Lezioni di Programmazione Web A.A. 2012-2013

Docente: Novella Bartolini
Ricevimento: Giovedì 10:00-12:00
Via Salaria 113, terzo piano, stanza 309
Email: bartolini@di.uniroma1.it

Informazioni sul corso

- Tutto sul sito del corso

  http://twiki.dsi.uniroma1.it/twiki/view/Lab_prog_rete/WebHome

  (per ora!)

- Orario lezioni
- Orario di ricevimento
- Testi di riferimento
- Slide delle lezioni
- Prenotazione esami
- Comunicazioni varie

Orario lezioni e ricevimento

- Lezione: Lunedì e Martedì dalle 13.30 alle 15.00
  - Eventuali variazioni verranno segnalate sul sito
- Ricevimento: Giovedì dalle 10.00 alle 12.00
- **Organizzazione delle lezioni *(perlopiù…)*:**
  - **Lunedì: teoria**
  - **Martedì: esercitazioni pratiche**

Testi di riferimento

- Dispensa su XHTML (Deitel & Deitel) , Prentice Hall and Deitel & Associates
- Dispensa su CSS (Deitel & Deitel), Prentice Hall and Deitel & Associates
- Marty Hall, "Core Servlets and Java Server Pages“, Prentice Hall & Sun Microsystems

Modalità di esame per tutti

- Per sostenere l'esame è richiesta la conoscenza del linguaggio Java (livello base)
- L'esame finale consiste in due prove: una prova pratica di realizzazione di un'applicazione web, e una prova di teoria.
- Il voto finale sarà la media dei voti conseguiti nella prova pratica e in quella teorica.

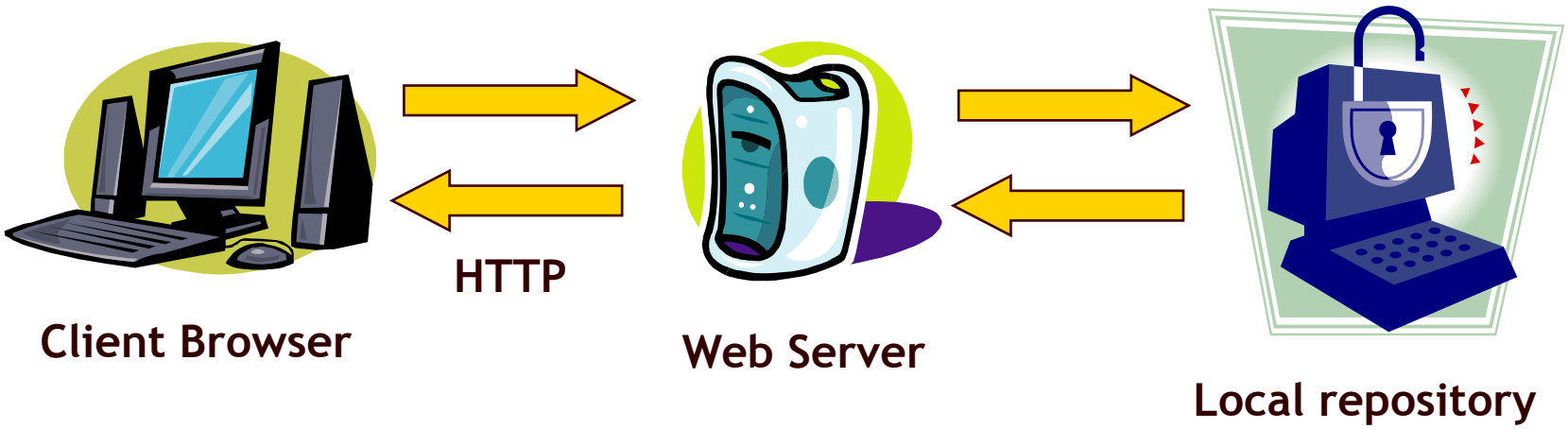
WIS

- Un Web-based Information System è un sistema informatico basato sul web.
- Non è un insieme di pagine web
- Ha un'elevata complessità sia in termini di dati che di applicazioni
- E' spesso integrato con sistemi diversi come DBMS, sistemi transazionali ecc.

Tipologie di servizi elettronici

- Informativi
  - Per la fornitura su richiesta di informazioni strutturate e classificate
- Di comunicazione
  - Interazione bidirezionale per interagire con altri individui o gruppi di individui
- Transazionali
  - Per acquistare prodotti o servizi online, o per trasmettere dati

Architetture realizzative di un WIS



Web Browser

Software per reperire pagine web, attraverso il protocollo HTTP e visualizzarle

- Invia richieste e riceve risposte secondo il protocollo HTTP
- Interpreta comandi di formattazione espressi in HTML
- Visualizza file di tipo diverso (espresso attraverso un'*estensione MIME*)

MIME

- Multipurpose Internet Mail Extensions
- Standard che specifica tipi di oggetto non testuali per la trasmissione in applicazioni Internet (WWW, e-mail)
- Attraverso l'estensione MIME è possibile associare un oggetto ad un'applicazione che lo gestisca
- MIME specifica solo il formato degli oggetti, che vengono trasmessi secondo una codifica standard (base64)
- standard che prevede tutte le possibili funzionalità per la trasmissione dei documenti; non è detto che un applicativo sia in grado di realizzarle tutte

Web Server

- Programma in grado di fornire pagine web (memorizzate sulla macchina su cui viene eseguito) attraverso il protocollo HTTP
- Processo demone con socket in ascolto sulla porta TCP 80
- I più diffusi:
  - Apache
  - Internet Information Server (IIS)

World Wide Web Consortium W3C

- W3C
  - Fondato nel 1994 da Tim Berners-Lee
    - per lo sviluppo e l'integrazione di tecnologie per il World Wide Web.
  - Ente di standardizzazione
    - W3C *Recommendations*: documenti che specificano il funzionamento delle tecnologie per il WWW (es: Extensible HyperText Markup Language - XHTML, Cascading Style Sheets – CSS, e Extensible Markup Language - XML)

XHTML

- Extensible HyperText Markup Language
  - Evoluzione dell'HTML
- Documenti XHTML
  - Costituiscono la codifica sorgente delle pagine web
  - Editabili attraverso un comune editor testuale
  - Estensione del nome dei file `.html` o `.htm`
- Web server
  - Memorizza i documenti XHTML
- Web browser
  - Effettua richieste di documenti XHTML

Testo di riferimento su XHTML

- Presente sul sito

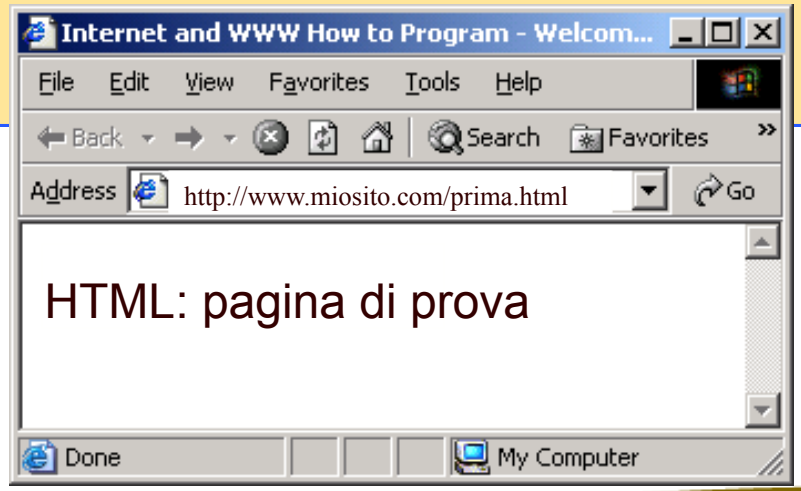
  http://twiki.dsi.uniroma1.it/twiki/view/Lab_prog_rete/WebHome

  Introduction to XHTML, Deitel & Deitel

Struttura di un documento XHTML

- La struttura del documento viene definita attraverso
  - Start tag (es. <html>)
    - Definiscono attributi attraverso espressioni *name* = *value*
  - End tag (es. </html>)
- Intestazione (head)
  - Definisce il titolo del documento
  - Include fogli di stile e di scripting
- Corpo centrale (body)
  - Definisce il contenuto che viene visualizzato dal browser
- Commenti XHTML
  - Iniziano con <!-- e terminano con -->

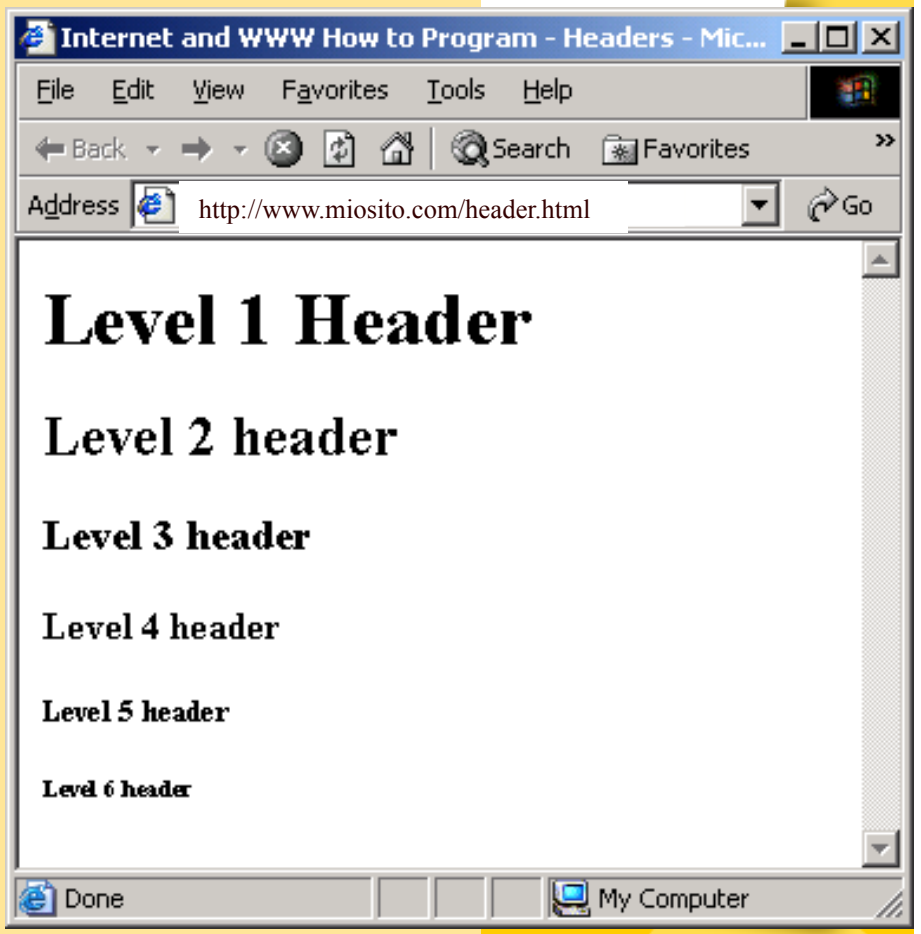
```
1  <html>
2    <head>
3      <title>Internet and WWW How to Program - Welcome</title>
4    </head>
5
6    <body>
7      <p>HTML: pagina di prova  </p>
8    </body>
9  </html>
```



