Laboratorio di Basi di Dati e Multimedia

Laurea in Tecnologie dell'Informazione: Multimedia

Docente: Alessandra Di Pierro

Email: dipierro@sci.univr.it

Lezione 7

Java DataBase Connectivity

- JDBC consente di interfacciare una base di dati ed un programma java
  - 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

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)

- Statement: oggetto che, tramite una connessione, consente di inviare delle istruzioni SQL e di ricevere i risultati. Esistono due tipi di istruzioni:

  - Statement: utilizzata per eseguire interrogazioni SQL statiche. Un'istruzione Statement può essere creata con Connection.createStatement()

  - PreparedStatement: estensione di Statement che utilizza codice SQL precompilato con parametri di input definiti in modo dinamico. Permette 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) di un oggetto Statement o PreparedStatement.
  - Metodo next() per iterare fra le righe di un ResultSet e metodi getxxx() per estrarre il valore delle colonne dove xxx e' il tipo di dati Java

- SQLException: classe base per eccezioni utilizzata dall'API JDBC. Offre metodi che possono fornire il valore SQLState per ogni codice d'errore specifico del database

Operazioni di base di JDBC

1. Caricamento del driver JDBC

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

2. Apertura connessione con il DB

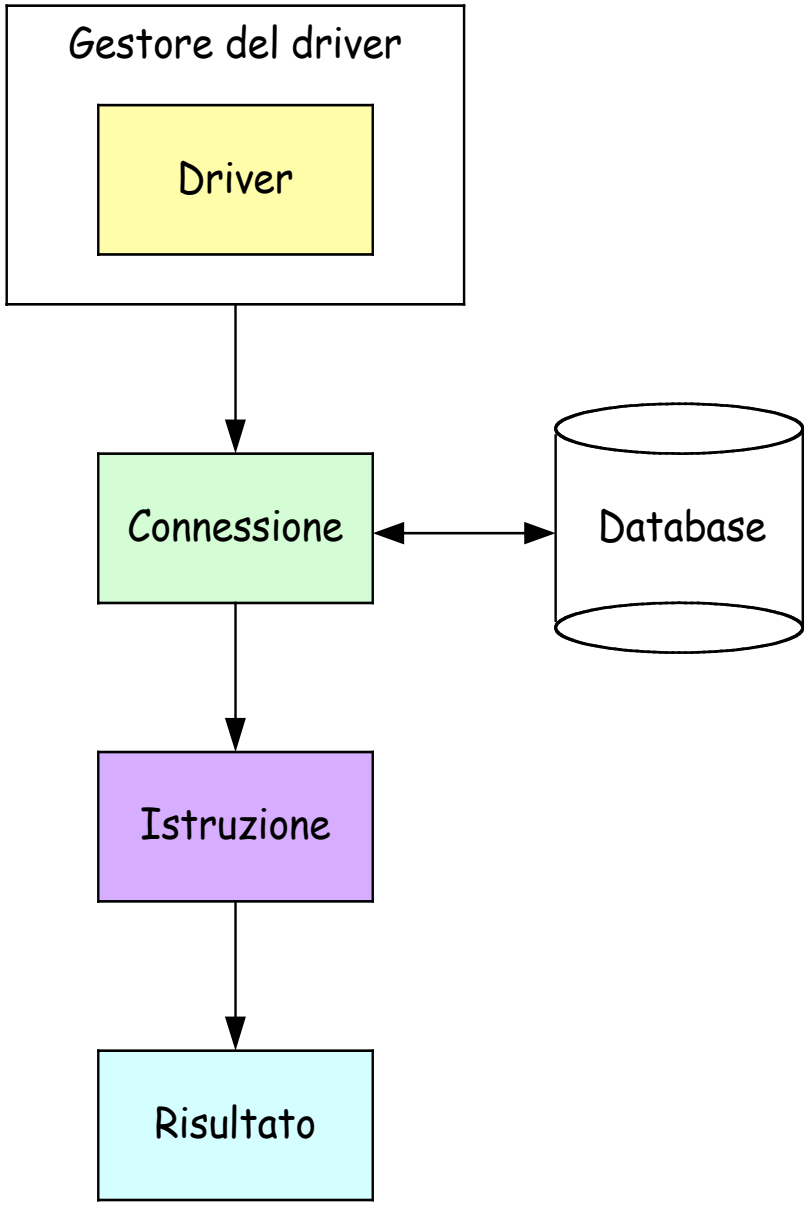
```
DriverManager.getConnection
  (jdbc:tipoDBMS://URL/Database)
```

