

**Universita' degli Studi di Bologna**  
**Facolta' di Ingegneria**

**Anno Accademico 2008-2009**

**Laboratorio di Tecnologie Web**

**Pagine JSP**

**Pattern DAO**

**<http://www-lia.deis.unibo.it/Courses/TecnologieWeb0809>**

## Concetti fondamentali

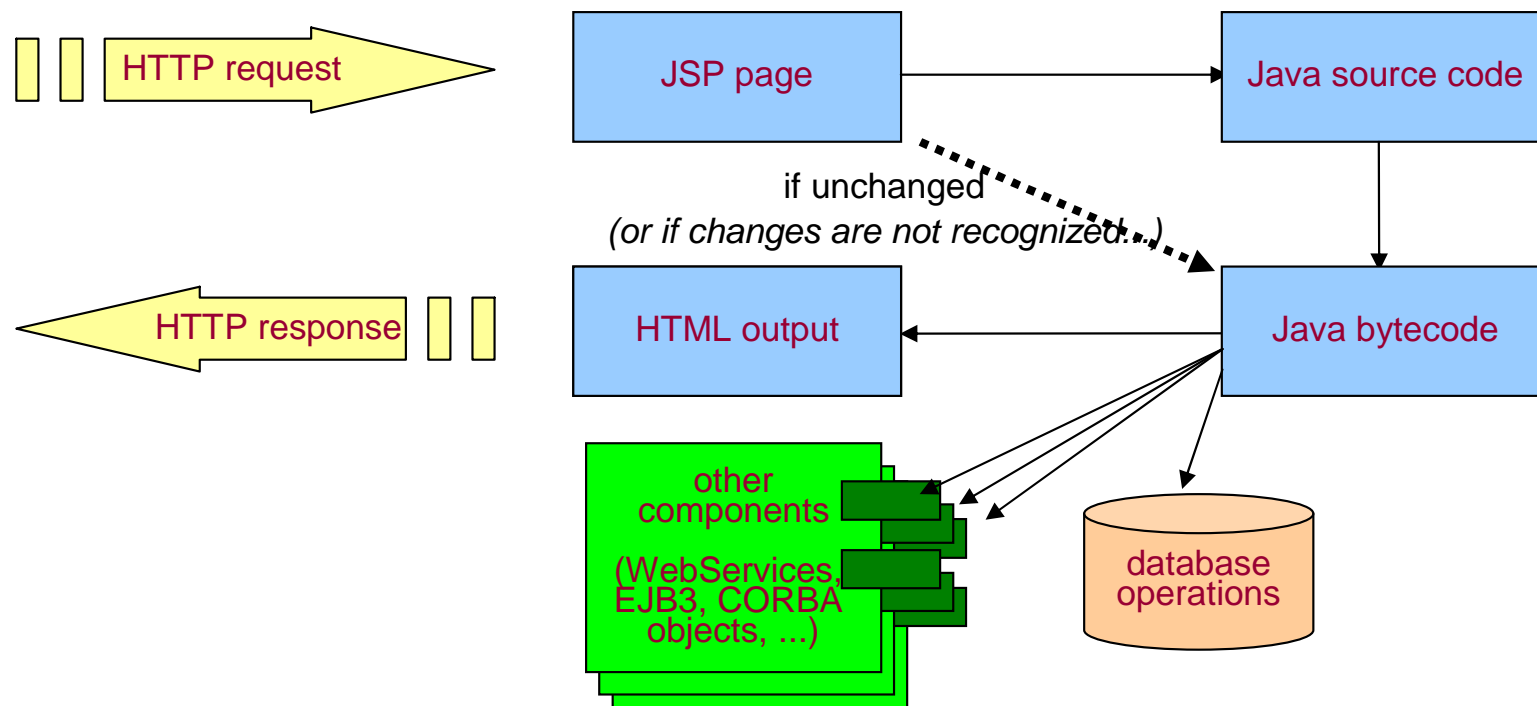
---

- Pagine “HTML” che includono codice Java
  - elaborate lato server, ma **non interpretate** bensì:
    - **tradotte in Servlet** al volo alla prima richiesta
      - Tomcat salva su file i `.java` che genera automaticamente!
      - Si può curiosare:  
`$TOMCAT_HOME/work/Catalina/localhost/$WEBAPP_NAME/org/apache/jsp/$PATH_TO_PAGE/$PAGE_NAME.java`
    - **compilate**
      - Tomcat salva su file anche i `.class`
    - **eseguite** come normali Servlet a seguito di ogni richiesta per ottenere:
      - **logica di business** lato server
        - operazioni su sessione, parametri della richiesta, cookie, ...
        - modifica di un database, ...
        - ecc...
      - **generazione dinamica dell'HTML** restituito al browser
  - l'HTML che giunge sul browser in risposta alle richieste per tali pagine non coincide più (ovviamente) con il loro codice sorgente
    - confrontate i file in Eclipse con il “Visualizza sorgente pagina di firefox”



## Funzionamento lato server

- Le pagine JSP vengono gestite su Tomcat da una particolare Servlet
  - JspServlet
    - automaticamente “associata” a tutte le richieste per risorse di tipo *.jsp*
    - traduce il loro sorgente testuale (ibrido: HTML + Java) in codice sorgente di una classe Java che estende, attraverso la classe `HttpJspBase`, la classe `HttpServlet`



## 4 costrutti

### ▪ **Direttive:** `<%@ ... %>`

- definizione di proprietà della pagina, import di classi a cui si fa riferimento, inclusione di librerie di tag aggiuntive, inclusione di altri documenti, ecc...
- **valutate a tempo di compilazione**

### ▪ **Dichiarazioni:** `<%! ... %>`

- definizione di variabili e metodi utilizzabili nel resto della pagina
- l'equivalente di variabili membro e metodi non statici per una Servlet

### ▪ **Scriptlet:** `<% ... %>`

- codice valutato via via che il server genera la risposta per il client
- l'equivalente del codice nei metodi *doGet()* e *doPost()* di una Servlet
  - ...confrontate il sorgente di una *.jsp* con i *.java* generati da Tomcat!
- **devono complessivamente costituire del codice Java ben formato**
  - è possibile aprire un blocco (“{“) e chiuderlo (“}”) in scriptlet diversi, divisi da codice HTML”...
  - ...o anche scrivere il codice di un *if* in uno scriptlet, chiudere lo scriptlet, scrivere del codice HTML, quindi aprire un nuovo scriptlet e scrivere in esso il codice dell'*else*
- **anche il codice HTML che li intervalla deve risultare, alla fine, ben formato**
  - ...se ad esempio si apre un tag `<p>` prima dello scriptlet con un *if* e lo si conclude all'interno sia dell' *if* che dell' *else*, ci saranno due `</p>` nel sorgente, ma solo uno arriverà nell'HTML finale restituito al browser!

E' il motivo per cui **non si possono dichiarare metodi dentro uno scriptlet!** Sarebbe come definire un metodo all'interno di un altro metodo in una classe Java!

### ▪ **Espressioni** `<%= ... %>`

- espressioni Java il cui risultato è direttamente inserito nel codice HTML circostante
- **non istruzioni:** non c'è il “;” alla fine !