Elementi del linguaggio html

- Sei tipi di header
  - da `h1` a `h6`

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5  <!-- header.html -->
6  <!-- XHTML headers-->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Internet and WWW How to Program - Headers</title>
11    </head>
12
13    <body>
14
15       <h1>Level 1 Header</h1>
16       <h2>Level 2  header</h2>
17       <h3>Level 3  header</h3>
18       <h4>Level 4  header</h4>
19       <h5>Level 5  header</h5>
20       <h6>Level 6  header</h6>
21
22    </body>
23 </html>
```

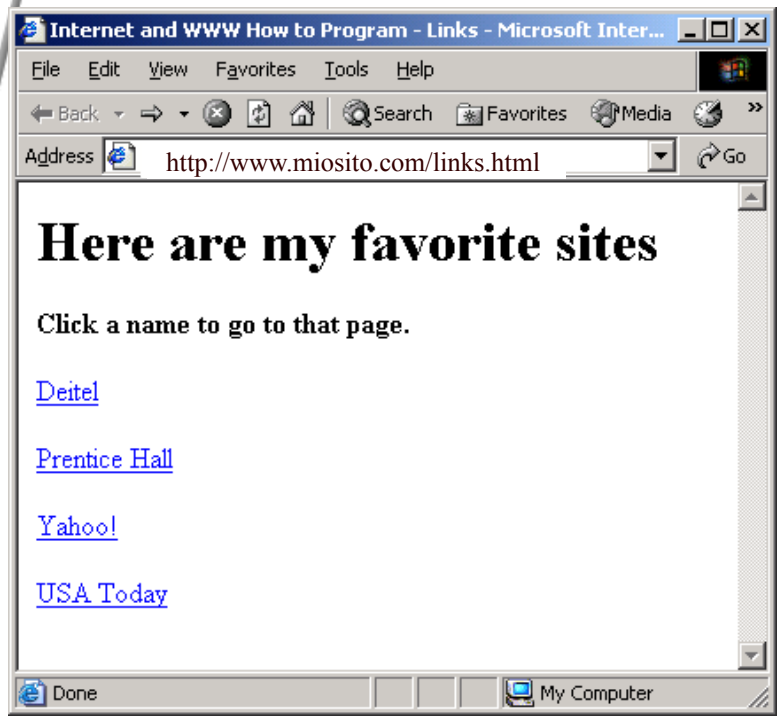


Hyperlinks

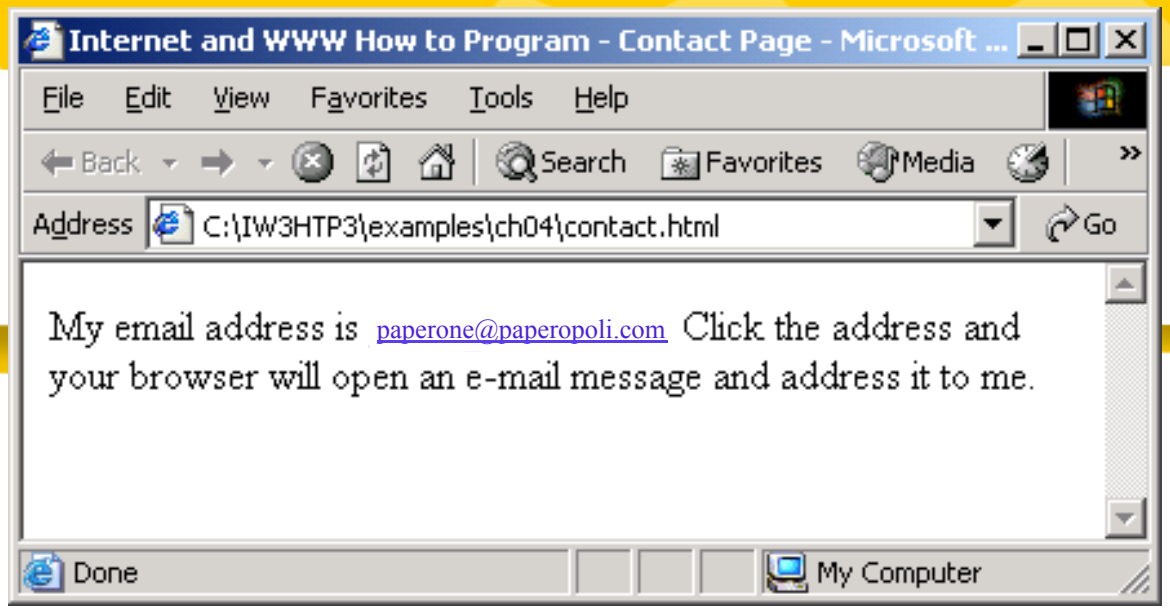
- Riferimenti ad altri file come documenti XHTML e immagini
- Sia il testo che le immagini possono costituire un hyperlink
- Vengono creati usando il tag **`<a>`** (anchor)
  - Attributo **`href`**
    - Specifica il percorso dell'oggetto del link
  - Il link può essere un indirizzo di posta elettronica, usando **`mailto:`** URL

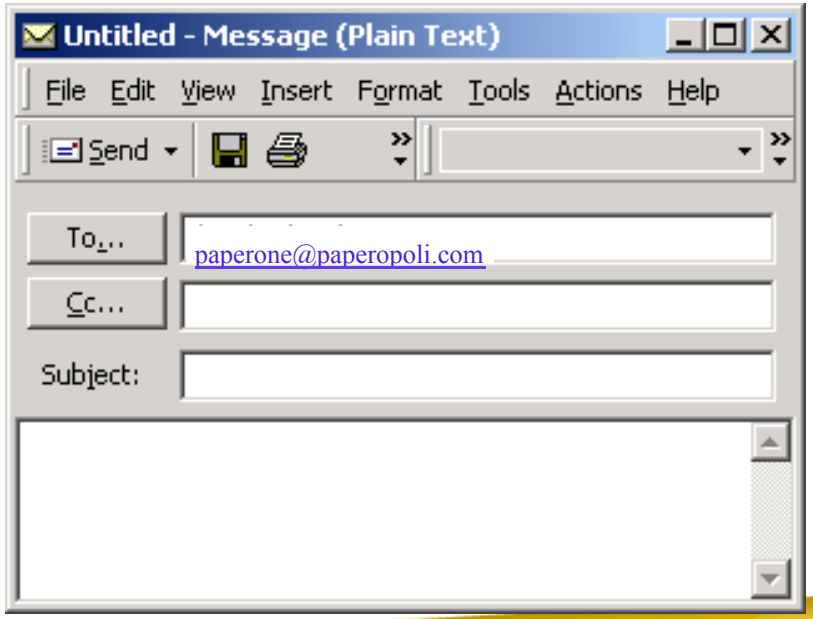
```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5  <!-- links.html -->
6  <!-- Introduction to hyperlinks -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title> Internet and WWW How to Program - Links </title>
11    </head>
12
13    <body>
14
15       <h1>Here are my favorite sites </h1>
16
17       <p><strong>Click a name to go to that page. </strong></p>
18
19       <!-- Create four text hyperlinks -->
20       <p><a href = "http://www.deitel.com">Deitel </a></p>
21
22       <p><a href = "http://www.prenhall.com">Prentice Hall </a></p>
23
24       <p><a href = "http://www.yahoo.com">Yahoo!</a></p>
25
```

```
26          <p><a href =    "http://www.usatoday.com"   >USA Today</a></p>
27
28       </body>
29    </html>
```



```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5  <!-- contact.html -->
6  <!-- Adding email hyperlinks -->
7
8  <html xmlns = "http://wwww3.org/1999/xhtml">
9     <head>
10       <title> Internet and WWW How to Program- Contact Page</title>
11    </head>
12
13    <body>
14
15       <p>
16          My email address is
17          <a href = "mailto: paperone@paperopoli .com">
18             paperone@paperopoli.com
19          </a>
20          Click the address and your browser will
21        open an e -mail message and address it to me.
22       </p>
23    </body>
24 </html>
```


Internet and WWW How to Program - Contact Page - Microsoft ...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\IW3HTP3\examples\ch04\contact.html
Go
My email address is paperone@paperopoli.com Click the address and your browser will open an e-mail message and address it to me.
Done
My Computer

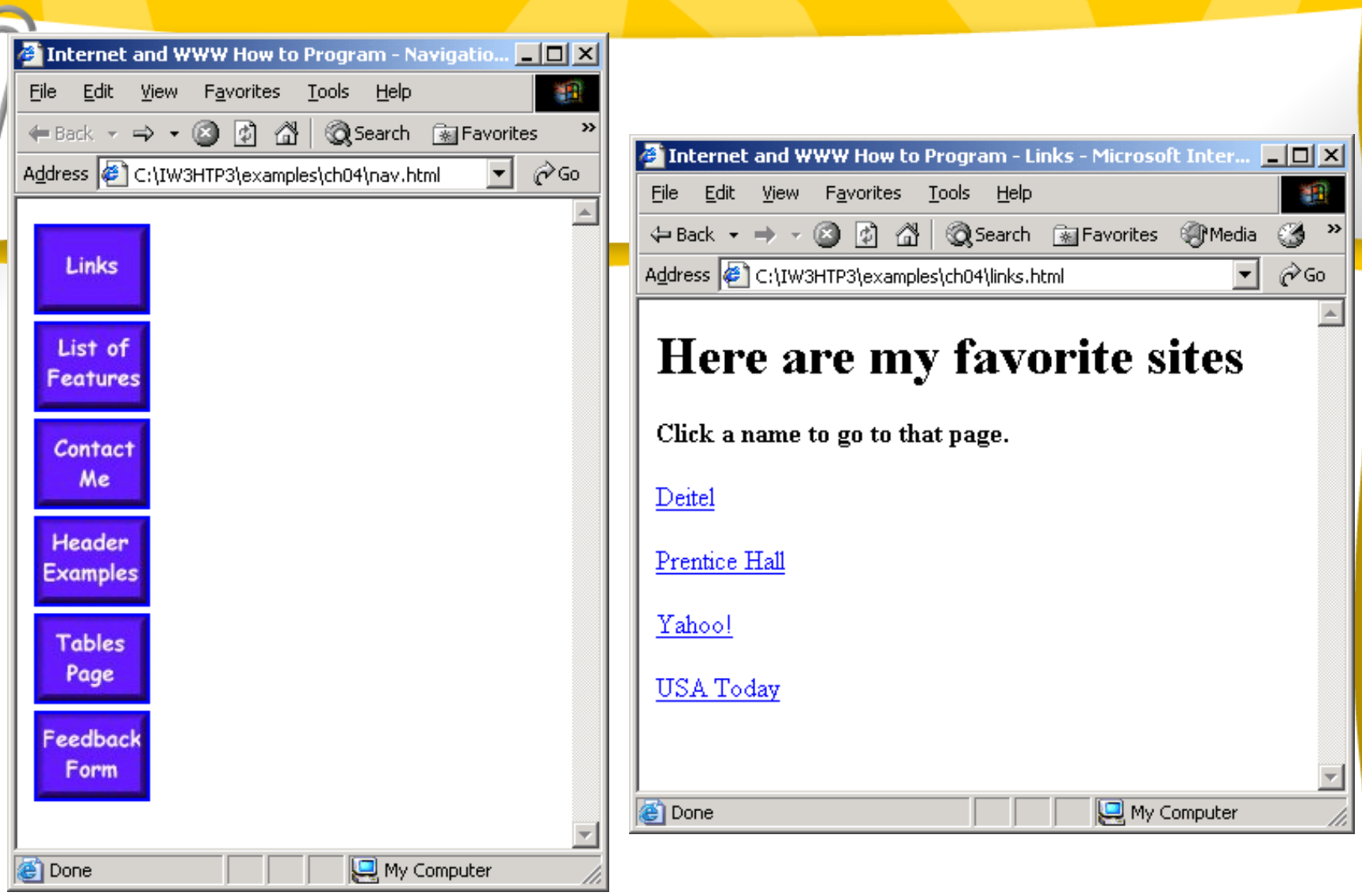

Untitled - Message (Plain Text)
File Edit View Insert Format Tools Actions Help
Send
To...
paperone@paperopoli.com
Cc...
Subject:

Immagini

- Formati più comuni sul web: gif, jpg, png
- Si includono usando l'elemento `img`
  - Attributo `src`
    - Specifica la locazione del file contenente l'immagine
  - Attributi `width` and `height`

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5  <!-- nav.html -->
6  <!-- Using images as link anchors -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title> Internet and WWW How to Program- Navigation Bar
11       </title>
12    </head>
13
14    <body>
15
16       <p>
17          <a href = "links.html" >
18             <img src = "buttons/links.jpg" width = "65"
19                height = "50" alt = "Links Page" />
20          </a><br />
21
22          <a href = "list.html" >
23             <img src = "buttons/list.jpg" width = "65"
24                height = "50" alt = "List Example Page" />
25          </a><br />
```

```
26
27        <a href = "contact.html" >
28          <img src = "buttons/contact.jpg" width = "65"
29             height = "50" alt = "Contact Page" />
30        </a><br />
31
32        <a href = "header.html" >
33          <img src = "buttons/header.jpg" width = "65"
34             height = "50" alt = "Header Page" />
35        </a><br />
36
37        <a href = "table1.html" >
38          <img src = "buttons/table.jpg" width = "65"
39             height = "50" alt = "Table Page" />
40        </a><br />
41
42        <a href = "form.html" >
43          <img src = "buttons/form.jpg" width = "65"
44             height = "50" alt = "Feedback Form" />
45        </a><br />
46      </p>
47
48    </body>
49  </html>
```


Internet and WWW How to Program - Navigatio...
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\IW3HTP3\examples\ch04\nav.html
Go
Links
List of Features
Contact Me
Header Examples
Tables Page
Feedback Form
Done
My Computer
Internet and WWW How to Program - Links - Microsoft Inter...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\IW3HTP3\examples\ch04\links.html
Go
Here are my favorite sites
Click a name to go to that page.
Deitel
Prentice Hall
Yahoo!
USA Today
Done
My Computer

Tag di formattazione del testo

- `del`
  - Testo barrato
- `sup`
  - Testo formattato come apice
- `sub`
  - Testo formattato come pedice
- `br`
  - Interruzione di linea
- `hr`
  - Linea orizzontale

Tabelle XHTML

- Si usano per organizzare dati in righe e colonne
- Tag `table` per definire una tabella
  - Attributo **border**
    - » Specifica la larghezza del bordo della tabella in pixel
  - Elemento **caption**
    - » Introduce una didascalia
  - L'elemento `tr` definisce una riga della tabella
  - Le celle di dati sono definite con il tag `td`

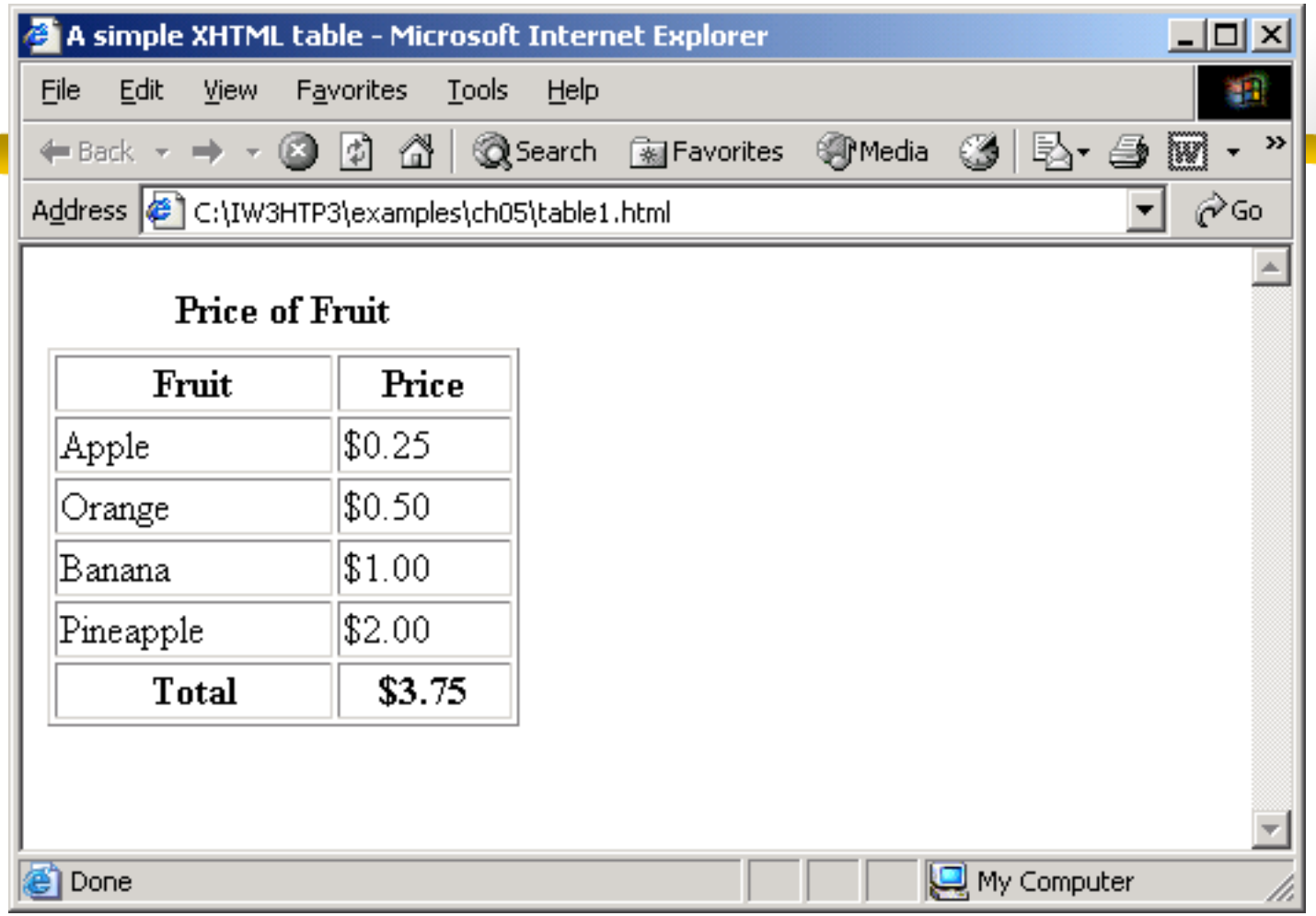
Tabelle XHTML

- Intestazione della tabella definita attraverso il tag **`thead`**, sezione conclusiva definita con l'elemento **`tfoot`**
  - L'elemento **`th`** (definisce le colonne della sezione di intestazione e della sezione conclusiva)
- Il corpo della tabella è definito con l'elemento **`tbody`**

