Basi di dati - Laboratorio

Corso di Laurea in Bioinformatica

Docente: Barbara Oliboni

Lezione 7

Contenuto della lezione

- Java DataBase Connectivity (JDBC)
  - Principali classi JDBC
  - Operazioni di base di JDBC
  - Esecuzione di interrogazioni
- Java Data Bean
- Classe DBMS.java

Java DataBase Connectivity

- JDBC consente di interfacciare una base di dati ed un programma java.
- JDBC fornisce un'interfaccia standard per tutte le basi di dati.
- JDBC è un'interfaccia operante a livello delle chiamate: un programma può accedere alle funzioni JDBC utilizzando semplici metodi o chiamate a funzioni.
- La connessione alla base di dati avviene utilizzando un driver JDBC rappresentato da una classe Java.

JDBC: funzionamento - operazioni necessarie

1. Caricamento del driver JDBC per la propria base di dati.
   - Per specificare il nome della classe driver si usa un'istruzione Class.forName(<Nome Driver>)
2. Apertura della connessione con la base di dati.
   - L'operazione viene svolta con una chiamata al metodo statico getConnection(URL) della classe DriverManager. URL indica il tipo di driver e l'origine dei dati da utilizzare.
3. Interazione con la base di dati.
   - Creazione di oggetti Statement tramite i quali possono essere creati i comandi SQL statici.
4. Elaborazione dei risultati.
   - L'interfaccia ResultSet fornisce dei metodi per esaminare le righe restituite ed estrarre i valori da ciascuna colonna.

Principali classi JDBC (1)

- L'interfaccia JDBC è contenuta nei package java.sql e javax.sql.
- Le classi più utilizzate sono:
  - Connection: collegamento attivo con una base di dati, tramite il quale un programma Java può leggere e scrivere i dati.
    - Un oggetto Connection può essere creato tramite una chiamata a DriverManager.getConnection()

Principali classi JDBC (2)

- L'interazione con il DBMS avviene attraverso oggetti generati invocando metodi dell'oggetto Connection, che consentono di inviare delle istruzioni SQL e di ricevere i risultati. Ci sono due classi di oggetti per tale interazione:
  - Statement: utilizzato per eseguire interrogazioni SQL statiche. Un oggetto Statement può essere creato con Connection.createStatement().
  - PreparedStatement: estensione di Statement che consente di precompilare interrogazioni SQL con parametri di input etichettati con il simbolo '?' e attualizzati successivamente con metodi specifici prima dell'esecuzione effettiva. Un oggetto PreparedStatement può essere creato con Connection.prepareStatement(stringaSQL).

Principali classi JDBC (3)

- ResultSet: risultato composto da un insieme ordinato di righe prodotte da un server SQL.
  - Un ResultSet può essere restituito dalla chiamata al metodo executeQuery(stringaSQL).
- SQLException: classe base per eccezioni utilizzata dall'API JDBC.

Operazioni di base di JDBC

1) Caricamento del driver JDBC:

```
Class.forName("nome-driver");
```

2) Apertura della connessione con la base di dati:

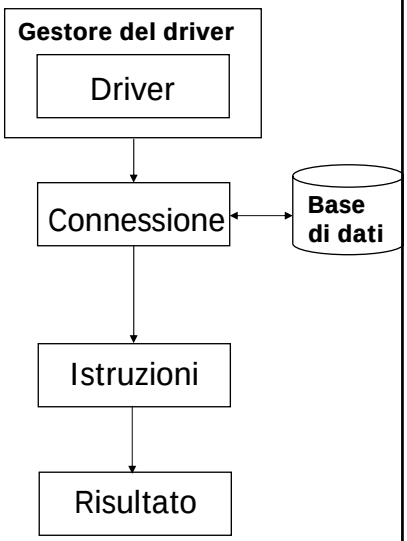
```
DriverManager.getConnection
(jdbc:tipoDBMS://URL/Database)
```

3) Invio delle istruzioni SQL:

```
stmt = con.createStatement();
stmt.executeQuery
("SELECT * FROM Tabella");
```

4) Elaborazione del risultato:

```
while(rs.next()){
   name = rs.getString("name");
   amount = rs.getInt("amt"); }
```



Connessione (1)

- Caricare il driver JDBC specifico del DBMS. Ad esempio per il database PostgreSQL:

```
try
{
  Class.forName("org.postgresql.Driver");
 }
catch (ClassNotFoundException cnfe)
{
  out("Driver jdbc non trovato: " + cnfe.getMessage());
 }
```

- Solitamente si verifica un'eccezione quando il class loader non riesce a trovare nel path la libreria postgresql.jar.