## 4 costrutti (versione con delimitatori XML)

---

- Modalità alternativa di dichiarare gli stessi costrutti appena visti
  - seguendo la sintassi XML
  - scrivendo pagine JSP i cui sorgenti sono documenti XML ben formati

- **Direttive:**

`<jsp:directive.type attribute />`

- **Dichiarazioni:**

`<jsp:declaration> methods & members definitions </jsp:declaration>`

- **Scriptlet:**

`<jsp:scriptlet> java code </jsp:scriptlet>`

- **Espressioni:**

`<jsp:expression> java expression </jsp:expression>`

## 6 azioni

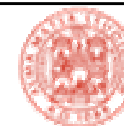
---

- Comandi JSP **valutati a tempo di esecuzione** della richiesta

*<jsp:nomeComando attributiComando ... />*

- Lista:

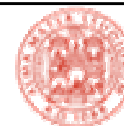
- **useBean**: istanzia un oggetto conforme alle convenzioni JavaBean e lo rende disponibile al codice che segue tramite un preciso identificativo e un preciso scope di validità
- **getProperty**: ritorna in forma di oggetto la property indicata
- **setProperty**: imposta il valore della property indicata
- **include**: include a request time (non a compile time, come le direttive) il contenuto di un file nel sorgente della JSP valutato dal server
- **forward**: cede la gestione della richiesta a un'altra risorsa
- **plugin**: genera il contenuto necessario per scaricare un plug-in Java



## 8 oggetti built-in

---

- Sono risorse...
  - ...rese automaticamente disponibili dal servlet container
  - ...accessibili **semplicemente per nome** all'interno del codice della pagina
  - ...equivalenti agli oggetti omologhi di norma accessibili all'interno di una Servlet
- Lista:
  - **page** (la pagina e le sue proprietà)
  - **config** (dati di configurazione)
  - **out** (il print writer su cui scrivere l'HTML della response)
  - **request** (la richiesta HTTP ricevuta e i suoi attributi, header, cookie, parameteri, ecc...)
  - **response** (la risposta HTTP e le sue proprietà)
  - **application** (dati condivisi da tutte le pagine della web application)
  - **session** (dati specifici della sessione utente corrente)
  - **exception** (eventuali eccezioni lanciate dal server; utile per pagine di errore)
  - **pageContext** (dati di contesto per l'esecuzione della pagina)



## Approfondimento - Tag library (taglib)

---

- Funzionalità aggiuntive...
  - ...rese disponibili sotto forma di librerie di tag
  - ...utilizzabili all'interno della pagina a seguito della direttiva

```
<%@ taglib uri="mnemonico dichiarato dalla tag library"  
    prefix="prefisso usato nella pagine per marcare i tag di tale libreria" %>
```
  
- Esempi: richiamare attraverso la dichiarazione di tag di una tag library...
  - ...frammenti di HTML piu' complessi
  - ...funzionalità fornite da librerie Java (es: JSTL)
    - **non ci sarà all'esame**
    - se comunque volete curiosare, Tomcat mostra alcuni esempi
    - è interessante, davvero! 😊
  - ...funzionalità fornite da componenti Java personalizzati



## Expression language (EL)

---

- Valutazione di espressioni racchiuse all'interno dei caratteri `${ ... }`
- Accesso efficace alle proprietà dei JavaBean dichiarati
  - accesso ai campi dei bean semplicemente per nome (in maniera analoga a come si accede alle proprietà degli oggetti Javascript)
  - risoluzione trasparente dei corrispondenti metodi getter/setter

`${ miobean.miocampo }`

- Risultato sostituito...
  - ...direttamente nell'HTML (scrittura del contenuto finale della pagina)
  - ...come valore degli attributi dei tag di una taglib (valutazione da parte del processore JSP)

`<c:if test="${ miobean.miocampo > 0 }">`

`...`

`</c:if>`

# Esempi di pagine JSP

- Tomcat fornisce out-of-the-box alcuni esempi di come utilizzare le API JSP
  - utili soprattutto come **cheat sheet**
    - dispersive per imparare
    - valide come riferimento veloce per sapere come fare operazioni precise...
  - visionabili a partire da:

<http://localhost:8080/jsp-examples>

codice sorgente completo

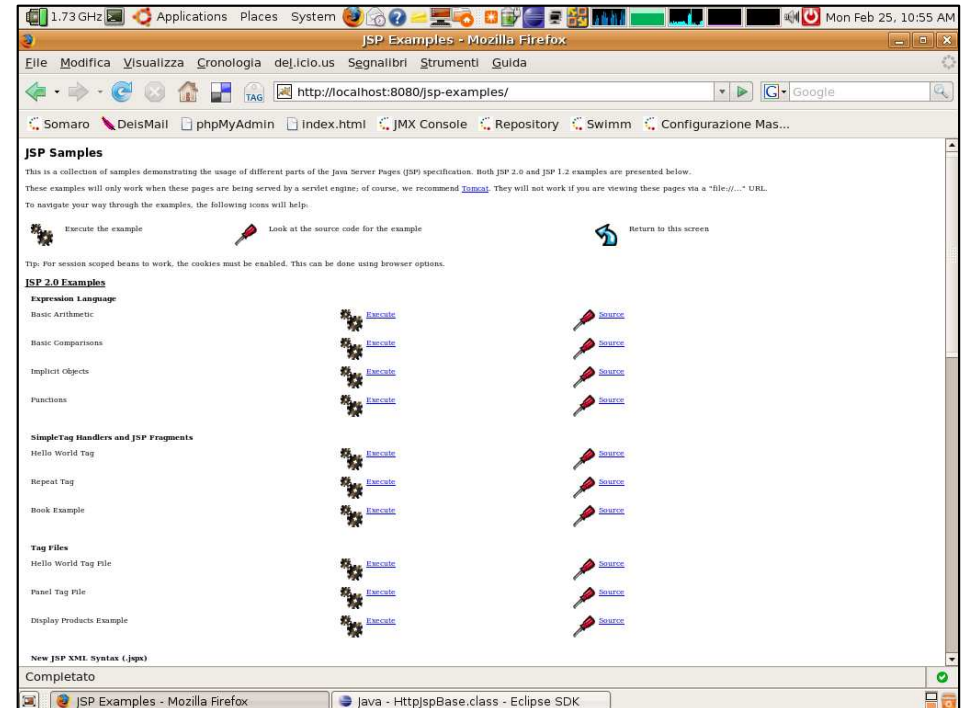
disponibile su *file system*,

nella corrispondente directory

di deployment

```
$TOMCAT_HOME/
```

```
webapps/jsp-examples/...
```



## Progetto di esempio

---

- Scaricare (dal sito del corso)...
  - ...importare (come *'existing project'* in Eclipse)...
  - ...configurare (modificando *environment.properties*)...
  - ...deployare (lanciando il target di ANT *'deploy.as.XXX'*)...
  - ...il progetto di esempio [TemplateJSPandDAO.zip](#)
- 
- Funziona tutto?