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5  <!-- table1.html  -->
6  <!-- Creating a basic table  -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9    <head>
10     <title>A simple XHTML table</title>
11   </head>
12
13   <body>
14
15     <!-- the <table> tag opens a table -->
16     <table border = "1" width = "40%">
17
18
19
20       <!-- the <caption> tag summarizes the table's   -->
21       <!-- contents (this helps the visually impaired) -->
22       <caption><strong>Price of Fruit</strong></caption>
23
```

```
24      <!    -- the <thead> is the first section of a table          -- >
25      <!    -- it formats the table header area                     -- >
26          <thead>
27              <tr>                  <!-- <tr> inserts a table row      -- >
28                  <th>Fruit </th>  <!-- insert a heading cell       -- >
29                  <th>Price </th>
30              </tr>
31          </thead>
32
33      <!    -- the <tfoot> is the last section of a table           -- >
34      <!    -- it  formats the table footer                         -- >
35          <tfoot>
36              <tr>
37                  <th>Total </th>
38                  <th>$3.75</th>
39              </tr>
40          </tfoot>
41
42      <!    -- all table content is enclosed             -- >
43      <!    -- within the <tbody>                        -- >
44          <tbody>
45              <tr>
46                  <td>Apple </td>  <!-- insert a data cell       -- >
47                  <td>$0.25</td>
48              </tr>
```

```
49
50          <tr>
51                  <td> Orange</td>
52                  <td>$0.50</td>
53          </tr>
54
55          <tr>
56                  <td>Banana</td>
57                  <td>$1.00</td>
58          </tr>
59
60          <tr>
61                  <td>Pineapple  </td>
62                  <td>$2.00</td>
63          </tr>
64            </tbody>
65
66        </table>
67
68    </body>
69 </html>
```


A simple XHTML table - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\IW3HTP3\examples\ch05\table1.html
Go
Price of Fruit
Fruit
Price
Apple
$0.25
Orange
$0.50
Banana
$1.00
Pineapple
$2.00
Total
$3.75
Done
My Computer

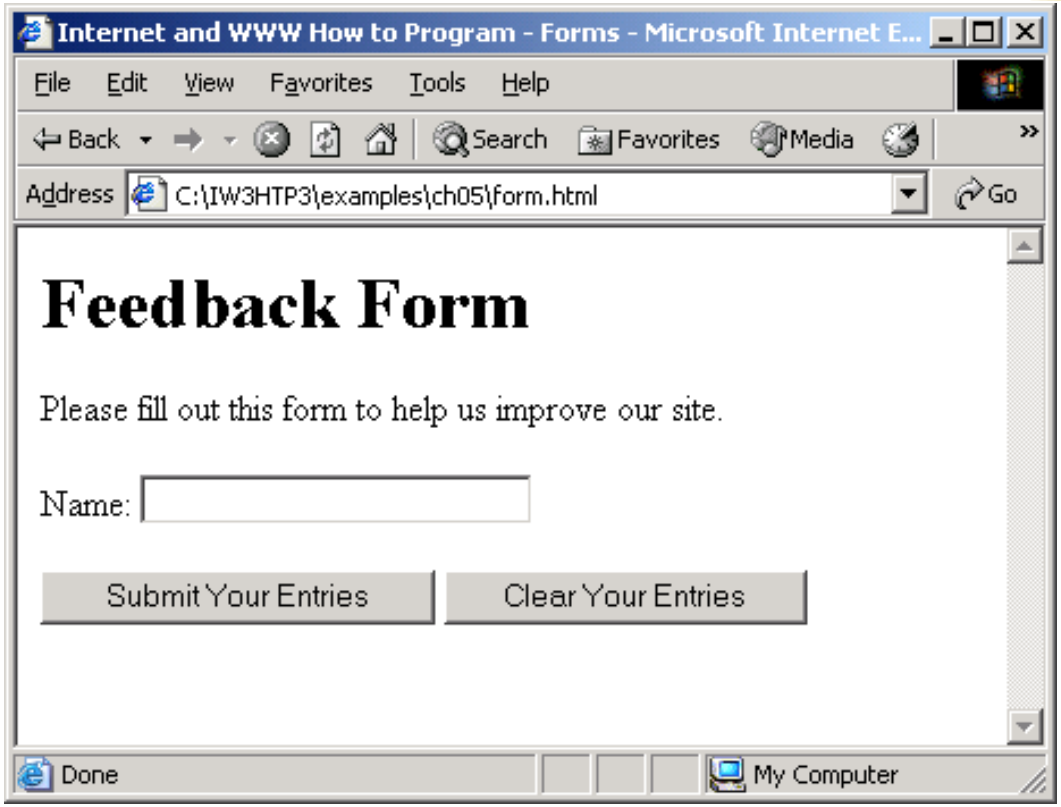
Form XHTML

- Elemento `form`
  - Attributo `method`
    - Specifica come inviare i dati del form al Web Server
    - `method = "post"`
      - I dati del form vengono inclusi nel messaggio di richiesta
    - `method = "get"`
      - I dati del form vengono appesi alla fine dell'URL
  - Attributo `action`
    - Specifica l'indirizzo di destinazione della richiesta, ovvero uno script sul Web server
- Elemento `input`
    - Specifica i dati da fornire allo script che elabora il form

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5
6  <!-- Form Design Example 1 -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Internet and WWW How to Program - Forms</title>
11    </head>
12
13    <body>
14
15       <h1>Feedback Form</h1>
16
17       <p>Please fill out this form to  help
18          us improve our site.</p>
19
20       <!-- this tag starts the form, gives the   -->
21       <!-- method of sending information and the -->
22       <!-- location of form scripts              -->
23       <form method = "post" action = "/cgi-bin/formmail">
24
```

```
25          <p>
26       <!      -- hidden inputs contain non      - visual    -- >
27       <!      -- information                                 -- >
28             <input type =    "hidden"   name =  "recipient"
29                value =   "deitel@deitel.com"      />
30          <input type =      "hidden"   name =  "subject"
31             value =     "Feedback Form"   />
32          <input type =      "hidden"   name =  "redirect"
33             value =     "main.html"    />
34        </p>
35
36          <!-- <input type = "text"> inserts a text box          -- >
37          <p><label> Name:
38                <input name =   "name" type =   "text"   size =   "25"
39                   maxlength =   "30"  />
40             </label></p>
41
42        <p>
43             <!-- input types "submit" and "reset" insert          -- >
44             <!-- buttons for submitting and clearing the          -- >
45             <!-- form's   contents                                -- >
46             <input type =    "submit"   value =
47                "Submit Your Entries"      />
48             <input type =    "reset"    value =
49                "Clear Your Entries"      />
50        </  p>
```

```
51
52          </form>
53
54       </body>
55    </html>
```



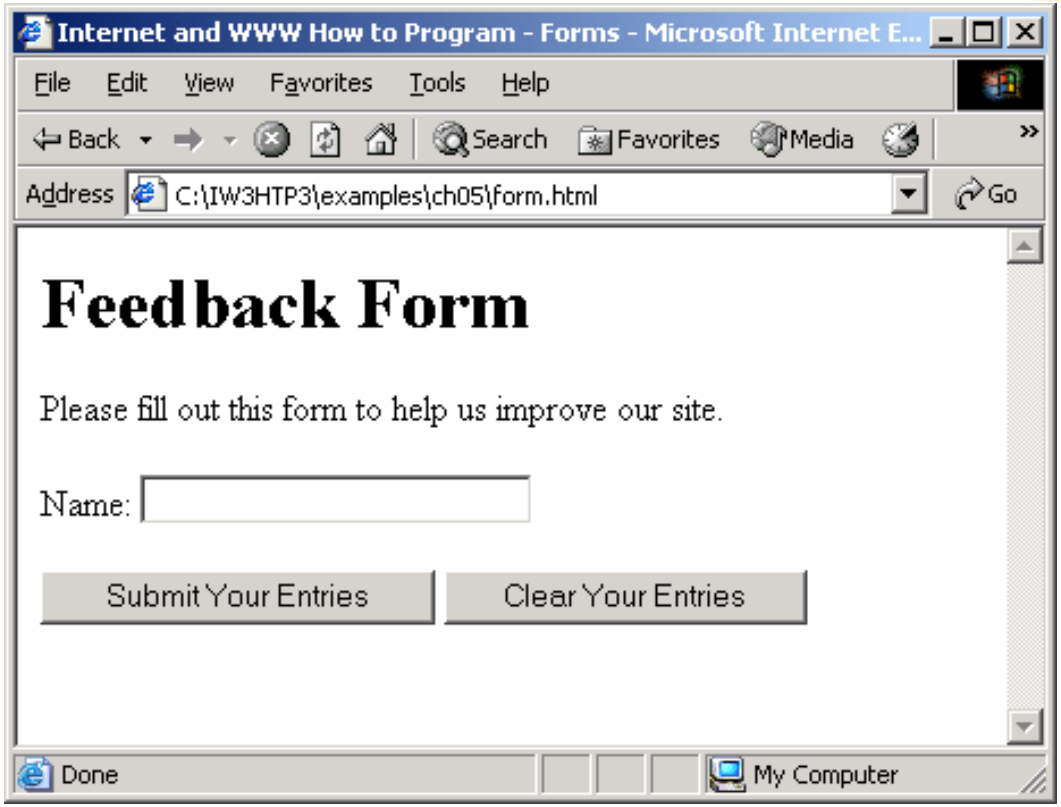
Form XHTML

- Elemento `form`
  - Attributo `method`
    - Specifica come inviare i dati del form al Web Server
    - `method = "post"`
      - I dati del form vengono inclusi nel messaggio di richiesta
    - `method = "get"`
      - I dati del form vengono appesi alla fine dell'URL
  - Attributo `action`
    - Specifica l'indirizzo di destinazione della richiesta, ovvero uno script sul Web server
- Elemento `input`
    - Specifica i dati da fornire allo script che elabora il form

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5
6  <!-- Form Design Example 1 -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Internet and WWW How to Program - Forms</title>
11    </head>
12
13    <body>
14
15       <h1>Feedback Form</h1>
16
17       <p>Please fill out this form to  help
18          us improve our site.</p>
19
20       <!-- this tag starts the form, gives the   -->
21       <!-- method of sending information and the -->
22       <!-- location of form scripts              -->
23       <form method = "post" action = "/cgi-bin/formmail">
24
```

```
25          <p>
26       <!      -- hidden inputs contain non      - visual    -- >
27       <!      -- information                                 -- >
28             <input type =    "hidden"   name =  "recipient"
29                value =   "deitel@deitel.com"      />
30          <input type =      "hidden"   name =  "subject"
31             value =     "Feedback Form"   />
32          <input type =      "hidden"   name =  "redirect"
33             value =     "main.html"    />
34        </p>
35
36          <!-- <input type = "text"> inserts a text box          -- >
37          <p><label> Name:
38                <input name =   "name" type =   "text"   size =   "25"
39                   maxlength =   "30"  />
40             </label></p>
41
42        <p>
43             <!-- input types "submit" and "reset" insert          -- >
44             <!-- buttons for submitting and clearing the          -- >
45             <!-- form's   contents                                -- >
46             <input type =    "submit"   value =
47                "Submit Your Entries"      />
48             <input type =    "reset"    value =
49                "Clear Your Entries"      />
50        </  p>
```