3. Esecuzione di una query 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/dblabxx";
String user = "postgres login";
String passwd = "";
```

- 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

  - Statement: rappresenta una query semplice con tutti i dati specificati all'atto della creazione

  - PreparedStatement: permette di sviluppare uno schema di query (con parametri) che può essere utilizzato più volte con valori differenti

NB: Per dettagli consultare
http://java.sun.com/j2se/1.3/docs/api/java/sql/PreparedStatement.html

Esempio: Uso di Statement

```
String nomeCantante = "Vasco";
String cognomeCantante = "Rossi";

Statement stmt;
ResultSet rs;

sql   = " SELECT * ";
sql += " FROM cantante ";
sql += " WHERE nome = ' "+ nomeCantante + " ' ";
sql += " AND cognome = ' " + cognomeCantante + " ' ";

stmt = connection.createStatement();

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

Esempio: Uso di PreparedStatement

```
String nomeCantante = "Vasco";
String cognomeCantante = "Rossi";

PreparedStatement pstmt;
ResultSet rs;

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

pstmt = con.prepareStatement(sql);
pstmt.clearParameters()
pstmt.setString(1, nomeCantante);
pstmt.setString(2, cognomeCantante);

rs=pstmt.executeQuery();
```

Esempi di Servlet

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.
*/

public class ServletQuery extends HttpServlet {

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


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

…continua…

Esempio di Servlet (2)

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

String url = "jdbc:postgresql://sqlserver.sci.univr.it/dblabxxx";
String user = "userlabxxx";
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());
}
```

…continua…

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("<Negozio di dischi</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 = ?";
```

…continua…

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();
```

…continua…

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>");
        }
```

…continua…

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 per inserimento dati

```
<!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> Negozio di dischi </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>
```

Esempio da scaricare

1. Scaricare nella directory ~/tomcat/webapps/ROOT/ il pacchetto Lezione7BMM.tgz dalla pagina web del corso
2. Scompattare il pacchetto: tar xzvf Lezione7BMM.tgz
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 e passw.

Servlet e immagini

Dati multimediali

Richiedono molti bytes per essere rappresentati, quindi e' necessario memorizzarli e trasmetterli in forma compressa.

- Per le immagini il formato piu' usato e' JPEG (Joint Picture Expert Group).
- MPEG sfrutta le similarita' tra i frames di una sequenza per ottenere un elevato grado di compressione. Esistono varie versioni. MPEG-1 comprime 1 minuto di 30 frames per secondo di video e audio in 12,5 megabytes (VHS), invece dei 75 richiesti da JPEG. MPEG-2 ne usa 17 ma introduce una migliore qualita' video (DVD). MPEG-4 usa tecniche per ottenere una migliore compressione.

Bytea

- In PostgreSQL possiamo utilizzare il dominio bytea per dati multimediali come ad esempio immagini o video.
- Il tipo di dati bytea permette di memorizzare stringhe binarie, cioe' sequenze di bytes.
- Le stringhe binarie si distinguono dalle stringhe di caratteri perche' consentono di codificare anche valori che non sono ammessi dalla codifica dei caratteri scelta per il DB. Inoltre le operazioni sono operazioni generiche su bytes e non dipendono dalla codifica scelta per i caratteri.

Esempio: Servlet per il caricamento di immagini (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
 * inserisce un'immagine nella base di dati.

public class ServletStore extends HttpServlet {

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

     Connection con = null;
```

Continua ….

Esempio: Servlet per il caricamento di immagini (2)

```
String url = "jdbc:postgresql://sqlserver.sci.univr.it/dblabxxx";
String user = "userlabxxx";
String passwd = "";

File f = new File("idefix.jpg");
FileInputStream fis = new FileInputStream(f);

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

Continua ….