Connessione (2)

- Preparare l'URI del database, il nome utente e la password per l'autorizzazione al collegamento.
  - Ad esempio per collegarsi al database esercitazioni del DBMS presente su sqlserver, si possono preparare tre variabili String:

```
String uri =
   "jdbc:postgresql://sqlserver.sci.univr.it/dblab200";
String user = "userlab200";
String passwd = "xxx";
```

- L'URI è sempre composto da: "*jdbc:tipoDBMS:URLDatabase*"
  dove l'URL del database solitamente ha il formato //host/nomeDatabase

Connessione (3)

- Attivare una connessione con il metodo statico DriverManager.getConnection().

  Ad esempio:

```
Connection connection =
DriverManager.getConnection(uri,user,passwd);
```

- A questo punto l'oggetto connection rappresenta la connessione al database. Tutte le operazioni che si possono eseguire sul database sono date dai metodi di questo oggetto.

Esecuzione di interrogazioni

- Per definire una query esistono due classi: Statement e PreparedStatement.
- Uno Statement rappresenta una query semplice con tutti i dati specificati all'atto della creazione.
- Un preparedStatement permette di sviluppare uno schema di query (con parametri) che può essere utilizzato più volte con valori differenti.

Con Statement

```
String nomeAutore = “Umberto";
String cognomeAutore = “Eco";

Statement stmt;
ResultSet rs;

sql   = " SELECT * ";
sql += " FROM autore ";
sql += " WHERE nome = ' "+ nomeAutore + " ' ";
sql += " AND cognome = ' " + cognomeAutore + " ' ";

stmt = connection.createStatement();

rs=stmt.executeQuery(sql);
```

Con PreparedStatement

```
String nomeAutore = “Umberto”;
String cognomeAutore = “Eco”;

PreparedStatement pstmt;
ResultSet rs;

sql   = " SELECT * ";
sql += " FROM autore ";
sql += " WHERE nome = ? AND cognome = ?";

pstmt = con.prepareStatement(sql);
pstmt.clearParameters()
pstmt.setString(1, nomeAutore);
pstmt.setString(2, cognomeAutore);

rs=pstmt.executeQuery();
```

Esempio di Servlet (1)

```
import java.io.*;
    import java.util.*;
    import java.sql.*;
    import javax.servlet.*;
    import javax.servlet.http.*;

    /**
     * ServletQuery. Esempio di Servlet che si connette alla base di dati ed
     * esegue una semplice interrogazione.
     * I parametri vengono passati da una FORM HTML.
     *
     * @author Gabriele Pozzani, Barbara Oliboni
     * @version 1.0   06/05/2008
     */

    public class ServletQuery extends HttpServlet {

        public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {

            PrintWriter out = response.getWriter();
```

Esempio di Servlet (2)

```
    String sql;
    Connection con = null;
    PreparedStatement pstmt;
    ResultSet rs;

    String url = "jdbc:postgresql://sqlserver.sci.univr.it/dblab200";
    String user = “userlab200";
    String passwd = "";

    String cognomeAutore = "";
    String nomeAutore = "";

    /**
    * Caricamento del driver JDBC per il database
    */
    try {
            Class.forName("org.postgresql.Driver");
          } catch (ClassNotFoundException cnfe) {
            log("Driver jdbc non trovato: " + cnfe.getMessage());
    }
```

Esempio di Servlet (3)

```
response.setContentType("text/html; charset=ISO-8859-1");

out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\"");
out.println("      \"http://www.w3.org/TR/REC-html40/loose.dtd\">");


out.println("<html>");
out.println("<head>");
out.println("<title>Libri on line</title>");
out.println("</head>");
out.println("<body>");
out.println("<h1>Autore</h1>");

try {
            /**
             * Interrogazione parametrica per recuperare i dati di
             * un autore.
             */
            sql =  "SELECT * " +
                      "FROM autore " +
                      "WHERE cognome = ? AND nome = ?";
```