```
51
52          </form>
53
54       </body>
55    </html>
```



XHTML Form (continua)

- Elemento **textarea**
  - Inserisce un'area di testo
  - Attributi rows e cols
- Elemento input di tipo **password**
  - Inserisce un'area di testo che non viene visualizzata dal browser
- Elemento input di tipo **checkbox** (quadratino)
  - Abilita la selezione di un elenco di opzioni (nessuna, una o più di una)
- Elemento **select**
  - Fornisce una lista "drop-down" di elementi
  - Elemento **option**
    - Definisce gli elementi ad una lista drop-down
    - Attributo selected specifica quale item mostrare come selezionato
- Elemento input di tipo **radio** (cerchietto)
  - Permette di selezionare o deselezionare un'opzione (nessuna o una)

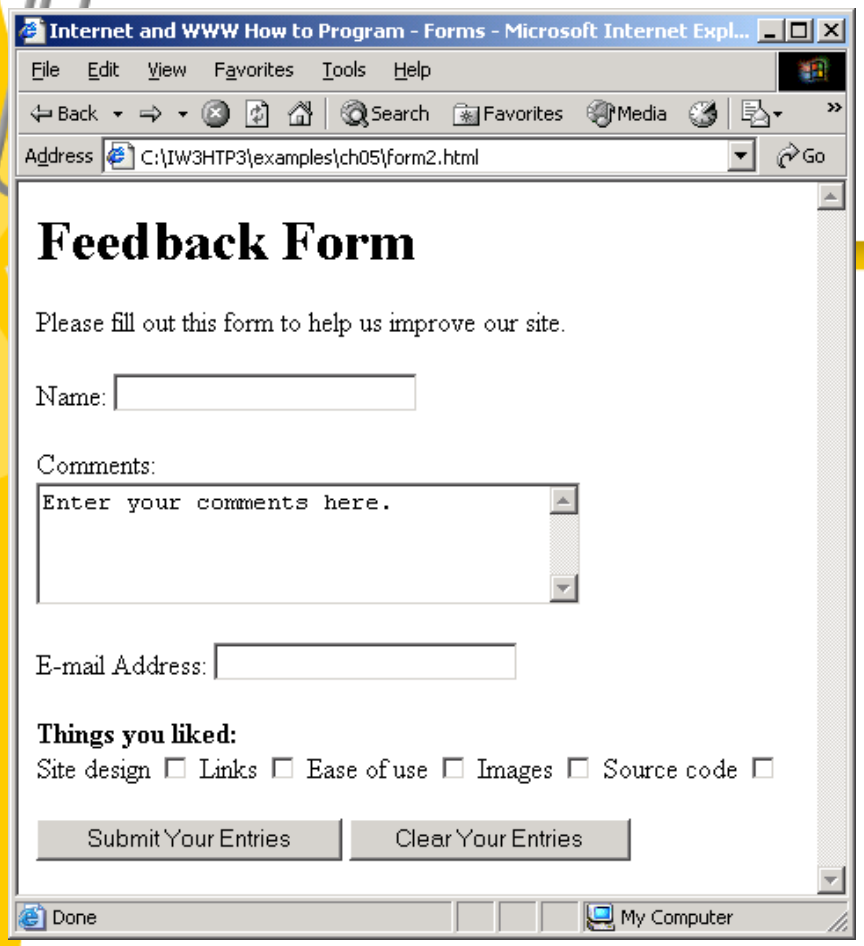
```
1  <?xml version =     "1.0" ?>
2  <!DOCTYPE html PUBLIC    "-//W3C//DTD XHTML 1.1//EN"
3      "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"    >
4
5  <!--  Fig. 5.4: form2.html          -->
6  <!--  Form Design Example 2       -->
7
8  <html xmlns =    "http:  //www.w3.org/1999/xhtml">
9    <head>
10         <title>  Internet and WWW How to Program    -  Forms </title>
11    </head>
12
13    <body>
14
15         <h1>Feedback Form </h1>
16
17         <p>Please fill out this form to help
18         us improve our site.          </p>
19
20         <form method =  "post"   action =    "/cgi -bin/formmail"   >
21
```

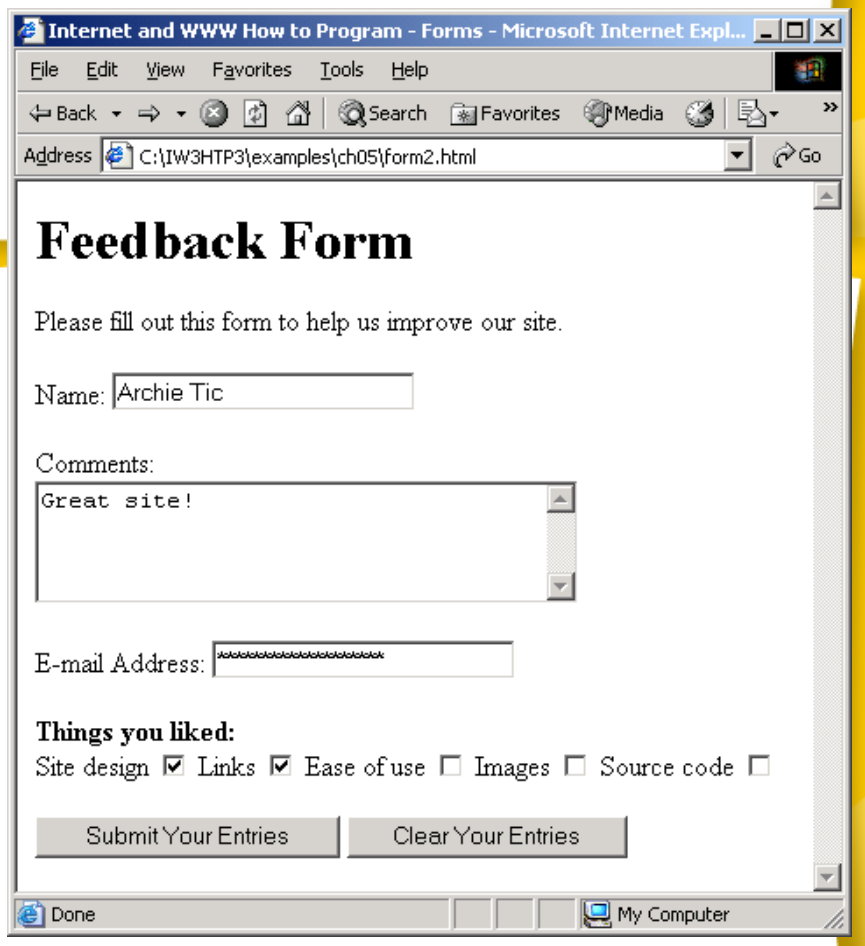
```
22          <p>
23             <input type =    "hidden"   name = "recipient"
24                value =    "deitel@deitel.com"      />
25             <input type =    "hidden"   name = "subject"
26                value =    "Feedback Form"   />
27             <input type =    "hidden"   name = "redirect"
28                value =    "main.html"    />
29          </p>
30
31          <p><label> Name:
32             <input name =   "name"  type =   "text"   size =   "25"  />
33          </label></p>
34
35          <!--  <textarea> creates a multiline textbox          -- >
36          <p><label> Comments:<br />
37             <textarea name =    "comments"  rows =   "4"  cols =    "36">
38 Enter your comments here.
39             </textarea>
40          </label></p>
41
```

```
42      <!-- <input type = "password"> inserts a   -->
43      <!-- textbox whose display is masked with  -->
44      <!-- asterisk characters                   -->
45      <p><label>E-mail Address:
46        <input name = "email" type = "password"
47          size = "25" />
48      </label></p>
49
50      <p>
51        <strong>Things you liked:</strong><br />
52
53        <label>Site design
54        <input name = "thingsliked" type = "checkbox"
55          value = "Design" /></label>
56
57        <label>Links
58        <input name = "thingsliked" type = "checkbox"
59          value = "Links" /></label>
60
61        <label>Ease of use
62        <input name = "thingsliked" type = "checkbox"
63          value = "Ease" /></label>
64
65        <label>Images
66        <input name = "thingsliked"  type = "checkbox"
67          value = "Images" /></label>
```

Notare che *(con l'input type checkbox)* possono essere inviati più valori in corrispondenza dello stesso nome di parametro

```
68
69              <label> Source code
70              <input name =   "thingsliked"     type =  "checkbox"
71                  value =  "Code"  /></label>
72          </p>
73
74          <p>
75              <input type =    "submit"  value =
76          "Submit Your Entries"              />
77              <input type =    "reset"   value =
78          "Clear Your Entries"               />
79          </p>
80
81      </form>
82
83    </body>
84 </html>
```


Internet and WWW How to Program - Forms - Microsoft Internet Expl...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\IW3HTP3\examples\ch05\form2.html
Go
Feedback Form
Please fill out this form to help us improve our site.
Name:
Comments:
Enter your comments here.
E-mail Address:
Things you liked:
Site design Links Ease of use Images Source code
Submit Your Entries
Clear Your Entries
Done
My Computer


Internet and WWW How to Program - Forms - Microsoft Internet Expl...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\IW3HTP3\examples\ch05\form2.html
Go
Feedback Form
Please fill out this form to help us improve our site.
Name: Archie Tic
Comments:
Great site!
E-mail Address:
Things you liked:
Site design Links Ease of use Images Source code
Submit Your Entries
Clear Your Entries
Done
My Computer

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5
6
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Internet and WWW How to Program - Forms</title>
11    </head>
12
13    <body>
14
15       <h1>Feedback Form</h1>
16
17       <p>Please fill out this form to help
18          us improve our site.</p>
19
20       <form method = "post" action = "/cgi-bin/formmail">
21
22          <p>
23             <input type = "hidden" name = "recipient"
24                value = "deitel@deitel.com" />
25             <input type = "hidden" name = "subject"
```

```
26              value =     "Feedback Form"   />
27          <input type =     "hidden"   name = "redirect"
28              value =     "main.html"   />
29      </p>
30
31      <p><label>   Name:
32          <input name =      "name" typ e =  "text"   size =   "25"  />
33        </label></p>
34
35      <p><label>   Comments:<br />
36          <textarea name =      "comments" rows =  "4"
37            cols =       "36"></textarea>
38        </label></p>
39
40      <p><label>   E- mail Address:
41          <input name =      "email"   type =  "password"
42          size =         "25"  /></label></p>
43
44      <p>
45          <strong>   Things you liked:    </strong><br />
46
47          <label>   Site design
48            <input name =      "thingsliked"     type =  "checkbox"
49              value =      "Design"   /></label>
50
```

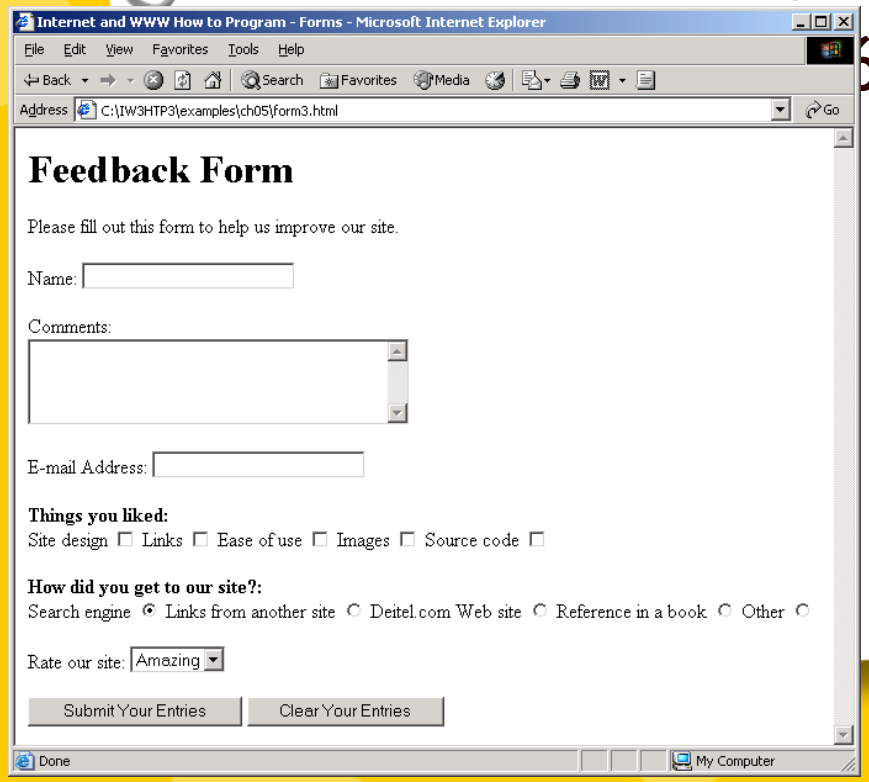
```
51          <label>    Links
52            <input name =      "thingsliked"     type =  "checkbox"
53               value =      "Links"    /></label>
54
55          <label>    Ease of use
56            <input name =      "thingsliked"     type =  "checkb ox"
57               value =      "Ease"   /></label>
58
59          <label>    Images
60            <input name =      "thingsliked"     type =  "checkbox"
61               value =      "Images"   /></label>
62
63          <label>    Source c  ode
64            <input name =      "thingsliked"     type =  "checkbox"
65               value =      "Code"  /></label>
66        </p>
67
68          <!--  <input type = "radio" /> creates a radio                  -- >
69          <!--  button. The differ      ence between radio buttons        -- >
70          <!--  and checkboxes is that only one radio button              -- >
71          <!--  in a group can be selected.                               -- >
72        <p>
73           <strong>   How did you get to our site?:      </strong><br   />
74
```

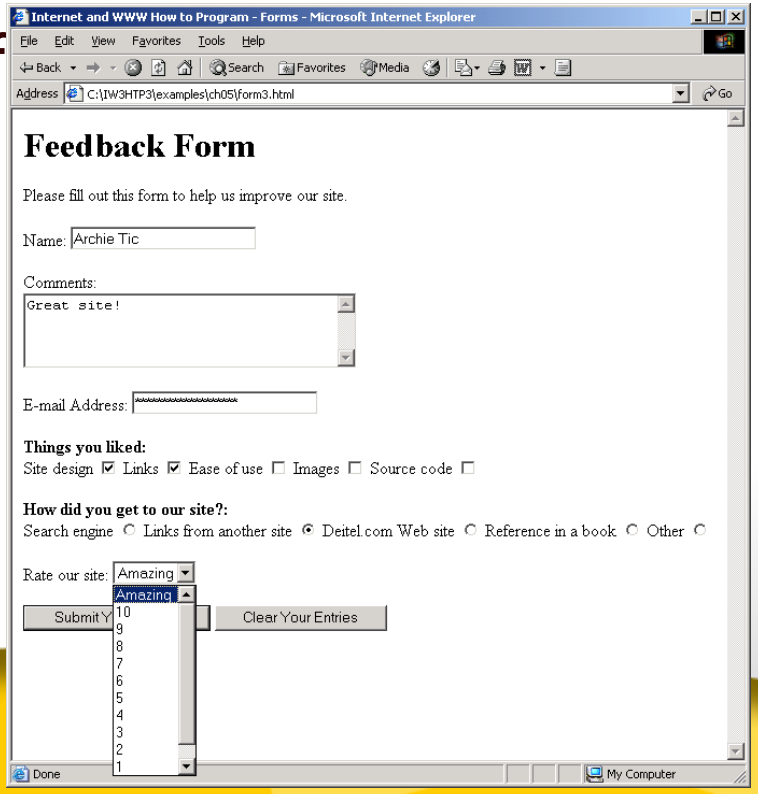
```
75          <label>    Search engine
76              <input name =    "howtosite"    type =   "radio"
77                  value =   "search engine"     checked =   "checked"  />
78      </label>
79
80          <label>    Links from    another site
81            <input name =       "howtosite"    type =   "radio"
82              value =       "link"    /></label>
83
84          <label>    Deitel.com Web site
85            <input name =       "howtosite"   type =   "radio"
86              value =       "deitel.com"     /></label>
87
88          <label>    Reference in a book
89            <input name =       "howtosite"    type =   "radio"
90              value =       "book"  /></label>
91
92          <label>    Other
93            <input name =       "howtosite"    type =   "radio"
94                  value =   "other"   /></label>
95
96       </p>
97
```

```
98      <p>
99       <label>Rate our site:
100
101         <!-- the <select> tag presents a drop-down -->
102         <!-- list with choices indicated by the    -->
103         <!-- <option> tags                         -->
104        <select name = "rating">
105          <option selected = "selected">Amazing</option>
106          <option>10</option>
107          <option>9</option>
108          <option>8</option>
109          <option>7</option>
110          <option>6</option>
111          <option>5</option>
112          <option>4</option>
113          <option>3</option>
114          <option>2</option>
115          <option>1</option>
116          <option>Awful</option>
117        </select>
118
119       </label>
120      </p>
121
```

```
122          <p>
123             <input type =      "submit"   value =
124           "Submit Your Entries"              />
125             <input type =      "reset"   value =   "Clear Your Entries"       />
126          </p>
127
128       </form>
129
130    </bo  dy>
131 </html>
```

form3.html





Meta-elementi

- Specificano informazioni su un documento attraverso l'uso del tag <meta>
- Attributo name
  - Identifica il tipo di metalemento
  - "keywords" (name = "keywords")
    - Fornisce ai motori di ricerca un elenco di parole con cui indicizzare la pagina
  - "description" (name = "description")
    - Fornisce la descrizione del sito
- Attributo content
  - Definisce il contenuto del meta-tag, (es. la lista delle keywords o la descrizione)

```
1  <?xml version =     "1.0" ?>
2  <!DOCTYPE html PUBLIC    "-//W3C//DTD XHTML 1.1//EN"
3      "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"    >
4
5  <!--  Fig. 5.8: main.html          -->
6  <!--  <meta> tag                   -->
7
8  <html xmlns =    "http://ww w.w3.org/1999/xhtml" >
9    <head>
10     <title>      Internet and WWW How to Program     -  Welcome</title>
11
12     <!     --  <meta> tags provide search engines with          -->
13     <!     --  information used to catalog a site               -->
14        <meta name = "keywords"  content =     "Web page, design,
15           XHTML, tutorial, personal, help, index, form,
16           contact, feedback, list, links, frame, deitel"                 />
17
18        <meta name =  "description"      content =    "This Web site will
19           help you learn the basics of XHTML and Web page design
20           through the use of interactive examples and
21           instruction."       />
22
23   </head>
24
```

```
25    <body>
26
27       <h1>Welcome to Our Web Site!</h1>
28
29       <p>We have designed this site to teach about the wonders
30       of <strong><em>XHTML</em></strong>. <em>XHTML</em> is
31       better equipped than <em>HTML</em> to represent complex
32       data on the Internet. <em>XHTML</em> takes advantage of
33       XML's strict syntax to ensure well-formedness. Soon you
34       will know about many of the great new features of
35       <em>XHTML.</em></p>
36
37       <p>Have Fun With the Site!</p>
38
39    </body>
40 </html>
```