## [index.jsp](#)

### ▪ direttive:

- inclusione nella pagina del contenuto di altri documenti testuali:
  - menu
  - footer
- affidamento della gestione di eventuali eccezioni a una pagina specifica

### ▪ dichiarazioni

- metodi di utilità
- variabili della pagina

### ▪ scriptlet

- generazione di contenuto via invocazione dei metodi di cui sopra

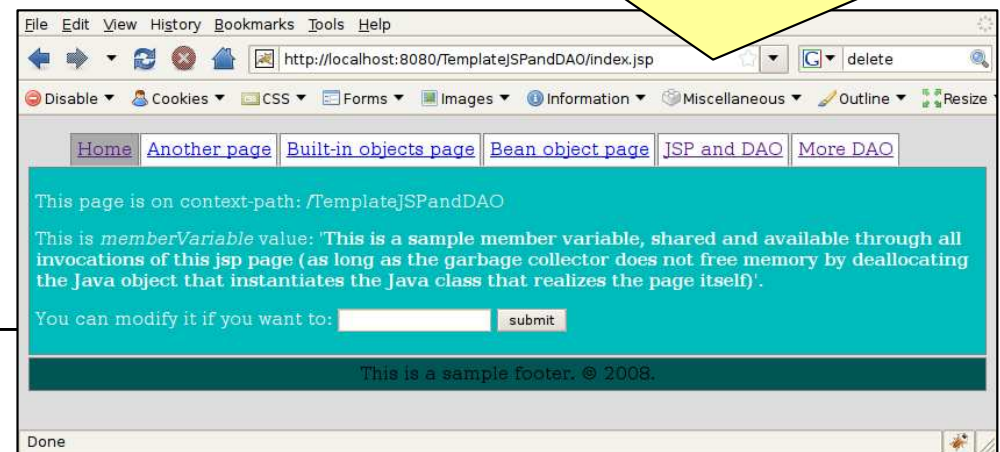
### ▪ ...frizzi e lazzi:

- al caricamento della pagina ricerca la linguetta del menu' che la contraddistingue e le cambio il colore di sfondo, in modo da evidenziare la pagina corrente

- settate il valore nella form
- navigate altrove e tornate qui
- aprite la pagina con un altro browser

### IL VALORE NON SI RESETTA !!

- è definito all'interno di una dichiarazione (come una variabile membro della Servlet che corrisponde a questa pagina JSP)
- se non interviene il garbage collector per liberare memoria sul server, l'oggetto Java di tipo JSP/Servlet istanziato per servire la prima richiesta servirà anche le successive (non vengono istanziate sue repliche)
- se invece serve liberare memoria, il server rimuove le istanze di oggetti non utilizzate (ergo non è questo il sistema per condividere informazioni tra più utenti! Dichiarate piuttosto dei *jsp:bean* con scope di applicazione)



## failure.jsp

- *Easter egg*: se viene inserito il valore '666' nella *form* della pagine *index.jsp*, la successiva submission dei dati della form causerà il lancio di una eccezione



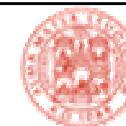
Tomcat rileva l'eccezione e, come indicato nelle direttive della pagina *index.jsp*, esegue il forward della richiesta (e dell'eccezione) alla pagina *failure.jsp*

- **direttive**

- dichiarazione che trattasi di pagina di errore
- si ottiene in questo modo accesso all'oggetto *exception* di tipo built-in

- ...ad esempio per stamparne lo *stacktrace* come in questo caso
- ...è invece buona norma gestire le eccezioni in modo da evitare output e comunicazioni "tecniche" all'utente

```
An exception was raised!  
java.lang.RuntimeException: You entered 666!!!  
  
Exception message is:  
You entered 666!!!  
  
Stacktrace is:  
java.lang.RuntimeException: You entered 666!!! at org.apache.jsp.index.jsp.modifyVariable(index.jsp:21) at  
org.apache.jsp.index.jsp.jspService(index.jsp:124) at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:97) at  
javax.servlet.http.HttpServlet.service(HttpServlet.java:802) at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:334) at  
org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:314) at org.apache.jasper.servlet.JspServlet.service(JspServlet.java:264) at  
javax.servlet.http.HttpServlet.service(HttpServlet.java:802) at  
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:252) at  
org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173) at  
org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:213) at  
org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:178) at  
org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:126) at  
org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:105) at  
org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:107) at  
org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:148) at  
org.apache.coyote.http11.Http11Processor.process(Http11Processor.java:869) at  
org.apache.coyote.http11.Http11BaseProtocol$Http11ConnectionHandler.processConnection(Http11BaseProtocol.java:664) at  
org.apache.tomcat.util.net.PoolTcpEndpoint.processSocket(PoolTcpEndpoint.java:527) at  
org.apache.tomcat.util.net.LeaderFollowerWorkerThread.run(LeaderFollowerWorkerThread.java:80) at  
org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run(ThreadPool.java:684) at java.lang.Thread.run(Thread.java:619)
```



## [another.jsp](#)

- Questa volta la variabile settata tramite la form è definita all'interno di uno scriptlet, non di una dichiarazione
- L'effetto è quello di avere una variabile in *stack* al momento dell'esecuzione del metodo *service()* della Servlet derivata dalla JSP:
  - cancellato al termine di ogni esecuzione
  - richieste successive verso la stessa pagina non ritrovano il valore precedentemente impostato





## Codice a confronto

- Il codice delle Servlet generate da Tomcat in corrispondenza delle richieste alle pagine *index.jsp* e *another.jsp* è visibile su file system in:

\$TOMCAT\_HOME/work/Catalina/localhost/TemplateJSPandDAO/org/apache/jsp

- In caso di dubbi può essere utile aprire i sorgenti e analizzare come vengono tradotti i diversi costrutti JSP

```
*index_jsp.java (/opt/apache-tomcat-5.5.25/work/Catalina/localhost/TemplateJSPandDAO/org/apache/jsp) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
es5_trace *index_jsp.java
import javax.servlet.jsp.*;

public final class index_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

private String memberVariable =
    /*This is a sample member variable, shared and available through all invocations of this jsp page * +
    *(as long as the garbage collector does not free memory by deallocating the Java object that instantiates * +
    *the Java class that realizes the page itself)*;

private void modifyVariable(String aVar)
    throws RuntimeException {
    if ( aVar != null && ! aVar.equals("") ) {
        memberVariable = aVar;
        if ( aVar.equals("666") )
            throw new RuntimeException("You entered 666!!!");
    }
}

/* ... */

public void _jspService(HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, ServletException {

/* ... */
    out.write("<!doctype html public "-//w3c//dtd html 4.0 transitional//en">\r\n");
    out.write("\n");
    out.write("\n");
    out.write("\n");
    out.write("\n");
    out.write("\n");
    out.write("<html>\r\n");
    out.write("<head>\r\n");
    out.write("<meta name='Author' content='pisi79'>\r\n");
    out.write("<title>Home_JSP</title>\r\n");
}
Ln 7, Col 75 INS
```

```
*another_jsp.java (/opt/apache-tomcat-5.5.25/work/Catalina/localhost/TemplateJSPandDAO/org/apache/jsp/otherpages) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
es5_trace *index_jsp.java *another_jsp.java
import javax.servlet.jsp.*;

public final class another_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

/*...*/

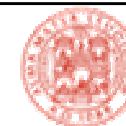
public void _jspService(HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, ServletException {

/*...*/
    try {
        _jspxFactory = JspFactory.getDefaultFactory();
        response.setContentType("text/html; charset=US-ASCII");
        pageContext = _jspxFactory.getPageContext(this, request, response,
            "/errorpages/failure.jsp", true, 8192, true);
        _jspx_page_context = pageContext;
        application = pageContext.getServletContext();
        config = pageContext.getServletConfig();
        session = pageContext.getSession();
        out = pageContext.getOut();
        _jspx_out = out;

        out.write("<!doctype html public "-//w3c//dtd html 4.0 transitional//en">\r\n");
        out.write("\n");
        out.write("\n");
        out.write("\n");

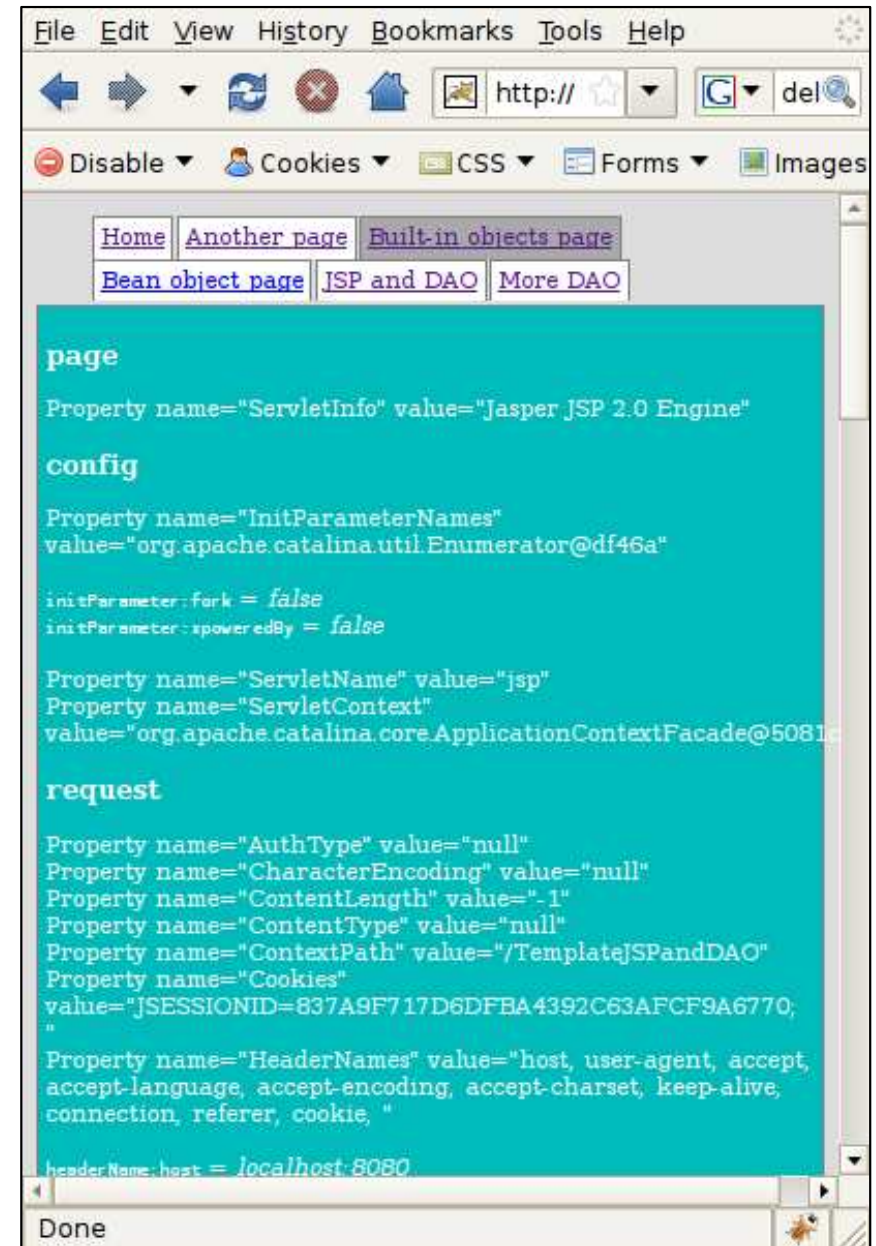
String methodVariable = "This is a method variable, available through all this jsp page, while producing the HTML result";

        out.write("\n");
        out.write("\n");
        out.write("<html>\r\n");
        out.write("<head>\r\n");
}
Ln 11, Col 1 INS
```



## [builtin.jsp](#)

- Una lunghisssssssima sfilza di stampe a video dei (piu' significativi) valori...
  - ...accessibili attraverso i 6 oggetti *built-in*
  - ...dall'interno di *scriptlet* ed *espressioni*
- Utile come riferimento di
  - cosa c'è
  - come vi si accede
- E basta.



```
page
Property name="ServletInfo" value="Jasper JSP 2.0 Engine"

config
Property name="InitParameterNames"
value="org.apache.catalina.util.Enumerator@df46a"

initParameter.fork = false
initParameter.sponsoredBy = false

Property name="ServletName" value="jsp"
Property name="ServletContext"
value="org.apache.catalina.core.ApplicationContextFacade@5081c"

request
Property name="AuthType" value="null"
Property name="CharacterEncoding" value="null"
Property name="ContentLength" value="-1"
Property name="ContentType" value="null"
Property name="ContextPath" value="/TemplateJSPandDAO"
Property name="Cookies"
value="JSESSIONID=837A9F717D6DFBA4392C63AFCF9A6770;"
"
Property name="HeaderNames" value="host, user-agent, accept,
accept-language, accept-encoding, accept-charset, keep-alive,
connection, referer, cookie, "
headerName.host = localhost:8080
```



## bean.jsp

- ...e pagine collegate
  - utilizzo di oggetti Java all'interno delle pagine JSP
  - gestione di scope differenti in funzione del tipo di semantica desiderata