Esempio di Servlet (4)

```
      /**
             * Parametri da passare all'interrogazione
             */
            cognomeAutore = request.getParameter("cognome");
            nomeAutore = request.getParameter("nome");

            /**
             * Connessione alla base di dati
             */
            con = DriverManager.getConnection(url, user, passwd);
            pstmt = con.prepareStatement(sql);
            pstmt.setString(1, cognomeAutore);
            pstmt.setString(2, nomeAutore);

      /**
             * Esecuzione dell'interrogazione
             */
            rs=pstmt.executeQuery();
```

Esempio di Servlet (5)

```
 /**
     * Stampa dei parametri passati
     */
            out.println("<p>Ricerca dati dell'autore +
request.getParameter("cognome")
      + " " + request.getParameter("nome") + ". </p>");
     /**
     * Analisi del result set
     */
            while (rs.next()) {
                out.println("<hr>");
                out.println("<p>");
                out.println("<strong>Codice Fiscale:</strong>
"+rs.getString("codice_fiscale"));
                out.println("<br>");
                out.println("<strong>Cognome:</strong> "+rs.getString("cognome"));
                out.println("<br>");
                out.println("<strong>Nome:</strong> "+rs.getString("nome"));
                out.println("<br>");
                out.println("<strong>Data di Nascita:</strong>
"+rs.getDate("data_nascita"));
                out.println("<br>");
          if (rs.getDate("data_morte") != null){
                    out.println("<strong>Data di Morte:</strong>
"+rs.getDate("data_morte"));
                out.println("<br>");}
          out.println("</p>");
                out.println("<hr>");
            }
```

Esempio di Servlet (6)

```
con.close();
    } catch (SQLException sqle) {
            log("drivermanager non trovato: " + sqle.getMessage());

        out.println("<p>Errore nella connessione al database: " +
    sqle.getMessage()+"</p>");
    }

        out.println("</body>");
        out.println("</html>");
    }
}
```

FORM inserimento dati

```
<!-- saved from url=(0022)http://internet.e-mail -->
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta name="generator" content=
    "HTML Tidy for Linux/x86 (vers 1st November 2002), see www.w3.org">
    <title>Biblioteca Università di Verona</title>
    <style type="text/css">
   div.c2 {font-size: 51%; text-align: center}
   h1.c1 {text-align: center}
    </style>
  </head>
  <body>
<form method="get" action="/servlet/ServletQuery">
<H2>Cognome:</H2>
<input name="cognome" type="text" maxlength="40">
<H2>Nome:</H2>
<input name="nome" type="text" maxlength="40">
<input type="submit">
<form>
 </body>
</html>
```

Java Bean

- Un Java Bean è un *componente* nella tecnologia Java.
- Con il termine *componente* si indicano essenzialmente quelle classi che permettono di essere utilizzate in modo standard in più applicazioni. Lo scopo dei componenti infatti è dare la possibilità di *riutilizzare* in modo *sistematico* gli oggetti in contesti diversi, aumentando la produttività.
- La definizione di Java Bean è volutamente generica. Un Java Bean è semplicemente una classe Java che risponde a due requisiti:
  - ha un costruttore con zero argomenti;
  - implementa Serializable o Externalizable per essere persistente.

Java Data Bean (1)

- Una classe Java per essere utilizzata come Java Data Bean deve essere scritta seguendo le seguenti direttive:
  - **deve implementare un costruttore senza argomenti.**
    Esempio: AutoreBean();
  - **Per ciascun campo della classe che si vuole rendere visibile, deve essere implementato un metodo con la signature getNomeCampo() dove NomeCampo è il nome del campo (con le iniziali maiuscole).**
    Esempio: se una bean ha un campo numeroTelefono il metodo dovrà essere getNumeroTelefono().
  - **Per ciascun campo della classe che si vuole rendere modificabile, deve essere implementato un metodo con la signature setNomeCampo(ClasseCampo v) dove NomeCampo è il nome del campo (con le iniziali maiuscole) e ClasseCampo è il tipo del campo.**
    Esempio: se un bean ha un campo numeroTelefono di tipo String, il metodo dovrà essere setNumeroTelefono(String value).
- Inoltre per consuetudine si usa nominare la classe con il suffisso "Bean".