Esempio: Servlet per il caricamento di immagini (3)

```
try {
    /**
     * Connessione alla base di dati
     */

    con = DriverManager.getConnection(url, user, passwd);
    String str = "INSERT INTO prova VALUES (1, ?)";

    PreparedStatement pst = con.prepareStatement(str);
    pst.clearParameters();
    pst.setBinaryStream(1, (InputStream)fis, (int)f.length());
    pst.execute();
    con.close();
```

Continua ….

Esempio: Servlet per il caricamento di immagini (4)

```
        PrintWriter out = response.getWriter();
        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>Caricamento immagine</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Immagine</h1>");

      } catch(Exception e) {
        e.printStackTrace();
      }
    }
}
```

Esempio: Servlet per la visualizzazione di immagini (1)

```
import java.io.*;
import java.util.*;
import java.sql.*;

import javax.servlet.*;
import javax.servlet.http.*;

import com.oreilly.servlet.MultipartRequest;
import com.sun.image.codec.jpeg.*;
import java.awt.image.*;

public class ServletShow extends HttpServlet {

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

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

Continua ….

Esempio: Servlet per la visualizzazione di immagini (2)

```
String url = "jdbc:postgresql://sqlserver.sci.univr.it/dblabxxx";
String user = “userlabxxx";
String passwd = "";

/**
 * Caricamento del driver JDBC per il database
 */

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

Continua ….

Esempio: Servlet per la visualizzazione di immagini (3)

```
try {
   sql =  "SELECT * " +
          "FROM prova " +
          "WHERE codice = 1";
    /**
     * Connessione alla base di dati
     */

     con = DriverManager.getConnection(url, user, passwd);
   pstmt = con.prepareStatement(sql);

    /**
     * Esecuzione dell'interrogazione
     */

    rs=pstmt.executeQuery();

   response.setContentType("image/jpeg");
```

Continua ….

Esempio: Servlet per la visualizzazione di immagini (4)

```
            while (rs.next()) {
            InputStream is = rs.getBinaryStream("img");
            JPEGImageDecoder decoder = JPEGCodec.createJPEGDecoder(is);
            BufferedImage bi = decoder.decodeAsBufferedImage();

            ServletOutputStream sos = response.getOutputStream();
            JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(sos);
            encoder.encode(bi);
        }
        con.close();
    }catch (SQLException sqle) {
        sqle.printStackTrace();
    }

  }
}
```

Java Bean

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

Java Data Bean (1)

- La maggior parte dei Java Bean offre delle proprietà
- Le proprietà sono attributi del Java Bean per i quali sono previsti metodi di lettura e/o scrittura

  – Il metodo di lettura per una proprietà è il metodo pubblico *getNomeProprieta*() dove *NomeProprieta* è il nome della proprietà in cui la prima lettera è convertita con lettera maiuscola

  – Il metodo di scrittura per una proprietà è il metodo pubblico *setNomeProprieta*()

Java Data Bean (2)

- Una classe Java per essere usata come Java Data Bean deve seguire le seguenti direttive:
  - Implementa un costruttore senza argomenti
    (es. `AutoreBean()`)
  - Per ciascun campo che si vuole rendere visibile deve essere implementato un metodo `getNomeCampo()`
  - Per ciascun campo che si vuole rendere modificabile deve essere implementato un metodo `setNomeCampo(ClasseCampo v)` dove `ClasseCampo` indica il tipo del campo

Java Data Bean (3)

Esempio: se una bean ha un campo numeroTelefono di tipo string allora dovrà essere implementato il metodo

- getNumeroTelefono() per rendere visibile un numero telefonico
- setNumeroTelefono(String value) per rendere modificabile un numero telefonico

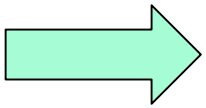
Java Data Bean (4)

- 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:
  - tanti campi **private** quanti sono gli attributi
  - un costruttore di default che assegna i valori di default ai campi
  - i metodi **pubblici** di accesso `getter` e `setter` per gli attributi che si vogliono esporre

Java Data Bean (5)