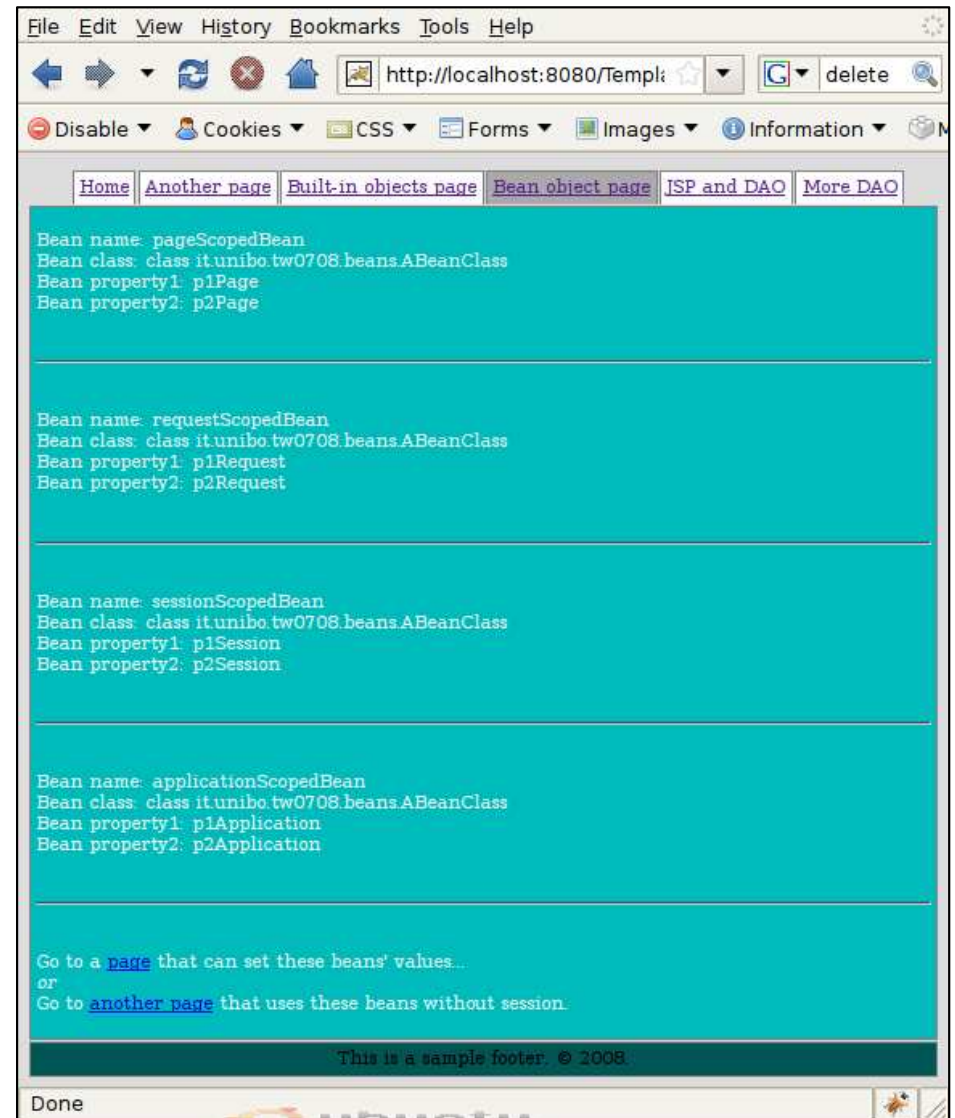
*page*

*request*

*session*

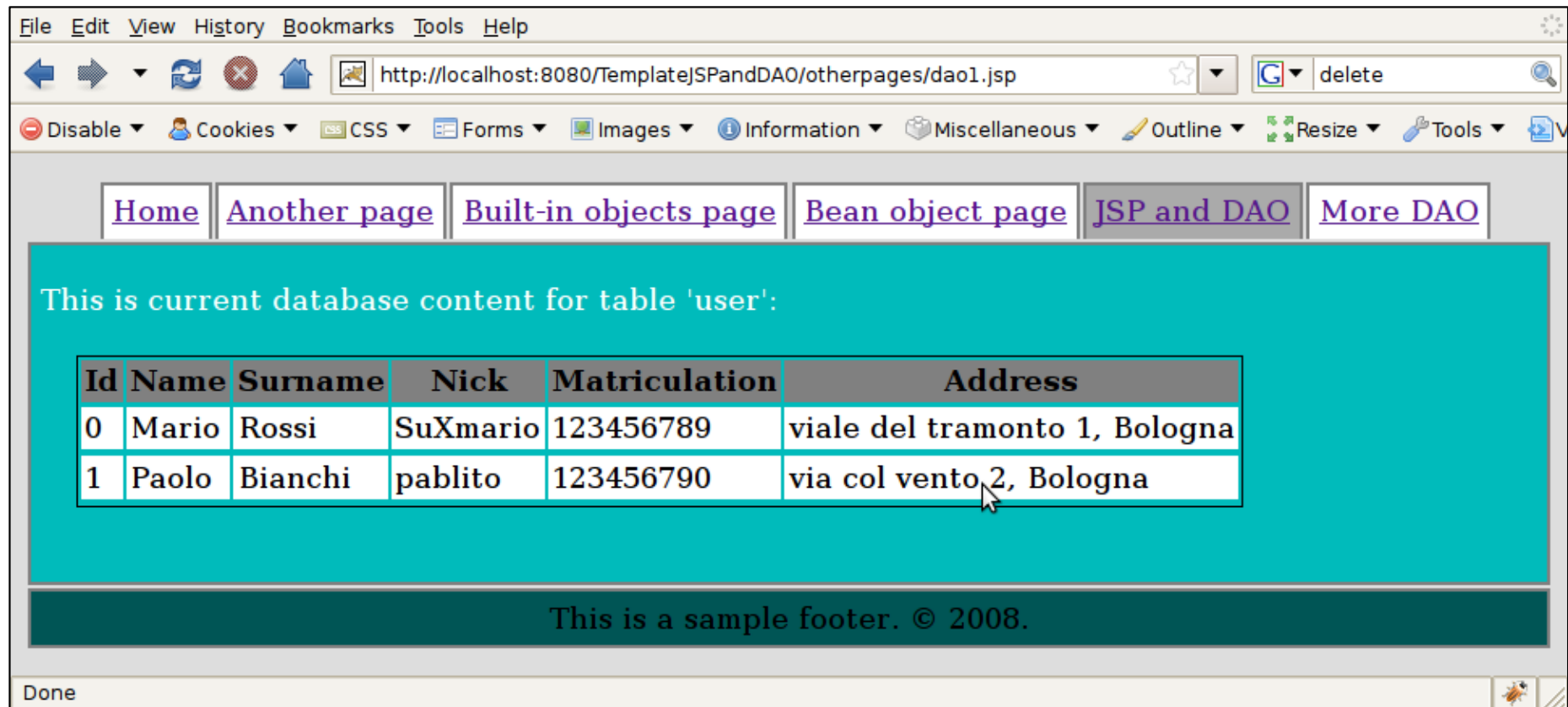
*application*

- Provare a navigare seguendo i link per comprendere il funzionamento
- La pagina *bean3.jsp* mostra un po' di EL



## [dao1.jsp](#)

- Una pagina JSP che legge da database un insieme di valori e li stampa a video in una *table* all'interno dell'HTML finale che produce



The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/TemplateJSPandDAO/otherpages/dao1.jsp`. The browser's menu bar includes File, Edit, View, History, Bookmarks, Tools, and Help. The address bar contains navigation buttons and a search icon. Below the address bar, there are various toolbars for disabling content, cookies, CSS, forms, images, information, miscellaneous, outline, resize, and tools. The main content area features a navigation menu with links: Home, Another page, Built-in objects page, Bean object page, JSP and DAO, and More DAO. The main content is on a teal background and includes the text "This is current database content for table 'user':" followed by a table with the following data:

<b>Id</b>	<b>Name</b>	<b>Surname</b>	<b>Nick</b>	<b>Matriculation</b>	<b>Address</b>
0	Mario	Rossi	SuXmario	123456789	viale del tramonto 1, Bologna
1	Paolo	Bianchi	pablito	123456790	via col vento 2, Bologna

Below the table, there is a dark green footer area with the text "This is a sample footer. © 2008." The browser's status bar at the bottom shows "Done" and a small icon.

- Avete lanciato il database? E' un programma a parte!
  - *...nuovo target di ANT*

## Pattern DAO: alla conquista del database

---

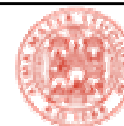
- Un “pattern” è un “modo di fare le cose” in modo efficace
- Il pattern “DAO” rappresenta “IL” modo di separare tra loro:
  - logica di business (es: Servlet, pagine JSP, ...)
  - logica di persistenza (es: scritture su DB, letture, ...)
- I componenti della logica di business NON DOVREBBERO MAI CONTENERE CODICE CHE ACCEDE DIRETTAMENTE AL DATABASE
  - scarsa manutenibilità
  - sovrapposizione di responsabilità
- Solo gli oggetti “DAO” hanno il permesso di “vedere” il DB
  - espongono metodi di accesso adatti per tutti gli altri componenti