  Esempio: AutoreBean

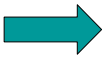
Java Data Bean (2)

- Un Java Data Bean risulta essere il miglior componente per mappare una tupla di un ResultSet in un oggetto Java.
- A tal fine, esso deve contenere:
  1. tanti campi **private** quanti sono gli attributi;
  2. un costruttore di default che assegna i valori di default ai campi;
  3. i metodi **pubblici** di accesso getter e setter per gli attributi che si vogliono esporre.

Java Data Bean (3)

Autore

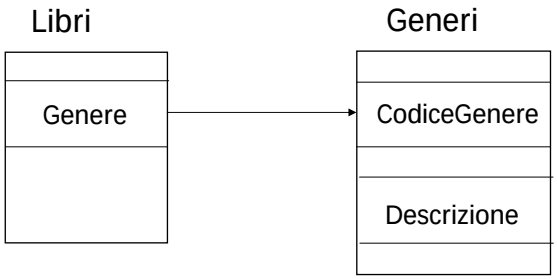
Codice Fiscale	Cognome	Nome		



```
public class AutoreBean {
private String CF, cognome, nome,
    …
…
AutoreBean(){
    CF = cognome =  nome = “”;
}
…
public String getCF(){
    return CF;
}
public void setCF(String c){
    …
}
```

Lavorare con i Bean (1)

- Non sempre il mapping uno a uno degli attributi di una tabella in un bean risulta essere il migliore approccio.
  Nel caso di join tra più tabelle è conveniente includere nel bean anche il valore (o i valori) provenienti da entrambe le tabelle.
  In tal modo esso risulta disponibile per la visualizzazione.



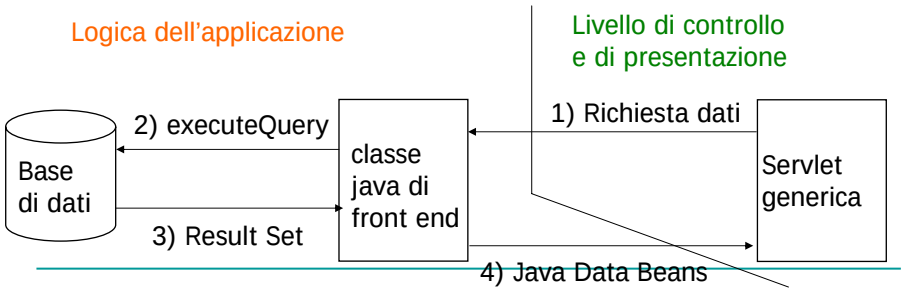
Lavorare con i Bean (2)

- Considerando il caso dell'esempio, è consigliabile completare il bean con l'attributo descrizione (e quindi con i metodi getDescrizione e setDescrizione nella classe LibroBean).
  Inoltre dovrà essere definito nella classe DBMS oltre al metodo makeLibroBean anche il metodo makeLibroBeanCompleto che riutilizza la definizione semplice aggiungendo l'attributo mancante:

```
private LibroBean makeLibroBeanCompleto(ResultSet rs)
      throws DBMSException {
      try {
              LibroBean bean = makeLibroBean(rs);
              bean.setDescrizione(rs.getString("descrizione"));
              return bean;
      }
       catch (SQLException e) {
              throw new DBMSException(e.getMessage());
      }
}
```

Basi di dati e Bean (1)

- I Java Data Bean sono dei componenti fondamentali nella strutturazione di una applicazione web che utilizzi una connessione ad una base di dati.
- I Java Data Bean, insieme ad una o più opportune classi java, permettono di separare la logica dell'applicazione dalla parte di controllo e di presentazione delle informazioni.



