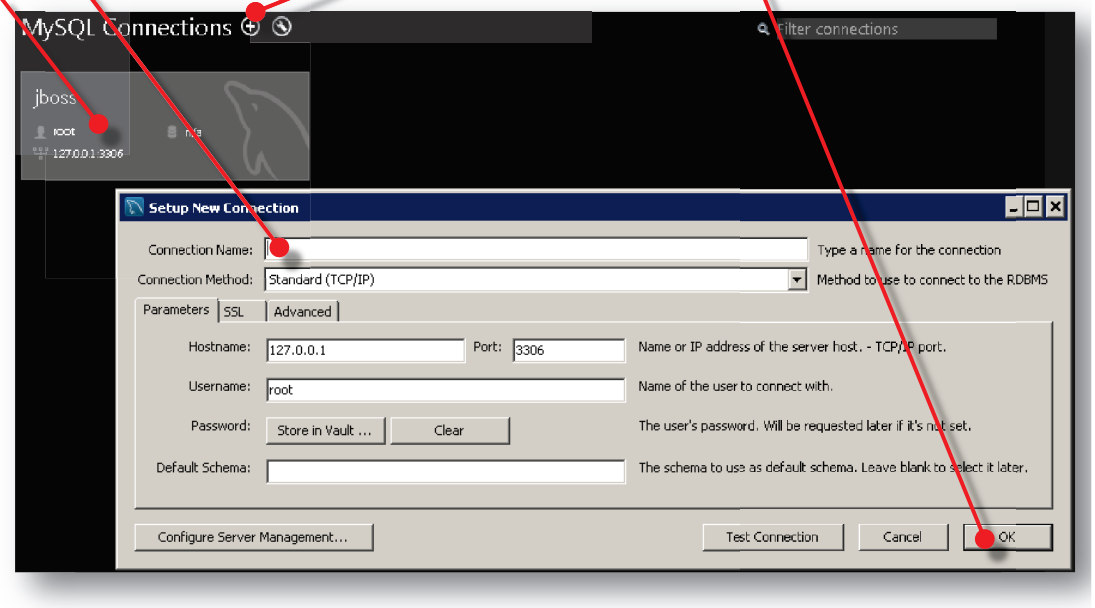
Per creare il jar da riga di comando:
- creare un file di testo mainClass.txt contenente una riga indicante la classe con il metodo main, nel formato: *Main-Class: Salvadanaio* ↵ (accapo)
- eseguire i comandi in figura



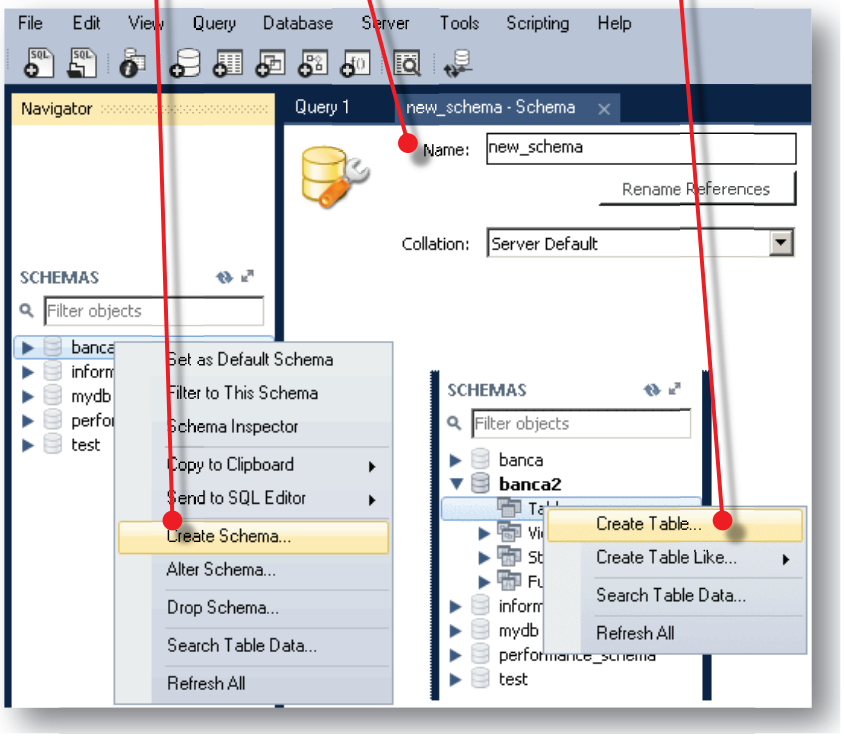
Basta distribuire solo il jar e la cartella myfiles con l'immagine

Tramite i package è possibile organizzare i file class in sottocartelle.
http://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html

1. Archiviare il valore delle monete in un database MySQL.
2. Avvio di MySQL Server: doppio click su *mysqlStart*, dopo qualche secondo apparirà una console con la porta di ascolto (3306).
3. In *MySQL Client 6.1* cliccare sul simbolo '+' di *MySQL Connections,*, digitare un nome della connessione e poi premere OK, quindi doppio click sul rettangolo relativo alla connessione.