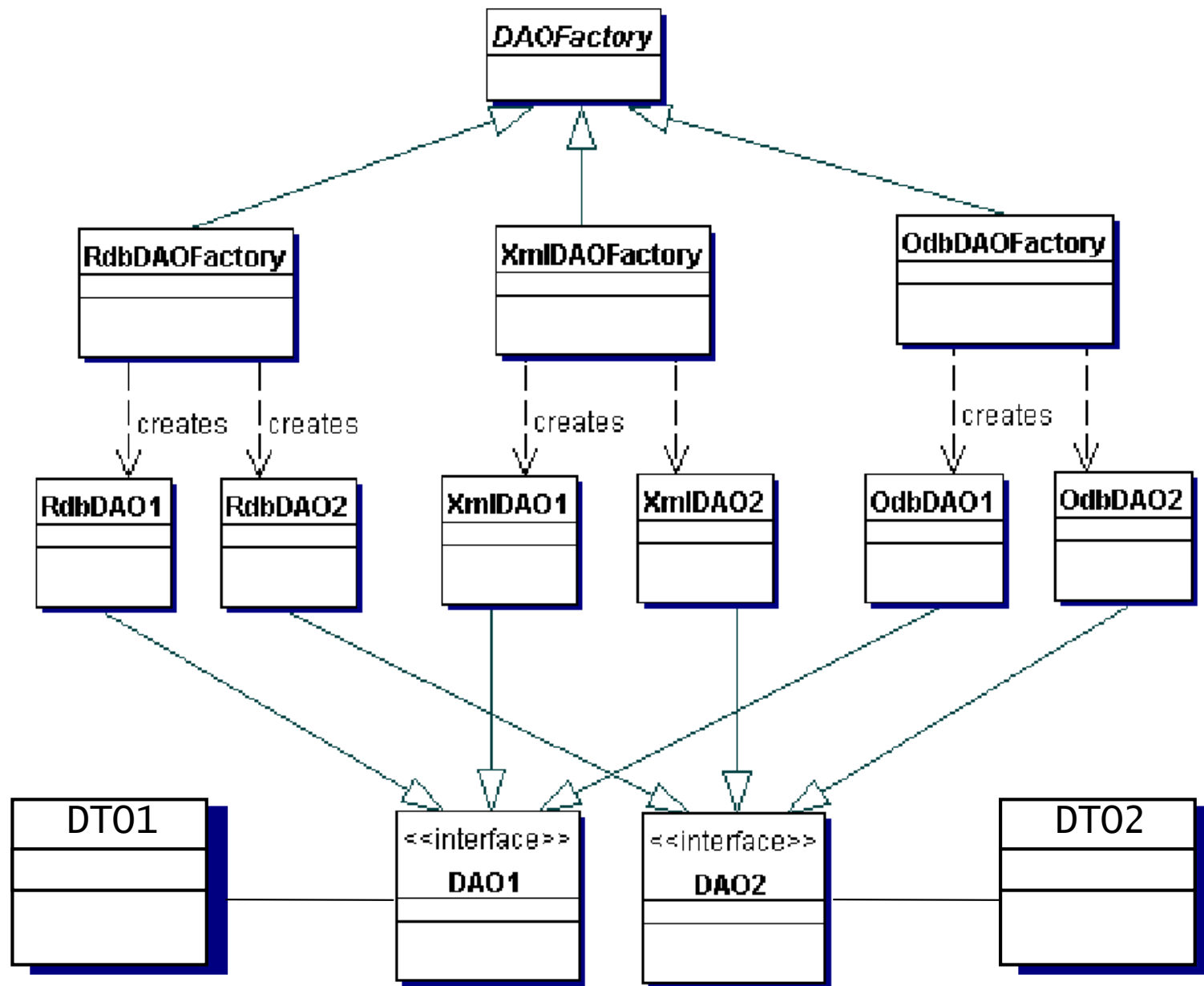
## Pattern DAO: principi fondamentali

---

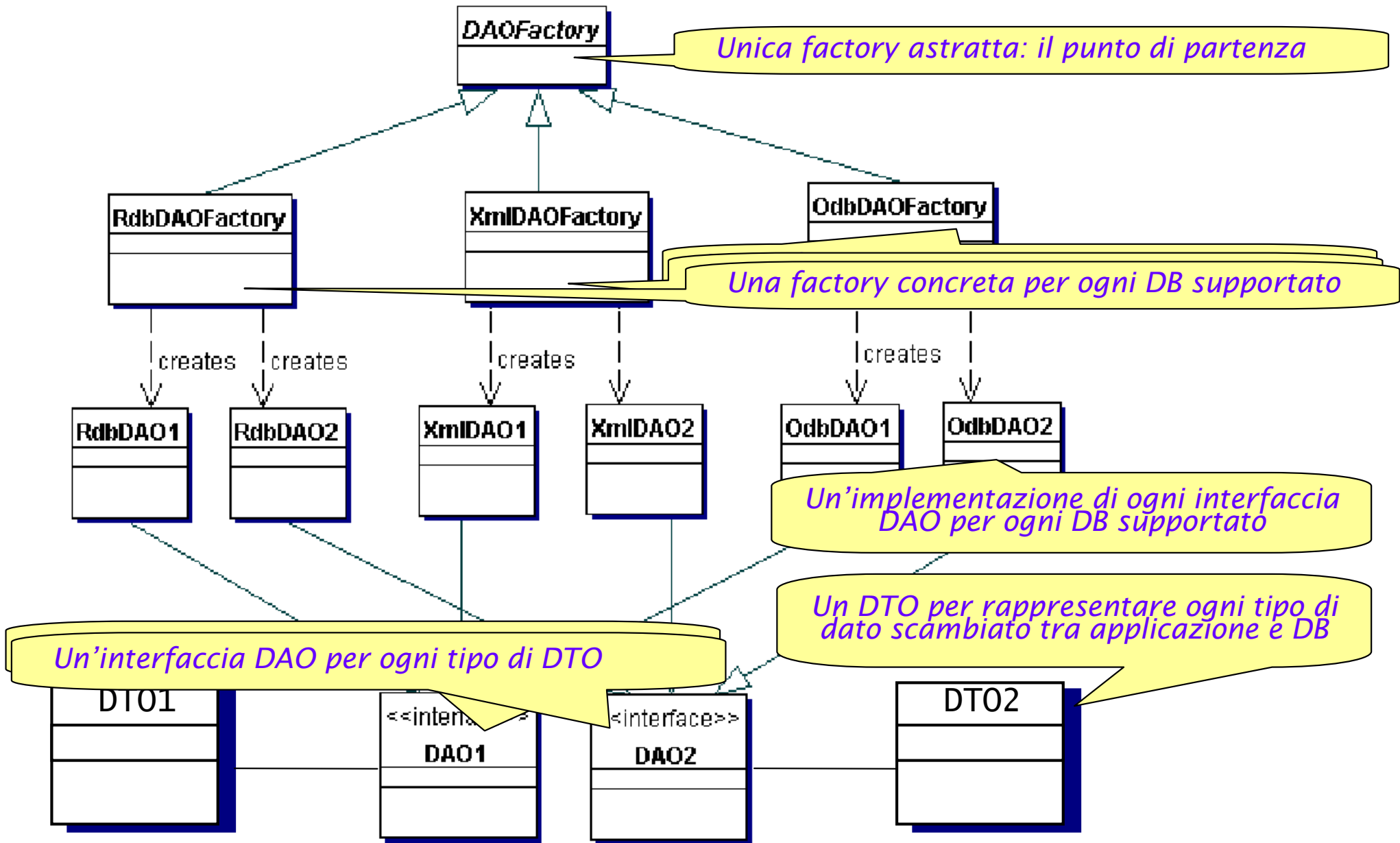
- I valori scambiati tra DB e resto dell'applicazione (attraverso gli oggetti "DAO") sono racchiusi in oggetti detti **Data Transfer Object (DTO)**:
  - campi privati per contenere i dati da leggere/scrivere su db
  - metodi getter e setter per accedere dall'esterno a tali campi
  - metodi di utilità (confronto, stampa, calcolo dell'hashcode, ...)
- Le operazioni che coinvolgono tali oggetti sono raggruppati in interfacce che definiscono i **Data Access Object (DAO)** disponibili
  - metodi *Create, Read, Update, Delete (CRUD)*
  - altri metodi (tipicamente di lettura con parametri custom)
- **Diverse implementazioni** di tali interfacce permettono l'accesso a diversi database
  - anche se si fa uso di un solo database, tale separazione migliora comunque la divisione delle responsabilità tra le parti dell'applicazione
  - diventa facile migrare l'applicazione su DB diversi un domani)
- Le implementazioni degli oggetti DAO **non sono istanziate direttamente** dai componenti (facendo `new Q1cDAO()` ~~per capirci~~), ma:
  - sono ottenute attraverso metodi *factory (come faremo noi)*
  - sono settate nei componenti da qualche altra entità (es: il container)