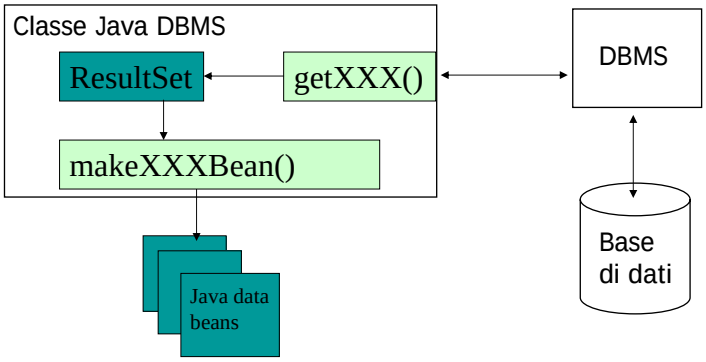
Basi di dati e Bean (2)

- La parte logica dell'applicazione è realizzata da un insieme di classi e un insieme di Java Data Bean che realizzano le interrogazioni e strutturano i risultati delle interrogazioni e altri dati in informazioni. Ad esempio, in una certa applicazione, tutte le interrogazioni che si possono fare alla base di dati possono essere raggruppate in un'unica classe, che fornirà i data bean (o array di data bean) come risultato di un'interrogazione.
- Il livello di presentazione e di flusso dell'applicazione è realizzata dalle servlet che analizzano le richieste, richiedono i Java Data Bean necessari per formare il documento di risposta (gestiscono il controllo) e preparano il documento con le informazioni richieste (gestiscono la presentazione).

Classe gestione interrogazioni

- Quando una applicazione web interagisce con una base di dati per recuperare le informazioni di interesse, tutte le query sono gestite all'interno di un'unica classe Java che, utilizzando un insieme opportuno di componenti Java Data Beans, realizza un'interfaccia tra le servlet e la base di dati.
- Assumiamo di chiamare tale classe DBMS.java. Un possibile struttura di questa classe è la seguente:
  1. Il costruttore di default deve caricare il driver del DBMS con cui ci si deve connettere.
  2. Ci devono essere tanti metodi quanti sono le possibili interrogazioni che si vogliono gestire. Ognuno dei metodi deve:
     - definire una connection;
     - associare eventuali parametri d'input con i parametri dell'interrogazione;
     - eseguire l'interrogazione;
     - creare un Java Data Bean o un array di Java Data Bean con i dati dalla query e restituirlo.

Classe DBMS



Classe DBMS.java

- Per strutturare meglio tale classe è opportuno creare dei metodi che, dato un result set, restituiscono il JDB/array JDB che rappresenta tale result set.
- Se si decide di rappresentare con un JDB ogni possibile result set, è possibile che il numero di JDB da definire cresca in modo significativo. Un approccio alternativo consiste nel definire JDB solo per tutte le entità e per le relazioni più importanti, offrire dei metodi per eseguire interrogazioni su queste entità/relazioni e lasciare alle servlet l'onere di combinare i risultati per ottenere risultati combinati.

- Un esempio di classe DBMS.java si trova nel pacchetto lezione7A.zip scaricabile dalla pagina del corso

Esempi da scaricare (1)

1. Scaricare nella directory ~/tomcat/webapps/biblio/WEB-INF/ il pacchetto Lezione7A.zip dalla pagina web del corso
2. Scompattare il pacchetto: tar xzvf Lezione7A.zip
3. Si otterrà la directory classes contenente la Servlet per la visualizzazione di tutti le mostre presenti nella tabella MOSTRA della base di dati e la directory biblio contenente la classe DBMS.java e il Bean relativo all'esempio.
4. Nella directory WEB-INF creare la directory lib
5. Dalla directory lib creare il link simbolico nel seguente modo

   ln -s /usr/share/java/postgresql-jdbc3.jar
6. Per far funzionare l'esempio è necessario modificare: la parte relativa alla connessione alla base di dati e la parte relativa alle tabelle da interrogare.
7. Per compilare i files contenuti nella directory biblio posizionarsi nella directory classes ed eseguire i seguenti comandi:
   - **javac ./biblio/MostraBean.java**
   - **javac ./biblio/DBMS.java**

Esempi da scaricare (2)

1. Scaricare nella directory ~/tomcat/webapps/ROOT/ il pacchetto Lezione7B.zip dalla pagina web del corso
2. Scompattare il pacchetto: tar xzvf Lezione7B.zip
3. Si otterrà il sorgente della Servlet ServletQuery.java e la relativa FORM di inserimento dati.
4. Per far funzionare l'esempio è necessario modificare la parte relativa alla connessione alla base di dati, inserendo la propria login.