Caratteristiche del protocollo HTTP

E' il protocollo per il trasferimento di iper-testi (HyperText Transfer Protocol)

- Corrisponde al livello applicativo (stack iso/osi)
  - Presuppone un protocollo (TCP) orientato alla connessione
- Meccanismo di Richiesta/Risposta (Client/Server)
- E' possibile un trasferimento bi-direzionale di informazioni
  - il client può inviare informazioni a un server tramite un form, il server invia (di solito) pagine web al client
- Basato sul meccanismo di naming degli URI
- **Senza stato**
  - **Ogni richiesta HTTP è autonoma, il server non tiene una cronologia delle richieste**

Gestione della sessione tramite parametri hidden

Gestione della sessione tramite parametri hidden

(*) *Il testo della pagina html viene generato dinamicamente dalla servlet in funzione dei parametri ricevuti nella richiesta, sia che fossero hidden sia che fossero visibili nel form*

Interazione Client - Web Server

- Sempre basata sul protocollo HTTP indipendentemente dalla soluzione adottata dal lato del server
- L'URL oggetto della richiesta è usato per selezionare la risorsa lato server che si vuole usare

Protocollo HTTP

- Protocollo stateless
  - Ad ogni richiesta del client segue una risposta del server. Nessuna correlazione tra richieste successive.
- Quando un client invia una richiesta al server, specifica un comando
- La prima linea della richiesta contiene il nome del comando, un URL e la versione del protocollo in uso:

```
GET /main.html HTTP/1.0
```

Protocollo HTTP (segue)

- Alla richiesta vengono appese informazioni opzionali
  - versione del browser,
  - i tipi di file che possono essere elaborati…

```
User-Agent: Mozilla/4.0 (compatible; MIE 4.0; Windows 95)
Accept: image/gif, image/jpeg, text/*, */*
```

Protocollo HTTP (segue)

- Il server elabora la richiesta e spedisce una risposta, specificando la versione del protocollo e uno status code (header della risposta):

```
HTTP/1.0 200 OK
```

- Altre informazioni inviate nell'header della risposta
  - Server software
  - Content-type della risposta

Protocollo HTTP: GET & POST

- GET e POST sono i comandi HTTP utilizzati più frequentemente
- Progettati inizialmente per scopi diversi
  - GETting information (read)
  - POSTing information (write)

- GET: informazioni utili per formulare la richiesta appese all'URL
- POST: informazioni incluse nel corpo della richiesta

Protocollo HTTP: GET

- I parametri della richiesta sono visibili
- La lunghezza della query string è limitata dal browser
  - Molti browser non consentono l'uso di query string di più di 240 caratteri
- La richiesta può essere inserita nei bookmark e ripetuta quante volte si vuole
- Si può ripetere la richiesta effettuando il "reload" dal browser

Protocollo HTTP: POST

- I parametri della richiesta non sono visibili all'utente
- La quantità di dati che si possono inviare è illimitata (anche megabytes)
- Le richieste di POST non possono essere inserite nei bookmark, né spedite via email o ricaricate.

Protocollo HTTP: GET e POST

- La distinzione funzionale con cui i metodi erano stati progettati si è persa MA persistono differenze sostanziali
- GET
  - parametri visibili
  - lista breve
  - inseribile nei bookmark e ripetibile
- POST:
  - parametri nascosti
  - lunghezza illimitata
  - non inseribile nei bookmark e non ripetibile
- Non usare richieste di GET per gestire ordini o aggiornamenti di un database o trasferire password

Sviluppo di applicazioni di rete

- Descrizione del paradigma client-server
- Implicazioni del paradigma client-server nel funzionamento dei sistemi web
  - Funzionamento di un web server
  - Generazione di pagine dinamiche con tecnologie lato client e lato server

Tecnologie lato client o lato server?

lato client all'interno di Applet, o attraverso metodi di scripting dal lato client (JavaScript), o in applicazioni stand-alone.

Richiedono il supporto del client

lato server associato a Servlet, agli Enterprise JavaBean, agli Agenti, alle API ed ai servizi di Transaction Management, agli Application Server ed ai protocolli di programmazione distribuita (es. CORBA)

Non richiedono alcun supporto da parte del client

Perché soluzioni lato server

- Soluzioni lato client
  - problemi di prestazioni e di portabilità
- Soluzioni lato server
  - accesso a informazioni che sono disponibili esclusivamente dal lato del server (es. database etc.)
  - minimi requisiti in termini di capacità di calcolo e storage dal lato del client (è il server a fare il grosso del lavoro)
  - il client non deve avere altro che il browser per interpretare le pagine html

Soluzioni lato server

- CGI (un processo - una richiesta a programma cgi)
  - Il web server inoltra le richieste a programmi esterni
  - L'output generato dal programma esterno viene intercettato dal web server e spedito al client nella forma di una pagina html (generata dinamicamente dal programma cgi)
  - Per ogni richiesta del client, il server deve creare un nuovo processo
    - Forte limitazione sul numero di richieste che possono essere gestite in parallelo
- Fast-CGI (un processo - un programma cgi)
  - Soffre ancora del problema della proliferazione dei processi, uno per ogni programma CGI

Altre soluzioni lato server

- ASP
  - Consente l'inserimento di codice all'interno delle pagine HTML (HTML-embedded), che viene eseguito dal server
  - E' ottimizzato per la generazione di piccole porzioni di contenuto dinamico
- PHP
  - Consente l'uso di codice "HTML-embedded"
  - Uso di un linguaggio interamente nuovo e poca disponibilità di API rispetto a JSP e Servlet

Servlet

- Programma applicativo che viene eseguito da un server
  - Accoglie ed elabora richieste provenienti dal client (attraverso comandi http: POST o GET, ma anche attraverso altri protocolli)
  - Produce una risposta HTTP contenente codice HTML generato dinamicamente
- Molto diffuse per applicazioni che fanno uso di database
  - Thin clients (client molto semplici che richiedono supporto minimo)
  - Meccanismo request/response

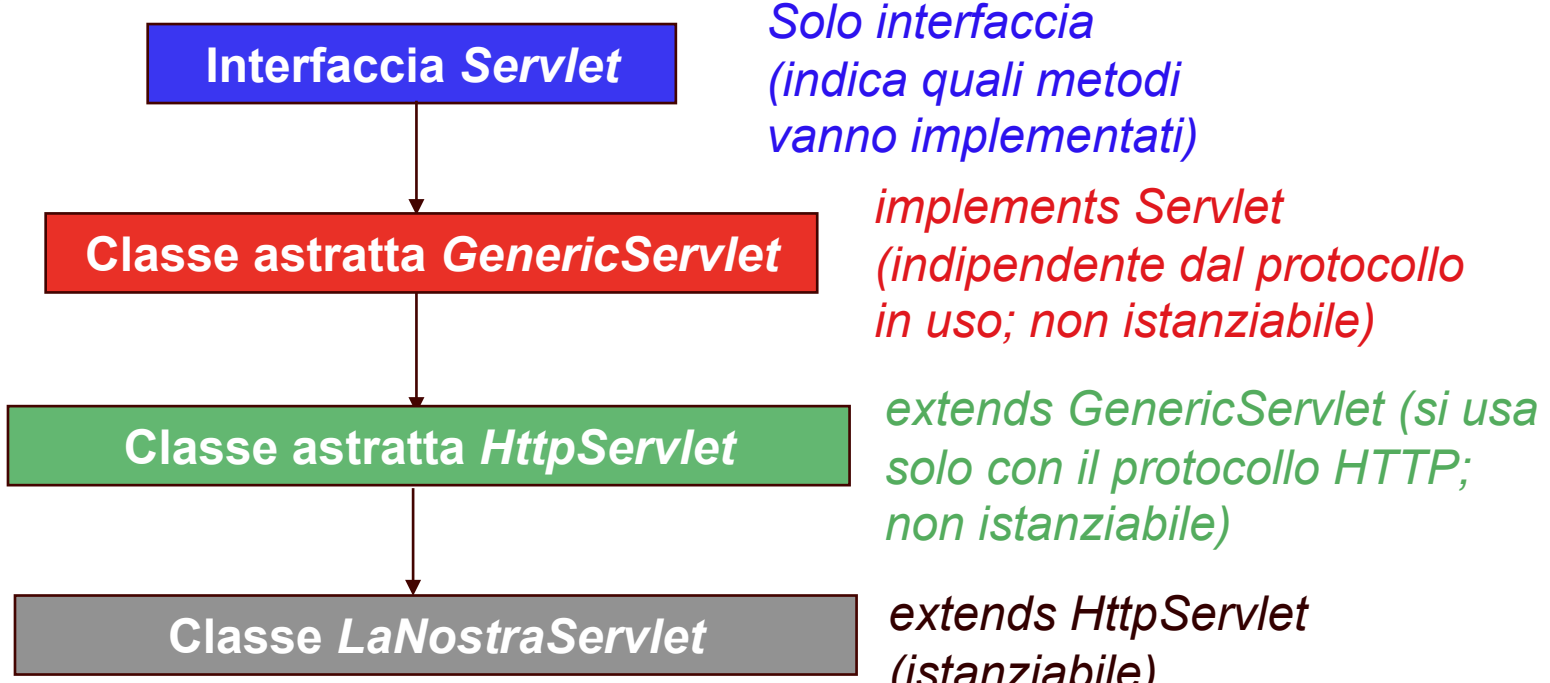
Servlet e Servlet Container

- Una servlet è una classe Java (che implementa l'interfaccia `Servlet`).
- Un servlet container può ospitare più servlet (con relativi alias).
- Quando una servlet viene invocata per la prima volta, il servlet engine genera un thread Java che inizializza l'oggetto Servlet.
  - Questo *persiste* per tutta la durata del processo relativo al servlet container (salvo esplicita de-allocazione).
- Ogni servlet è un thread all'interno del Servlet Container (vs CGI dove viene eseguito un processo esterno)

Possibili usi di una Servlet

- Supportare richieste multiple in modo concorrente, come per esempio conferenze on-line, servizi di comunicazione
- Aggiornare, eliminare o consultare dati contenuti in un DB remoto tramite protocollo TCP/IP
- Applicazioni basate sull'autenticazione degli utenti, gestione di sessioni di navigazione con creazione di oggetti persistenti
- Ottimizzazione delle prestazioni tramite redirezione di richieste ad altre Servlet in altri server, allo scopo di bilanciare il carico di lavoro

Servlet API



Solo interfaccia (indica quali metodi vanno implementati)

implements Servlet (indipendente dal protocollo in uso; non istanziabile)

extends GenericServlet (si usa solo con il protocollo HTTP; non istanziabile)

extends HttpServlet (istanziabile)

Servlet

- Le servlet possono essere utilizzate qualunque sia il servizio espletato dal server, ovvero qualunque sia il protocollo di interazione client/server: es HTTP, FTP
- Nella sua forma più generale, una servlet è un'estensione di una classe javax.servlet.GenericServlet che implementa l'interfaccia javax.servlet.Servlet
- Le servlet usate nel web sono estensioni della classe javax.servlet.http.HttpServlet che implementa l'interfaccia javax.servlet.Servlet

Interfaccia `Servlet`

- Interfaccia `Servlet` (package *`javax.servlet`*)
  - Tutte le servlet devono implementare questa interfaccia
  - Tutti i metodi dell'interfaccia `Servlet` vengono invocati dal servlet container secondo un prefissato *ciclo di vita*
    - (vi ricordate come funziona con le applet? Init-start-paint-stop-destroy vengono sempre chiamati in sequenza dal browser per eseguire una applet)

Ciclo di vita di una servlet

- Caricamento della servlet in memoria
- Il container delle servlet invoca il metodo `init`
  - Solo in questo momento la servlet è in grado di rispondere alla prima richiesta
  - Inizializzazione delle variabili globali
  - Invocato una sola volta
- Il metodo `service` gestisce le richieste
  - Riceve la richiesta
  - Elabora la richiesta
  - Confeziona un oggetto risposta
  - Invocato ad ogni richiesta del client
- Il metodo `destroy` rilascia le risorse allocate dalla servlet quando il container la termina

Classi astratte per definire le Servlet

- Esistono due classi astratte che implementano l'interfaccia *Servlet*
  - **`GenericServlet` (package `javax.servlet`)**
  - **`HttpServlet` (package `javax.servlet.http`)**
    (quest'ultima implementa l'interfaccia servlet indirettamente, estendendo GenericServlet)
- Queste classi forniscono l'implementazione di default di tutti i metodi dell'interfaccia *Servlet*
- Il metodo chiave è *service*:

  riceve gli oggetti ServletRequest e ServletResponse che forniscono accesso agli stream di i/o permettendo la ricezione e l'invio di informazioni al client