Autore

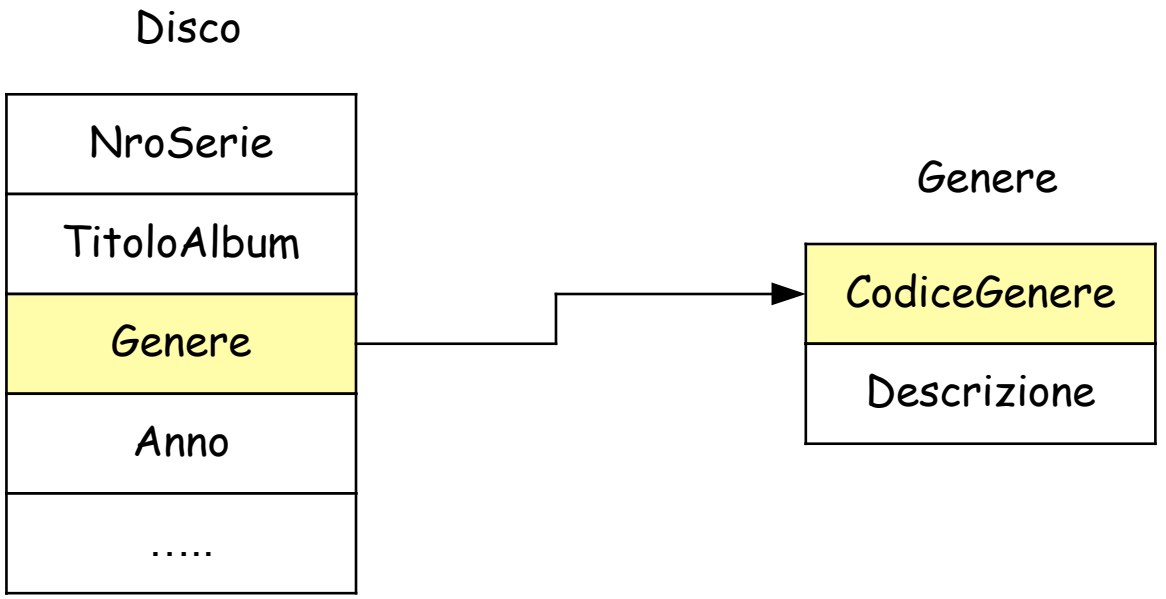
CFiscale	Cognome	Nome		Foto



```
public class AutoreBean {
private String CF, cogn, nom, …
…
AutoreBean(){
      CF = cogn = nom = "";
}
…
public String getCF(){
      return CF;
}

public InputStream getFoto(){
      return Foto;
}

public void setCF(String c){
      …
}

public void setFoto(InputStream f){
      …
}
```

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.



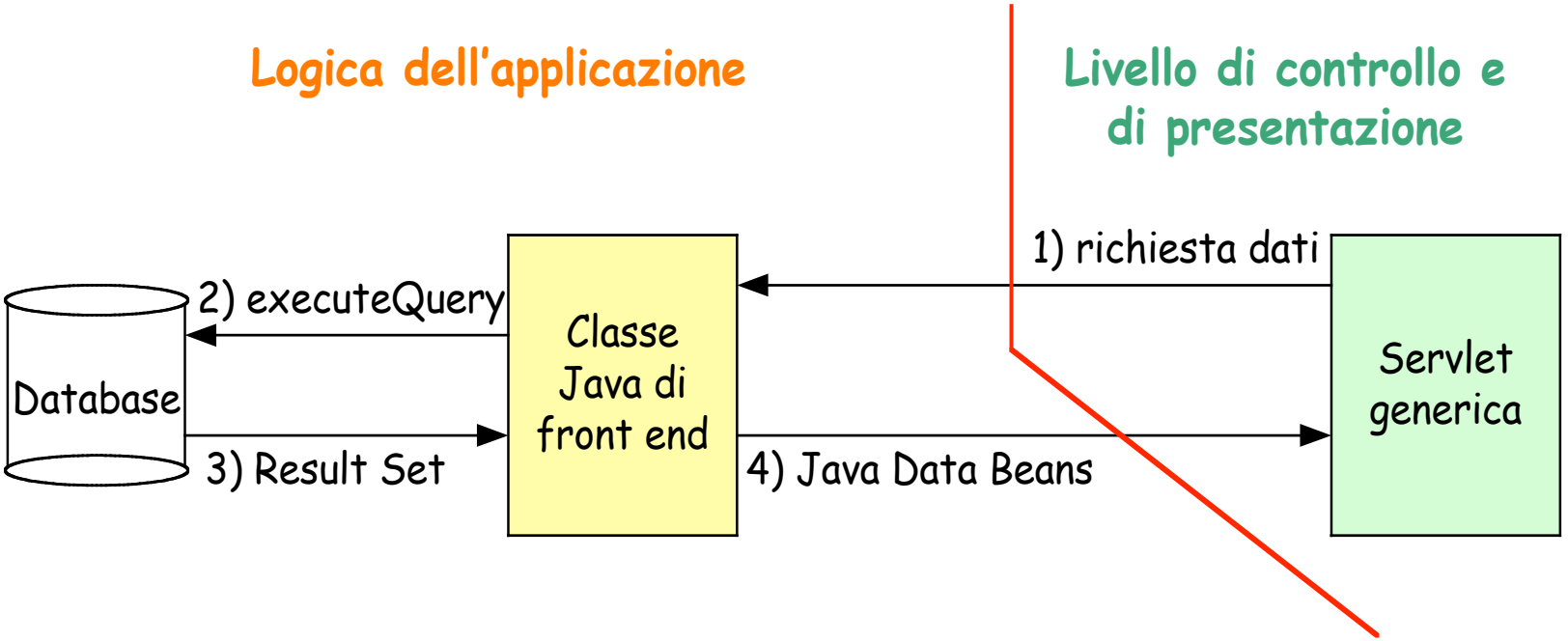
Lavorare con i Bean (2)