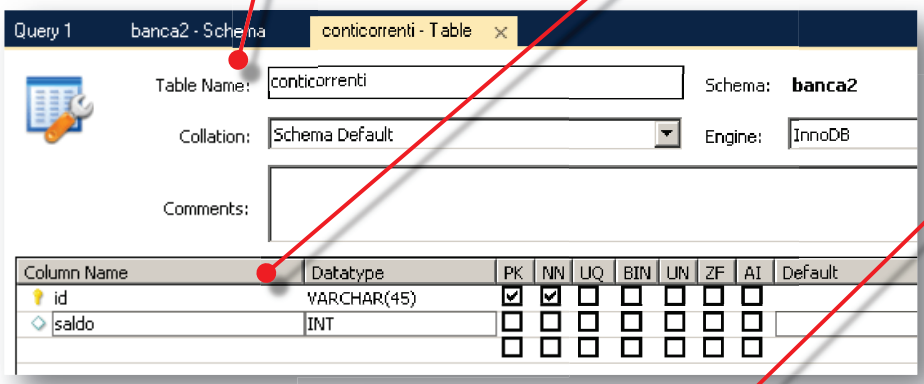
4. Con il tasto destro su uno dei database elencati nella tab *schemas*, selezionare *Create Schema,* digitare il nome *salvadanaio* e premere Apply, Apply, Finish.
5. Selezionare il nuovo db, tasto destro su Tables, selezionare *Create Table*

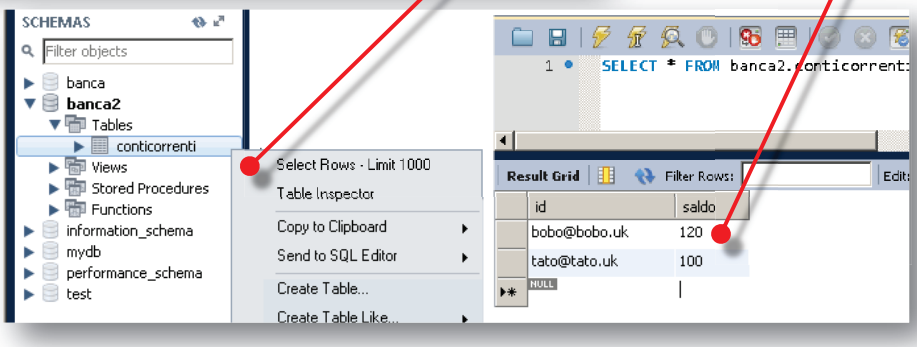


6. Inserire il nome della tabella, e i campi *id* (VARCHAR 45) e *saldo* (INT).

2. Tasto destro sulla Tabella, selezionare *Select Rows…*

3. Editare i valori in tabella

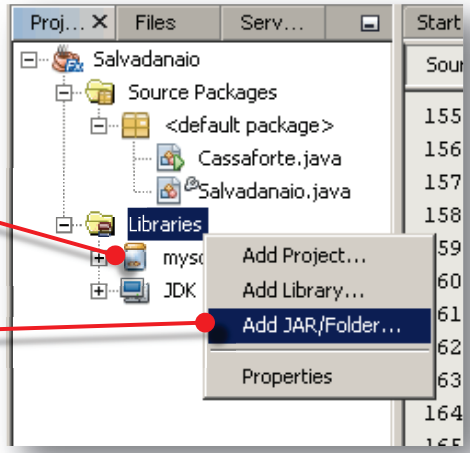




1. Per esportare uno script SQL in grado di ricostruire il db, selezionare in alto a sinistra, nella scheda MANAGEMENT, la voce *Data Export,* quindi selezionare il db da esportare, selezionare *Export to self-contained file*, cliccare su *Start Export.* Viene salvato un file sql.

2. Per ricostruire il database su un altro server, selezionare nella medesima scheda la voce *Data Import/Restore*, quindi *Import from self-contained file,* quindi cliccare su *Start Import.* Quindi cliccare sul tasto di refresh nella scheda SCHEMA.

3. Per poter collegare la libreria mysql-connector-java-5.1.34-bin.jar presente in c:\prg\exe all'interno di NetBeans, tasto destro su Libraries dela scheda Project, selezionare *Add JAR/Folder*



4. In alternativa, da riga di comando digitare:

C:\prg\jdk8\bin\javac -classpath mysql-connector-java-5.1.34-bin.jar *.java
C:\prg\jdk8\bin\java - classpath mysql-connector-java-5.1.34-bin.jar; Test.java