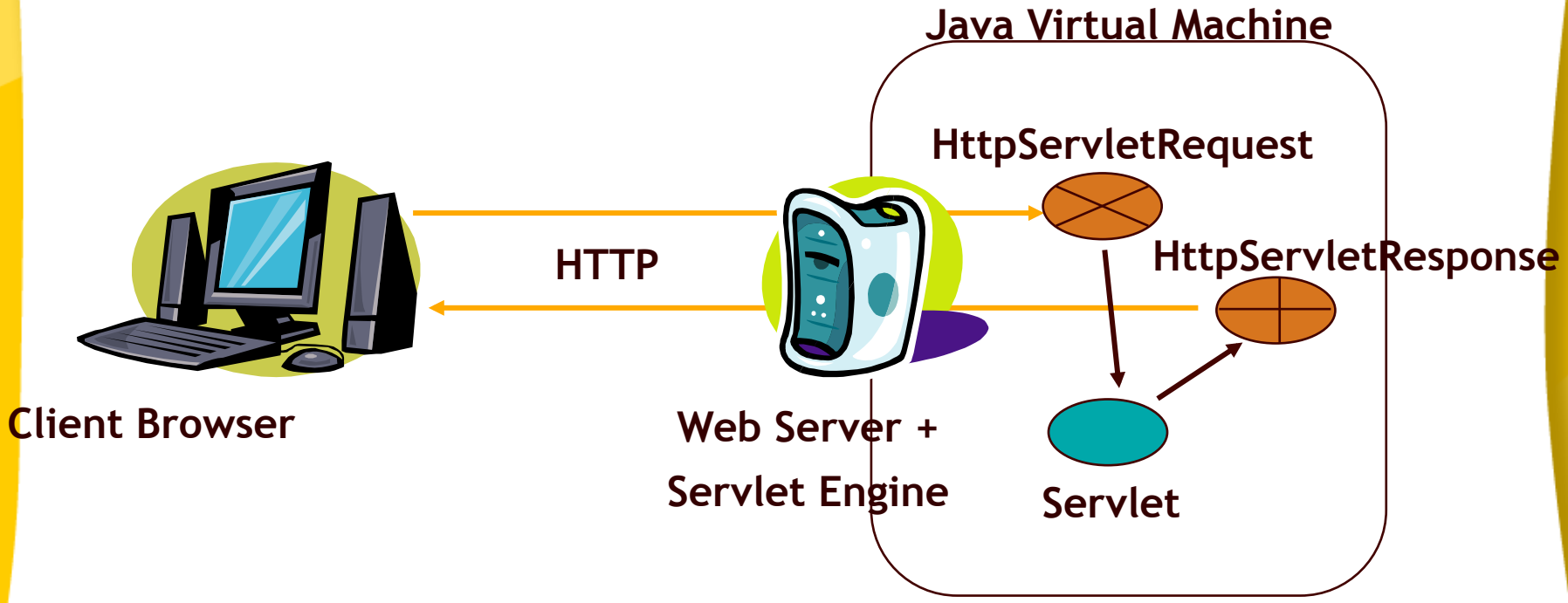
Metodi dell'interfaccia Servlet

Method	Description
void init(ServletConfig config)	
	The servlet container calls this method once during a servlet's execution cycle to initialize the servlet. The ServletConfig argument is supplied by the servlet container that executes the servlet.
ServletConfig getServletConfig()	
	This method returns a reference to an object that implements interface ServletConfig. This object provides access to the servlet's configuration information such as servlet initialization parameters and the servlet's ServletContext, which provides the servlet with access to its environment (i.e., the servlet container in which the servlet executes).
String getServletInfo()	
	This method is defined by a servlet programmer to return a string containing servlet information such as the servlet's author and version.
void service(ServletRequest request, ServletResponse response)	
	The servlet container calls this method to respond to a client request to the servlet.
void destroy()	
	This "cleanup" method is called when a servlet is terminated by its servlet container. Resources used by the servlet, such as an open file or an open database connection, should be deallocated here.

Funzionamento delle servlet HTTP

- Il server web riceve dal browser del client una richiesta HTTP GET o POST
- Il server web direziona la richiesta HTTP al motore servlet o servlet engine (pr. èngine) o servlet container
- Se la servlet non è ancora in memoria viene caricata e inizializzata dal servlet engine (esecuzione del metodo init)
- Il servlet engine incapsula la richiesta HTTP in una classe **HttpServletRequest** e la passa al metodo doPost o doGet della servlet
- La servlet risponde scrivendo il codice HTML nella **HttpServletResponse** che viene rimandata al web server e poi riconsegnata al client via HTTP

Servlet e Web Server



La classe HttpServlet

- Sovrascrive il metodo service della classe GenericServlet
- Prevede i metodi per rispondere alle richieste HTTP
  - GET
  - POST (ma anche HEAD, PUT, OPTIONS ecc.)
- Il metodo service() invoca i metodi doGet e doPost
  - Il metodo doGet() risponde alle richieste GET
  - Il metodo doPost() risponde alle richieste POST
    - Ricevono come parametri gli oggetti HttpServletRequest e HttpServletResponse
- Non implementare il metodo service se si usa questo tipo di servlet

Altri metodi della classe `HttpServlet`

Method	Description
doDelete	Called in response to an HTTP *delete* request. Such a request is normally used to delete a file from a server. This may not be available on some servers, because of its inherent security risks (e.g., the client could delete a file that is critical to the execution of the server or an application).
doHead	Called in response to an HTTP *head* request. Such a request is normally used when the client only wants the headers of a response, such as the content type and content length of the response.
doOptions	Called in response to an HTTP *options* request. This returns information to the client indicating the HTTP options supported by the server, such as the version of HTTP (1.0 or 1.1) and the request methods the server supports.
doPut	Called in response to an HTTP *put* request. Such a request is normally used to store a file on the server. This may not be available on some servers, because of its inherent security risks (e.g., the client could place an executable application on the server, which, if executed, could damage the server —perhaps by deleting critical files or occupying resources).
doTrace	Called in response to an HTTP *trace* request. Such a request is normally used for debugging. The implementation of this method automatically returns an HTML document to the client containing the request header information (data sent by the browser as part of the request).

Interfaccia `HttpServletRequest`

- Il servlet engine che esegue la servlet
  - Crea un oggetto `HttpServletRequest`
  - Lo passa al metodo `service` della servlet http
- L'oggetto `HttpServletRequest` contiene la richiesta del client
- Esistono molti metodi per estrarre informazioni dalla richiesta del client. Ne vediamo alcuni →

Interfaccia `HttpServletRequest` (Cont.)

Method	Description
String getParameter(String name)	
	Obtains the value of a parameter sent to the servlet as part of a get or post request. The name argument represents the parameter name.
Enumeration getParameterNames()	
	Returns the names of all the parameters sent to the servlet as part of a post request.
String[] getParameterValues(String name)	
	For a parameter with multiple values, this method returns an array of strings containing the values for a specified servlet parameter.
Cookie[] getCookies()	
	Returns an array of Cookie objects stored on the client by the server. Cookie objects can be used to uniquely identify clients to the servlet.
HttpSession getSession(boolean create)	
	Returns an HttpSession object associated with the client's current browsing session. This method can create an HttpSession object (true argument) if one does not already exist for the client. HttpSession objects are used in similar ways to Cookies for uniquely identifying clients.

Interfaccia `HttpServletResponse`

- Il servlet engine
  - Crea un oggetto `HttpServletResponse`
  - Lo passa al metodo `service` della servlet (attraverso i metodi doGet e doPost)
- Ogni chiamata ai metodi `doGet` e `doPost` riceve un oggetto che implementa l'interfaccia `HttpServletResponse`
- Esistono molti metodi che consentono la scrittura della risposta da inviare al client. Ne vediamo alcuni →

Interfaccia HttpServletResponse

Method	Description
void addCookie(Cookie cookie)	
	Used to add a Cookie to the header of the response to the client. The Cookie's maximum age and whether Cookies are enabled on the client determine if Cookies are stored on the client.
ServletOutputStream getOutputStream()	
	Obtains a byte-based output stream for sending binary data to the client.
PrintWriter getWriter()	
	Obtains a character-based output stream for sending text data to the client.
void setContentType(String type)	
	Specifies the MIME type of the response to the browser. The MIME type helps the browser determine how to display the data (or possibly what other application to execute to process the data). For example, MIME type "text/html" indicates that the response is an HTML document, so the browser displays the HTML page.

Servlet container

Jakarta project

- Progetto Jakarta (Apache Software Foundation)
- Implementazione di riferimento degli standard per la realizzazione di servlet e di Java Server Pages

- Obiettivo: *"[...] to provide commercial-quality server solutions based on the Java Platform that are developed in an open and cooperative fashion"*

Architetture per l'esecuzione di servlet

- Servlet container (o servlet engine)
  - E' il server che esegue una servlet
- Servlet e JSP sono supportate direttamente o attraverso plugin dalla maggior parte dei Web servers e application server
  - Apache TomCat
  - Sun ONE Application Server
  - Microsoft's Internet Information Server (IIS)
  - Apache HTTP Server + modulo JServ
  - BEA's WebLogic Application Server
  - IBM's WebSphere Application Server
  - World Wide Web Consortium's Jigsaw Web Server

Configurare il server Apache Tomcat (1)

1. Installare j2sdk
2. Definire la variabile PATH=c:\j2sdk---\bin (per poter trovare il compilatore)
5. Definire la variabile JAVAHOME=c:\j2sdk--- (perché tomcat trovi la jvm)
6. Installare il server TomCat

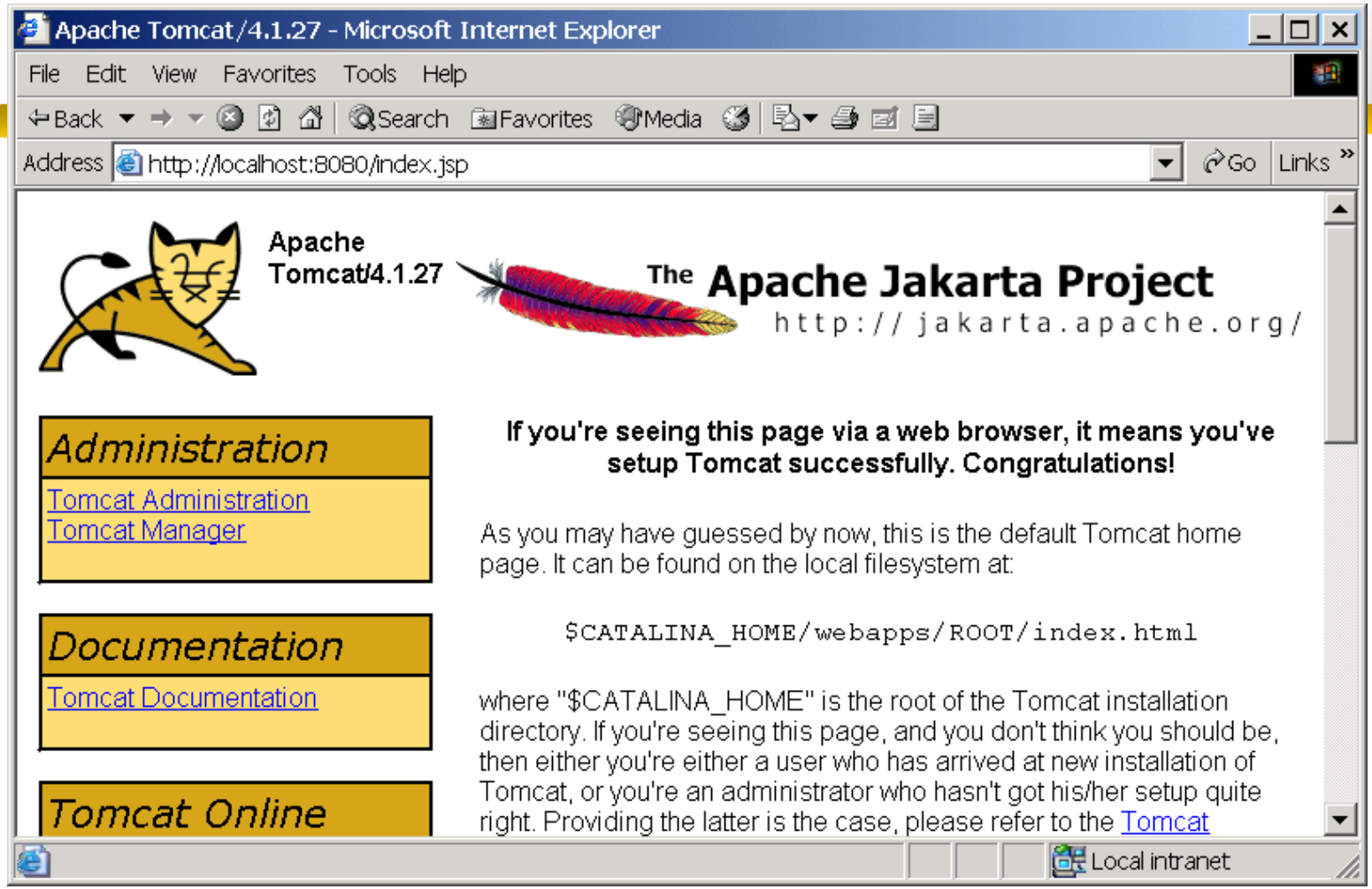
Configurare il server Apache Tomcat (2)

5. Definire la variabile CATALINA_HOME=c:\Programmi\Apache…\Tomcat---\

6. Definire la variabile CLASSPATH=c:\Programmi\Apache…\Tomcat---\common\lib\servlet-api.jar;c:\Programmi\Apache…\Tomcat---\common\lib\jsp-api.jar

7. TEST: Avviare il server Tomcat (startup), invocare il server da un browser all'indirizzo http://localhost:8080/

Configurare il server Apache Tomcat



Se Tomcat è installato correttamente dovreste vedere questa pagina all'indirizzo http://127.0.0.1:8080/

```
1
2  // A simple servlet to process get requests.
3
4  import javax.servlet.*;
5  import javax.servlet.http.*;
6  import java.io.*;
7
8  public class ServletDiSaluto extends HttpServlet {
9
10   // process "get" requests from clients
11   protected void doGet( HttpServletRequest request,
12     HttpServletResponse response )
13       throws ServletException, IOException
14    {
15     response.setContentType( "text/html" );
16     PrintWriter out = response.getWriter();
17
18     // send XHTML page to client
19
20     // start XHTML document
21     out.println( "<?xml version = \"1.0\"?>" );
22
23     out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
24       "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
25       "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
```

Importa i package `javax.servlet` e `javax.servlet.http`.

Estende `HttpServlet` in modo che gestisca le richieste HTTP `get` e `post`.

Sovrascrive il metodo `doGet` per fornire funzionalità personalizzate.

Utilizza l'oggetto `response` e il relativo metodo `setContentType` per specificare il tipo di contenuto dei dati che devono essere spediti in risposta al client.

Utilizza il metodo `getWriter` dell'oggetto `response` per ottenere un riferimento all'oggetto `PrintWriter` che abilita la servlet a spedire dati al client.

Crea il documento XHTML scrivendo stringhe attraverso il metodo `println` dell'oggetto `out`.