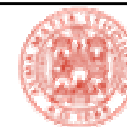
# The DAO pattern: UML schema



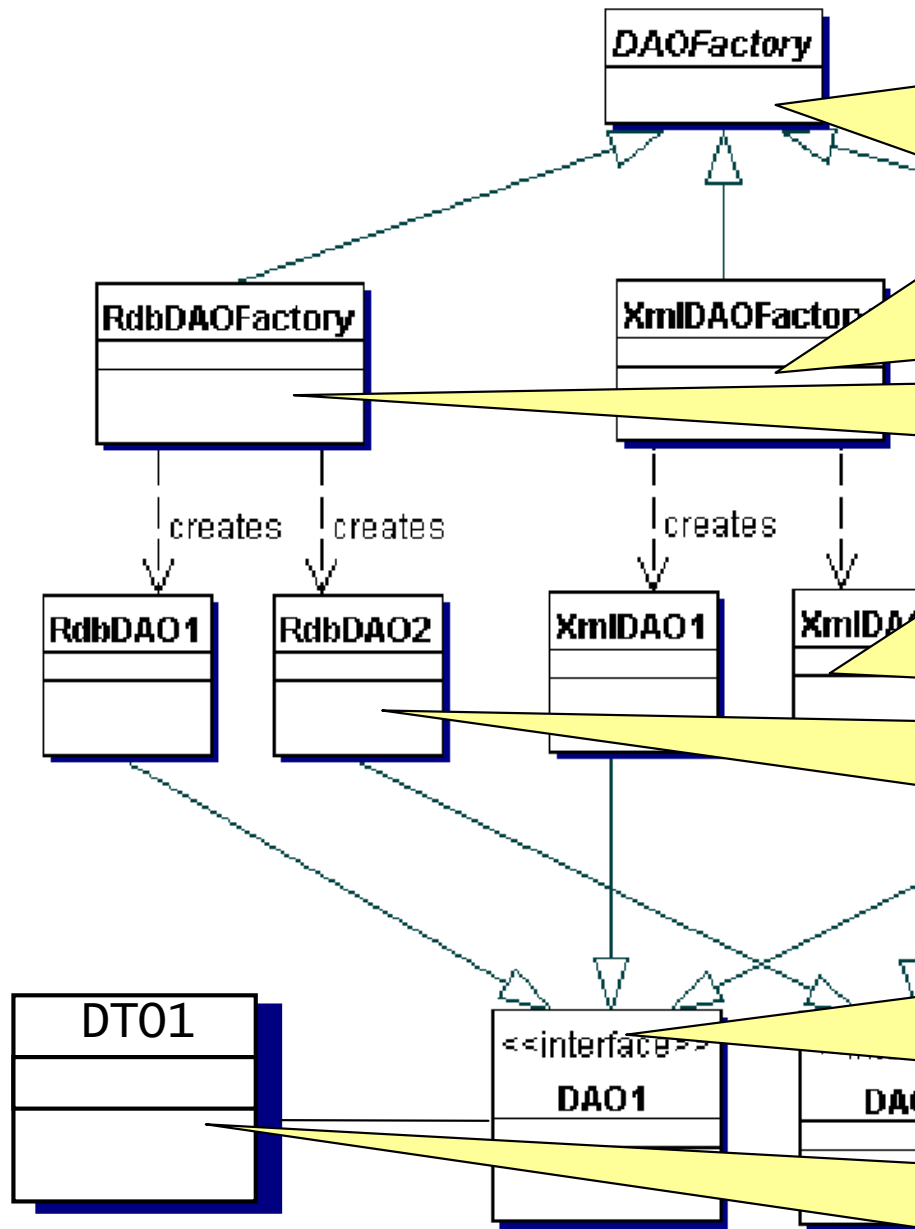
# The DAO pattern: UML schema



In genere si ha corrispondenza 1:1 tra i DTO gestiti dalle interfacce DAO e le tabelle su DB, ma non è strettamente necessaria!



# The DAO pattern: UML schema



*it.unibo.tw0708.dao.DAOFactory*

- astratta
- fornisce specifiche per le factory concrete
- espone un metodo creazionale per selezionare la factory concreta da usare

*it....dao.hsqldb.HsqldbDAOFactory*  
*it....dao.mysql.MySqlDAOFactory*

- implementano le specifiche fornendo metodi per istanziare gli oggetti DAO relativi al proprio tipo di database
- possono gestire connessioni, credenziali, autenticazione, ecc..

*it....dao.hsqldb.HsqldbUserDAO*  
*it....dao.mysql.MySqlUserDAO*

- oggetti DAO concreti contenenti il codice per la lettura e scrittura su database

*it.unibo.tw0708.dao.UserDAO*

- interfacce che definiscono quali metodi sono disponibili per interagire con il DB

*it.unibo.tw0708.dao.UserTO*

- oggetti che racchiudono i valori letti o da scrivere su DB

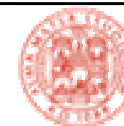
# JDBC

- Resta ancora da capire come scrivere il codice di questi DAO
- La risposta (per noi) è JDBC
  - Java fornisce una API standard per formulare statement SQL verso database relazionali
    - *SELECT \* FROM .... WHERE ....., UPDATE .... SET ... WHERE ....., ecc...*
    - Semplicemente si passano tali stringhe a opportuni metodi che le cui implementazioni sono in grado di sottoporle al database
      - query parametriche
      - lettura dei risultati
      - gestione delle connessioni
  - ogni database fornisce un “driver” che implementa tale API, sotto forma di una libreria *.jar* da includere nel *path* dell’applicazione
    - *hsqldb.jar* per HSQLDB
    - *mysql-connector-xxx.jar* per *MySql*
    - ecc...
  - Gli statement SQL tuttavia non sono sempre “portabili”: DB differenti parlano diversi dialetti
    - in realtà non su tutto... molte query sono standard, ma le ottimizzazioni no!
    - nel progetto di oggi già c’è un esempio di statement “dialettale”
    - ...ed ecco perché il pattern DAO

*Uno sguardo al codice vale più di mille parole!*

*Nota: nel caso di HSQLDB, il jar contiene sia il driver JDBC che il programma database!*