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

```
private DiscoBean makeDiscoBeanCompleto(ResultSet rs)
        throws DBMSException {
        try {
                DiscoBean bean = makeDiscoBean(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 (1)

- 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

- Tale classe, utilizzando un insieme opportuno di componenti Java Data Beans, realizza un'interfaccia tra le servlet e la base di dati

Classe gestione interrogazioni (2)

- Assumiamo di chiamare tale classe DBMS.java. Una possibile struttura di questa classe è la seguente:

  - Il costruttore di default deve caricare il driver del DBMS con cui ci si deve connettere
  - Ci devono essere tanti metodi quanti sono le possibili interrogazioni che si vogliono gestire. Ogni metodo 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.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 Immagini.tgz scaricabile dalla pagina del corso

Esempi da scaricare

1) Scaricare nella directory ~/tomcat/webapps/ il pacchetto Immagini.tgz dalla pagina web del corso

2) Scompattare il pacchetto: tar xzvf Immagini.tgz

3) Si otterrà la directory classes contenente:
   - **la Servlet per la visualizzazione di tutti i generi presenti nella tabella Genere della base di dati**
   - **la Servlet ServletStore per il caricamento delle immagini nella base di dati e la Servlet ServletShow per la visualizzazione di immagini provenienti dalla base di dati**
   - **la directory biblio contenente la classe DBMS.java e il Bean relativo all'esempio**
   - **la directory lib contenente il driver e la libreria necessari per la gestione delle immagini**

Funzionamento esempi (1)

- Per far funzionare l'esempio relativo ai Generi è necessario:
  - modificare la parte relativa alla connessione alla base di dati, inserendo la propria login e password
  - modificare il path di idefix.jpg (deve essere quello a partire dalla directory da cui e' stato lanciato Tomcat)
  - compilare i files contenuti nella directory biblio posizionandosi nella directory classes ed eseguendo i seguenti comandi:
    - javac ./biblio/*.java

Funzionamento esempi (2)

- Per far funzionare l'esempio relativo alle immagini, prima di compilare le servlet è necessario fare in modo che il CLASSPATH contenga anche la directory lib e dunque è necessario modificare il file .bashrc come segue:

```
CLASSPATH=$CLASSPATH:/usr/local/java/sdk/jre/lib/rt.jar
CLASSPATH=$CLASSPATH:~/.../tomcat/webapps/Immagini/WEB-INF/lib/cos.jar
CLASSPATH=$CLASSPATH:~/.../tomcat/webapps/Immagini/WEB-INF/classes
export CLASSPATH
```

- posizionarsi nella directory ~/tomcat/webapps/biblio/WEB-INF/classes/
- compilare le servlet

NB: il path dell'immagine è relativo alla directory da cui si lancia Tomcat