ServletDiSaluto.java

```
26
27      out.println(        "<html xmlns =    \"http://www.w3.org/1999/xhtml  \">" );
28
29      // head section of document
30      out.println(        "<head>" );
31      out.println(        "<title>A Simple Servlet Example</title>"          );
32          out.println(    "</head>" );
33
34      // body section of document
35      out.println(        "<body>" );
36      out.println(        "<h1>Welcome to Servlets!</h1>"    );
37      out.println(        "</body>" );
38
39      // end XHTML docu      ment
40      out.println(        "</html>" );
41          out.close();        // close stream to complete the page
42   }
43 }
```

Chiude l'output stream, esegue il flush del buffer di output e spedisce le informazioni al client.

ServletDiSaluto.java
(2 of 2)

File index.html

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!—File index.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml" >
8  <head>
9    <title>Handling an HTTP Get Request</title>
10  </head>
11
12  <body>
13    <form action = "/DirectoryDiSaluto/AliasServletDiSaluto" method = "get" >
14
15      <p><label>Click the button to invoke the servlet
16        <input type = "submit" value = "Get HTML Document" />
17      </label></p>
18
19    </form>
20  </body>
21  </html>
```

Struttura della directory di una web application

- Context root – top level directory di una applicazione web

  Si crea una sottodirectory nella directory webapps di Tomcat

Struttura della directory di una web application

Directory	Description
context root (nome che volete)	E' la directory radice della vostra applicazione web. Il nome è scelto dallo sviluppatore dell'applicazione (voi). Contiene tutti i file dell'applicazione, eventualmente organizzati in sottocartelle (file JSP, pagine HTML, immagini incorporate nelle pagine, possono risiedere dove volete, classi delle servlet e altre classi di supporto devono essere organizzate in cartelle e package. Si possono inserire sottocartelle nella context root per organizzarne meglio i contenuti. Negli esempi che seguono la context root si chiama **`directorydisaluto`**.
`WEB-INF`	Questa cartella contiene un file detto *deployment descriptor* (***`web.xml`***).
`WEB-INF/classes`	Questa cartella contiene le classi delle servlet e altre classi di supporto usate dalla web application. Se le classi fanno parte di un package, la struttura completa del package deve cominciare da questa cartella.
`WEB-INF/lib`	Questa cartella contiene archivi (JAR). I file JAR files possono contenere I file delle classi servlet e altre classi di supporto che sono utilizzate in una web application.

Deployment descriptor (web.xml)

- Dopo avere configurato l'albero delle directory, dobbiamo configurare l'applicazione web in modo che possa gestire le richieste

→ Si usa un file web.xml che chiamiamo

deployment descriptor

Specifica diversi parametri come:

- il nome della classe che definisce la servlet,
- i percorsi che causano l'invocazione della servlet da parte del container

```
1   <!DOCTYPE web-app PUBLIC
2     "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
3     "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
4
5   <web-app>
6
7     <!-- General description of your W
8     <display-name>
9       NomeDisplayDellaWebApplication
10        Viene visualizzato dall'admin
11    </display-name>
12
13    <description>
14      Si tratta di un'applcazione dove facciamo
15      Esempi di servlet.
16    </description>
17
18    <!-- Servlet definitions -->
37     <servlet>

20      <servlet-name>Paperino</servlet-nam
21
22      <description>
23        Una servlet che gestisce le richieste di GET
24      </description>
25
26      <servlet-class>
27        ServletDiSaluto
28      </servlet-class>
29    </servlet>
30
```

L'elemento <**web-app**> definisce la configurazione di tutte le servlet della applicazione web

L'elemento <**display-name**> specifica un nome che viene utilizzato dall'amministratore del server (e visualizzato dal pannello del manager)

L'elemento <**description**> specifica una descrizione della applicazione che viene visualizzata nel pannello del manager

L'elemento <**servlet**> descrive una specifica servlet: se ci sono più servlet ci saranno più elementi <servlet>

L'elemento <**servlet-name**> definisce un nome per la servlet all'interno di questo file

L'elemento <**description**> fornisce una descrizione di questa servlet specifica

L'elemento <**servlet-class**> specifica il nome completo della classe servlet compilata

```
31    <!-- Servlet mappings -->
32    <servlet-mapping>
33       <servlet-name>Paperino</servlet-name>
34       <url-pattern>/AliasServletDiSaluto</url-pattern>
35    </servlet-mapping>
36
37 </web-app>
```

L'elemento <**servlet-mapping**> specifica l'associazione tra il nome dato alla servlet nell'interno del file web.xml (<**servlet-name**>) e l'<**url-pattern**> con cui la servlet potrà essere invocata

n.b.: non si specifica la context root nell'url-pattern!

Ricordiamoci che all'interno del server la tecnologia è java, mentre il colloquio con il client avviene nel protocollo HTTP.

L'associazione tra l'URL usata nelle richieste HTTP e la classe java che deve essere eseguita dal server è lo scopo principale del deployment descriptor.

Tale associazione è creata nel file web.xml attraverso l'elemento <servlet-name> che viene associato prima con il nome della classe (<servlet-class>) e poi con l'URL (<url-pattern>).

n.b. se la servlet fosse stata parte di un package, avremmo dovuto includere nella directory **classes** l'intera struttura del package

```
31    <!-- Servlet mappings -->
32    <servlet-mapping>
33       <servlet-name>Paperino</servlet-name>
34       <url-pattern>/A/B/topolino</url-pattern>
35    </servlet-mapping>
36
37 </web-app>
```

L'elemento <**servlet-mapping**> specifica l'associazione tra il nome dato alla servlet nell'interno del file web.xml (<**servlet-name**>) e l'<**url-pattern**> con cui la servlet potrà essere invocata

Notare che /A e /B non corrispondono a nessuna cartella sul server!

Uso di percorsi relativi in una Web Application (1)

- Invocazione della servlet attraverso un **percorso relativo al server**:

  "/DirectoryDiSaluto/AliasServletDiSaluto"

  - **`/DirectoryDiSaluto`** specifica la context root
  - **`/AliasServletDiSaluto`** specifica l'URL pattern
  - Si tratta di un percorso relativo al server riferito alla radice del server (dir. webapps), comincia con il carattere "/"
  - Questo percorso va bene qualunque sia la posizione del file chiamante all'interno del server (anche al di fuori della context root, da un'altra applicazione purchè all'interno dello stesso server)

```
<form action = "/DirectoryDiSaluto/AliasServletDiSaluto" method = "get" >
```

Uso di percorsi relativi in una web application (2)

- Invocazione della servlet attraverso un **percorso relativo al file chiamante**:

  "percorso_x/AliasServletDiSaluto"

  - percorso_x specifica il percorso dalla cartella contenente il file chiamante alla context root dell'applicazione
  - Si tratta di un percorso relativo al file chiamante riferito alla posizione del file in cui è specificato l'url della servlet che si vuole invocare.
  - Il percorso relativo comincia senza il carattere "/"
  - Questo percorso va bene solo per la posizione del file chiamante e non può essere utilizzato in altre applicazioni, nemmeno se girano sullo stesso server

Uso dei percorsi relativi in una web application (3)

WelcomeServlet Web application directory and file structure

```
+ DirectoryDiSaluto
    - index.html
    + htmls
        - HtmlDiSaluto.html
    + WEB-INF
        - web.xml
        + classes
            -  ServletDiSaluto.class
```

Nel file web.xml l'url pattern indicato è
<url-pattern>/AliasServletDiSaluto</url-pattern>

Nel file index.html la servlet viene invocata nel seguente modo:

```
<form action = "AliasServletDiSaluto" method = "get" >
```

Nel file HtmlDiSaluto.html la servlet viene invocata nel seguente modo:

```
<form action = "../AliasServletDiSaluto" method = "get" >
```

Gestione di una richiesta HTTP `get` Contenente Dati

- Servlet **`WelcomeServlet2`**
  - Rispondere ad una richiesta di tipo **`get`** contenente dati

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9    <title>Processing get requests with data</title>
10 </head>
11
12 <body>
13   <form action = "/DirectoryDiSaluto/welcome2" method = "get">
14
15     <p><label>
16       Type your first name and press the Submit button
17       <br /><input type = "text" name = "firstname" />
18       <input type = "submit" value = "Submit" />
19     </p></label>
20
21   </form>
22 </body>
23 </html>
```

Get the first name from the user.

```
2  // Processing HTTP get requests containing data.
3
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8
9  public class WelcomeServlet2 extends HttpServlet {
10
11    // process "get" request from client
12    protected void doGet( HttpServletRequest request,
13       HttpServletResponse response )
14          throws ServletException, IOException
15    {
16       String firstName = request.getParameter( "firstname" );
17
18       response.setContentType( "text/html" );
19       PrintWriter out = response.getWriter();
20
21       // send XHTML document to client
22
23       // start XHTML document
24       out.println( "<?xml version = \"1.0\"?>" );
25
26       out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
27          "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
28          "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
29
30       out.println(
31          "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
32
```

Il metodo **getParameter** dell'oggetto **request** riceve come argomento il nome del parametro e restituisce il corrispondente vaolre sotto forma di una stringa (tipo **String**)

```
33      // head section of document
34      out.println( "<head>" );
35      out.println(
36         "<title>Processing get requests with data</title>" );
37      out.println( "</head>" );
38
39      // body section of document
40      out.println( "<body>" );
41      out.println( "<h1>Hello " + firstName + ",<br />" );
42      out.println( "Welcome to Servlets!</h1>" );
43      out.println( "</body>" );
44
45      // end XHTML document
46      out.println( "</html>" );
47      out.close();  // close stream to complete the page
48   }
49 }
```

La stringa ottenuta alla linea 16 viene usata per costruire la risposta al client.

Modifica del file web.xml per includere la nuova servlet

Descriptor element	Value
servlet *element*	
servlet-name	**Welcome2**
description	**Handling HTTP get requests with data.**
servlet-class	**WelcomeServlet2**
servlet-mapping *element*	
servlet-name	**Welcome2**
url-pattern	**/Welcome2**

Modifica del file web.xml per includere la nuova servlet

- Si aggiungono le seguenti direttive:

```
<servlet>
   <servlet-name>Welcome2</servlet-name>
   <display-name>
     NomeDisplayDiSalutoPersonalizzato</display-name>
   <description>Facciamo un test con personalizzazione</description>
   <servlet-class>WelcomeServlet2</servlet-class>
 </servlet>

<servlet-mapping>
   <servlet-name>Welcome2</servlet-name>
   <url-pattern>/Welcome2</url-pattern>
 </servlet-mapping>
```


Processing get requests with data - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/DirectoryDiSaluto/servlets/WelcomeServlet2.html
Go
Type your first name and press the Submit button
Paul
Submit
Done
Processing get requests with data - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/DirectoryDiSaluto/welcome2?firstname=Paul
Go
Hello Paul,
Welcome to Servlets!
Done
Local intranet

Servlet che gestisce richieste di POST

- HTTP post request
  - Spedire dati da un form HTML ad un handler di form localizzato sul server
  - Ricordare che i browser non fanno caching delle risposte a richieste di tipo POST
- Servlet WelcomeServlet3
  - Risponde ad una richiesta di tipo **`POST`** contenente dati

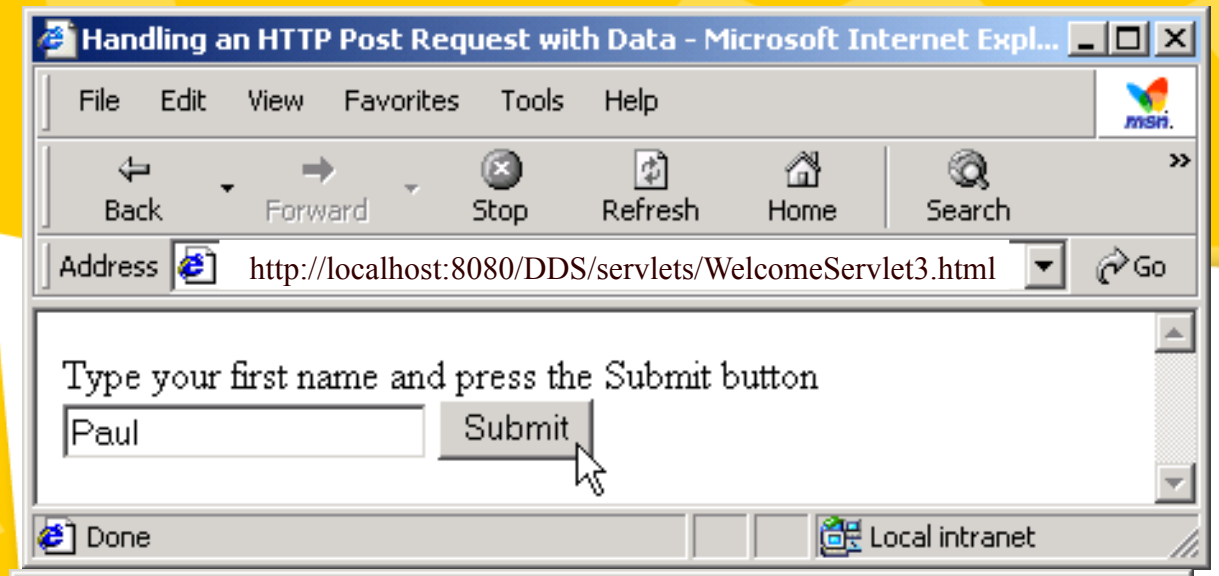
```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- WelcomeServlet3.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9    <title>Handling an HTTP Post Request with Data</title>
10 </head>
11
12 <body>
13   <form action = "/DirectoryDiSaluto/welcome3" method = "post">
14
15     <p><label>
16       Type your first name and press the Submit button
17       <br /><input type = "text" name = "firstname" />
18       <input type = "submit" value = "Submit" />
19     </label></p>
20
21   </form>
22 </body>
23 </html>
```