- *il driver serve nella webapp, quindi il file sta in WEB-INF/lib*
- *il db lo lanciamo da ANT, quindi il jar fa parte del classpath definito in build.xml*
- *infine, usiamo il driver anche attraverso 2 classi con main() lanciate via Eclipse: il jar è anche nel buildpath*





## JDBC: regole auree

(@see *HsqldbUserDAO.java*)

- Il consiglio è mio personale, ma questo modo di procedere nello scrivere il codice degli oggetti DAO basati su JDBC è... a prova di bomba!
  - Ogni metodo...
    1. *dichiari una variabile dove collocare il proprio risultato*
    2. *controlli la bontà dei parametri attuali ricevuti*
    3. *apra la connessione al DB*
    4. *formuli gli statement SQL che lo riguardano e imposti il risultato*
    5. *preveda di gestire le eventuali eccezioni*
    6. *rilasci SEMPRE E IN OGNI CASO la connessione in uso*
    7. *restituisca il risultato (eventualmente di fallimento)*
  - E per quanto riguarda gli **statement SQL** veri e propri
    1. *crei (se senza parametri) o prepari (se con parametri) lo statement da proporre al DB*
    2. *pulisca e imposti i parametri (se ve ne sono, ovviamente)*
    3. *esegua l'azione sul DB ed estrapola il risultato (se atteso)*
    4. *cicli sul risultato (se presente) per accedere a ogni sua tupla e impostare il proprio risultato con i valori in essa contenuti*
    5. *rilasci la struttura dati del risultato stesso*
    6. *rilasci la struttura dati dello statement*

*Anche in questo caso, uno sguardo al codice  
- COMMENTATISSIMO! - vale più di mille parole!*



## dao1.jsp

---

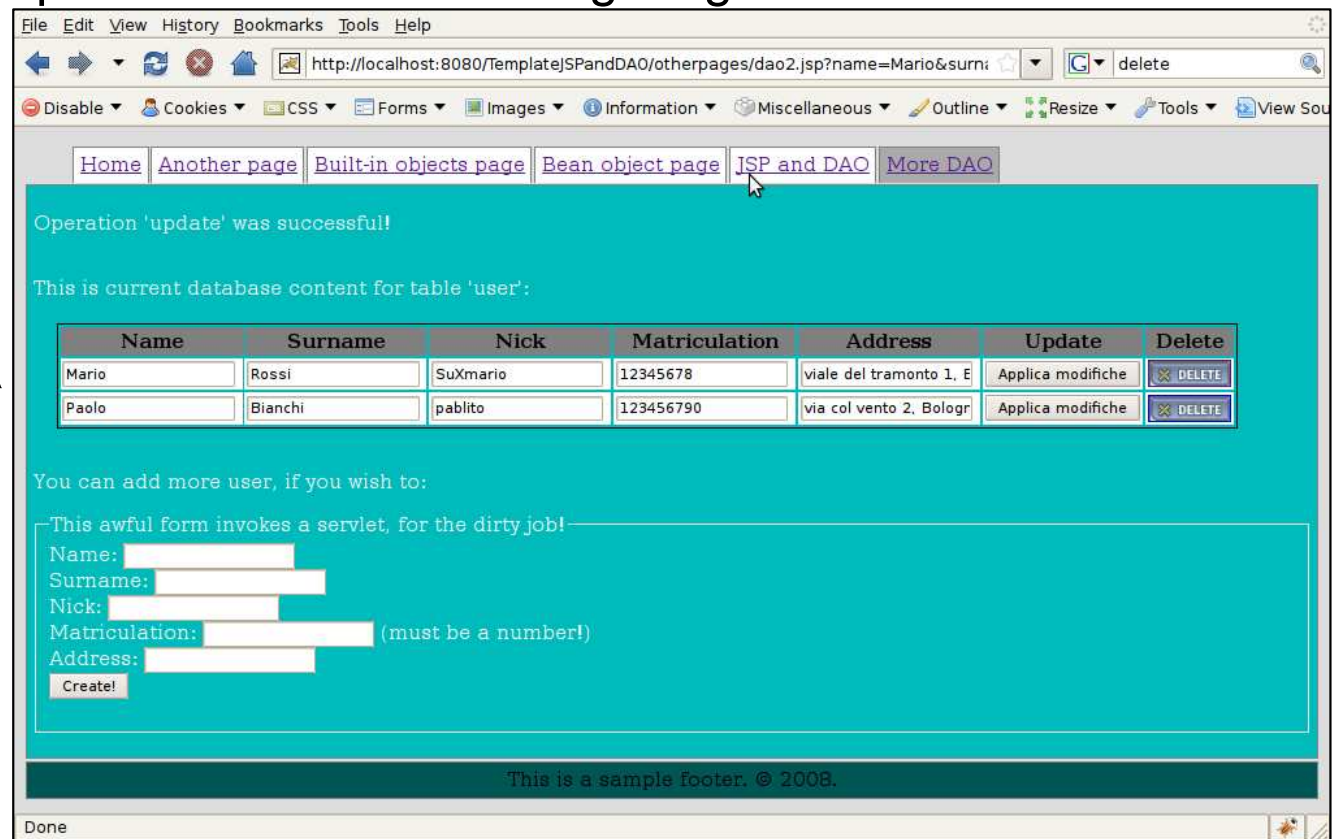
- Dichiarazioni:
  - metodi di utilità per la scrittura dell'HTML finale
- Scriptlet:
  - lettura del tipo di factory DAO da utilizzare a partire dal ServletContext
    - oggetto builtin di nome application,
    - settato qui in maniera dichiarativa via *web.xml*
  - istanziamento di un oggetto DAO per accedere al DB degli utenti
  - invocazione di un metodo di lettura
  - stampa a video del risultato



## dao2.jsp

- Anziché scrivere del testo nelle celle della tabella...
  - una form per ogni riga
  - un input field per ogni cella
  - un campo nascosto per veicolare l'id
  - un campo di tipo submit per lanciare la form di ogni riga

■ ...e inoltre, per ogni riga, un anchor che cabla nell' URL del proprio *href* l'id della riga in cui si trova e ordina un'operazione diversa rispetto al pulsante della form



Operation 'update' was successful!

This is current database content for table 'user':

Name	Surname	Nick	Matriculation	Address	Update	Delete
Mario	Rossi	SuXmario	12345678	viale del tramonto 1, E	Applica modifiche	DELETE
Paolo	Bianchi	pablito	123456790	via col vento 2, Bologr	Applica modifiche	DELETE

You can add more user, if you wish to:

This awful form invokes a servlet, for the dirty job!

Name:

Surname:

Nick:

Matriculation:  (must be a number!)

Address:

Create!

This is a sample footer. © 2008.

## Non solo pagine JSP

---

- Il pattern DAO non è limitato all'uso da parte di pagine JSP
  - disponibile anche per le Servlet
  - disponibile persino da Eclipse (due classi da lanciare col tasto destro del mouse, rispettivamente per inizializzare e ripulire le tabelle su DB)
  - ecc...
  
- La pagina *dao2.jsp* contiene anche una form la cui *action* chiama in causa la servlet *UpdateServlet.java*
  - lettura del tipo di factory DAO da usare dal *ServletContext*
  - lettura dei parametri della richiesta
  - esecuzione della logica desiderata
  - ....anziché produrre un HTML in risposta, attraverso il *RequestDispatcher* viene 'forwardata' la gestione della richiesta nuovamente alla pagina *dao2.jsp*