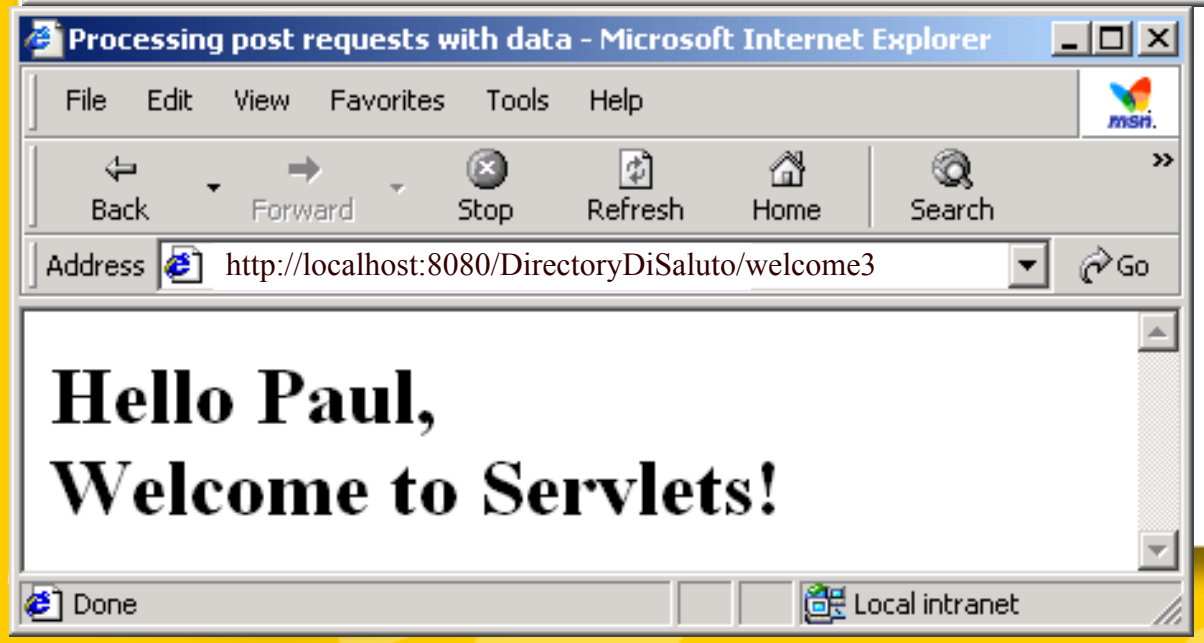
Viene fornito un **form** in cui l'utente può scrivere il proprio nome nell'elemento di input **firstname** di tipo **text**. Quando viene cliccato il bottone **Submit** viene invocata **WelcomeServlet3**.

```
2  // Processing post requests containing data.
3
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8
9  public class WelcomeServlet3 extends HttpServlet {
10
11    // process "post" request from client
12    protected void doPost( HttpServletRequest request,
13       HttpServletResponse response )
14          throws ServletException, IOException
15    {
16       String firstName = request.getParameter( "firstname" );
17
18       response.setContentType( "text/html" );
19       PrintWriter out = response.getWriter();
20
21       // send XHTML page to client
22
23       // start XHTML document
24       out.println( "<?xml version = \"1.0\"?>" );
25
26       out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
27          "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
28          "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
29
30       out.println(
31          "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
32
```

Define a **doPost** method to responds to post requests.

```
33      // head section of document
34      out.println( "<head>" );
35      out.println(
36         "<title>Processing post requests with data</title>" );
37      out.println( "</head>" );
38
39      // body section of document
40      out.println( "<body>" );
41      out.println( "<h1>Hello " + firstName + ",<br />" );
42      out.println( "Welcome to Servlets!</h1>" );
43      out.println( "</body>" );
44
45      // end XHTML document
46      out.println( "</html>" );
47      out.close();  // close stream to complete the page
48   }
49 }
```


Handling an HTTP Post Request with Data - Microsoft Internet Expl...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/DDS/servlets/WelcomeServlet3.html
Go
Type your first name and press the Submit button
Paul
Submit
Done
Local intranet


Processing post requests with data - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/DirectoryDiSaluto/welcome3
Go
Hello Paul,
Welcome to Servlets!
Done
Local intranet

Modifiche al file web.xml

Descriptor element	Value
servlet *element*	
servlet-name	**welcome3**
description	**Handling HTTP post requests with data.**
servlet-class	**WelcomeServlet3**
servlet-mapping *element*	
servlet-name	**welcome3**
url-pattern	**/welcome3**

Non dimenticate di ricaricare l'applicazione dopo avere ultimato e salvato le modifiche al file web.xml

Osservazione

Una stessa servlet può trattare in modo completamente diverso richieste di tipo GET e POST (basta scrivere in modi diversi i metodi doGet(…) e doPost(…).

Errori tipici

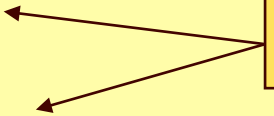
- Il compilatore java non trova il package javax.servlet.*
  - Avete scritto correttamente la variabile d'ambiente CLASSPATH?
- La classe viene compilata correttamente ma il tomcat manager non visualizza la vostra applicazione nella lista delle applicazioni attive
  - Avete rispettato la struttura delle cartelle di una web application? Il file web.xml è nella cartella WEB-INF?
  - Avete controllato la sintassi del file web.xml?
  - Avete messo la vostra classe compilata nella cartella WEB-INF/classes?
- L'applicazione è nell'elenco ma quando scrivete l'url pattern per invocare la servlet le cose non funzionano:
  - Sintassi del file web.xml: l'url pattern comincia con il carattere "/"?
  - L'associazione <servlet-class> all' <url-pattern> è descritta correttamente?
  - State invocando il server sulla porta 8080?
  - State usando il percorso corretto?

REDIREZIONE DI RICHIESTE

Inoltro delle richieste ad altre risorse

- Servlet di esempio **RedirectServlet**
  - Inoltra la richiesta ad una risorsa diversa
  - Fa uso del metodo sendRedirect() dell'oggetto HttpServletResponse
  - Il parametro di ingresso è una stringa: il percorso di destinazione della ridirezione
    - Il metodo accetta percorsi assoluti e relativi.
    - Un percorso relativo senza "/" iniziale viene interpretato rispetto alla URL corrente
    - Un percorso relativo con la "/" iniziale viene interpretato rispetto alla dir. radice del container

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!--  RedirectServlet.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9     <title>Redirecting a Request to Another Site</title>
10  </head>
11
12  <body>
13     <p>Click a link to be redirected to the appropriate page</p>
14     <p>
15     <a href = "/DirectoryDiSaluto/redirect?page=deitel">
16        www.deitel.com</a><br />
17     <a href = "/DirectoryDiSaluto/redirect?page=welcome1">
18        Welcome servlet</a>
19     </p>
20  </body>
21  </html>
```



Fornisce hyperlink che invocano **RedirectServlet**

```
1
2  // Redirecting a user to a different Web page.
3
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8
9  public class RedirectServlet extends HttpServlet {
10
11    // process "get" request from client
12    protected void doGet( HttpServletRequest request,
13      HttpServletResponse response )
14        throws ServletException, IOException
15    {
16      String location = request.getParameter( "page" );
17
18      if ( location != null )
19
20        if ( location.equals( "deitel" ) )
21          response.sendRedirect( "http://www.deitel.com" );
22        else
23          if ( location.equals( "welcome1" ) )
24            response.sendRedirect( "welcome1" );
25
51        // codice che viene eseguito solo se questa servlet non riesce a redirigere
53        // la richiesta come voluto
28
29      response.setContentType( "text/html" );
30      PrintWriter out = response.getWriter();
31
```

Attenzione all'uso di percorsi relativi all'interno di una servlet:

per non sbagliare pensate sempre attentamente a come viene formulato l'url per intero una volta che viene passato al browser

Il percorso relativo welcome1 viene aggiunto al percorso della servlet chiamante il metodo sendRedirect

Viene ottenuto il parametro **page** dalla richiesta.

Si valuta se il valore di **page** sia "deitel" o "welcome1"

Si inoltra la richiesta all'url: www.deitel.com.

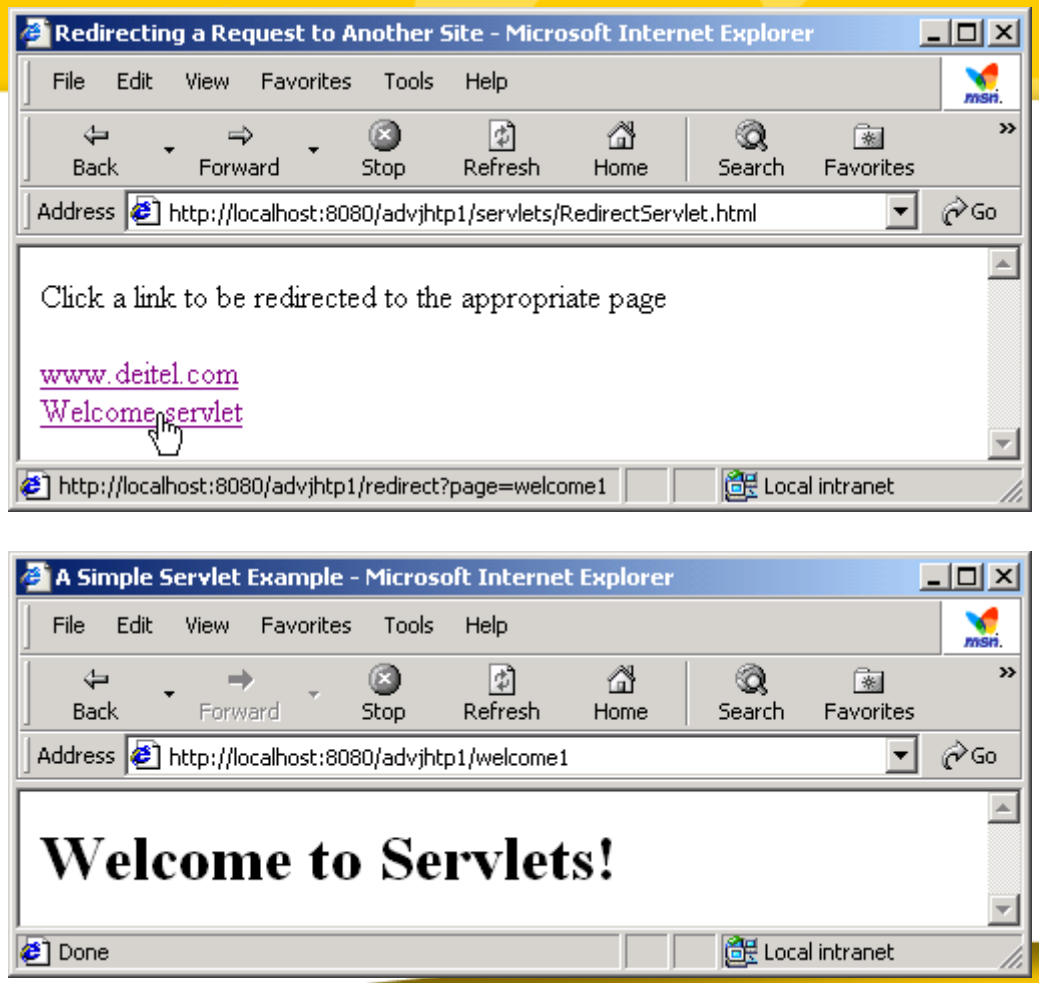
Si inoltra la richiesta al servlet **WelcomeServlet1**.

Pagina web di output che viene visualizzata nel caso in cui si sia ricevuta una richieta non valida (non viene invocato il metodo **sendRedirect**).

```
32      // start XHTML document
33      out.println( "<?xml version = \"1.0\"?>" );
34
35      out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
36        "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
37        "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
38
39      out.println(
40        "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
41
42      // head section of document
43      out.println( "<head>" );
44      out.println( "<title>Invalid page</title>" );
45      out.println( "</head>" );
46
47      // body section of document
48      out.println( "<body>" );
49      out.println( "<h1>Invalid page requested</h1>" );
50      out.println( "<p><a href = " +
51        "\"servlets/RedirectServlet.html\">" );
52      out.println( "Click here to choose again</a></p>" );
53      out.println( "</body>" );
54
55      // end XHTML document
56      out.println( "</html>" );
57      out.close();  // close stream to complete the page
58    }
59  }
```

Si noti come l'invocazione di altre risorse facenti parte della stessa web application non richieda di specificare esplicitamente la context root.

Si assume che la servlet invocata si trovi nella stessa context root a meno che non venga specificata una URL completa.


Redirecting a Request to Another Site - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/advjhtp1/servlets/RedirectServlet.html
Go
Click a link to be redirected to the appropriate page
www.deitel.com
Welcome servlet
http://localhost:8080/advjhtp1/redirect?page=welcome1
Local intranet
A Simple Servlet Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/advjhtp1/welcome1
Go
Welcome to Servlets!
Done
Local intranet

Inoltro delle richieste ad altre risorse: modifiche al file web.xml

Descriptor element	Value
servlet *element*	
servlet-name	**redirect**
description	**Redirecting to static Web pages and other servlets.**
servlet-class	**RedirectServlet**
servlet-mapping *element*	
servlet-name	**redirect**
url-pattern	**/redirect**

??????

Come mai per redirigere una richiesta faccio uso di un metodo dell'oggetto

HttpServletResponse response?

Alcune precisazioni…

1. L'oggetto della classe **`HttpServletResponse`** su cui viene invocato il metodo **`sendRedirect(String location)`** viene comunque utilizzato dal web server per costruire la risposta HTTP.
2. La risposta HTTP contiene un header di redirezione verso la nuova locazione
3. L'header di redirezione viene interpretato dal browser del client
4. Il client spedisce automaticamente una nuova richiesta verso la nuova locazione.

Alcune precisazioni (continua)

- La locazione può anche essere esterna alla applicazione web da cui viene invocato il metodo (il metodo sendRedirect accetta sia percorsi relativi che assoluti)
- La redirezione coinvolge il client (che può decidere di non accettare redirezioni)
- Può essere utilizzata esclusivamente per richieste di GET (la specifica HTTP richiede che tutte le richieste di redirezione debbano essere inoltrate attraverso richieste di GET) – non si può usare sendRedirect per inviare richieste di POST
- I parametri della richiesta sono visibili al client (appesi all'URL nella richiesta di GET)

Documentazione dell'interfaccia HttpServletResponse, metodo sendRedirect

- **sendRedirect**
- public void **sendRedirect**(java.lang.String location) throws java.io.IOException
  - Sends a temporary redirect response to the client using the specified redirect location URL. This method can accept relative URLs; the servlet container must convert the relative URL to an absolute URL before sending the response to the client. If the location is relative without a leading '/' the container interprets it as relative to the current request URI. If the location is relative with a leading '/' the container interprets it as relative to the servlet container root. If the response has already been committed, this method throws an IllegalStateException. After using this method, the response should be considered to be committed and should not be written to.
  - **Parameters:**
    - location - the redirect location URL
  - **Throws:**
    - java.io.IOException - If an input or output exception occurs
    - IllegalStateException - If the response was committed or if a partial URL is given and cannot be converted into a valid URL

Session Tracking

- Personalizzazione delle informazioni presenti nella pagina web
- Problema della tutela delle informazioni private
- HTTP – è un protocollo stateless
  - Non supporta la persistenza delle informazioni
- Per tracciare i client individualmente:
  - Cookie
  - Session tracking
  - **`hidden`** type **`input`**
  - URL rewriting (parametri get)

I cookie

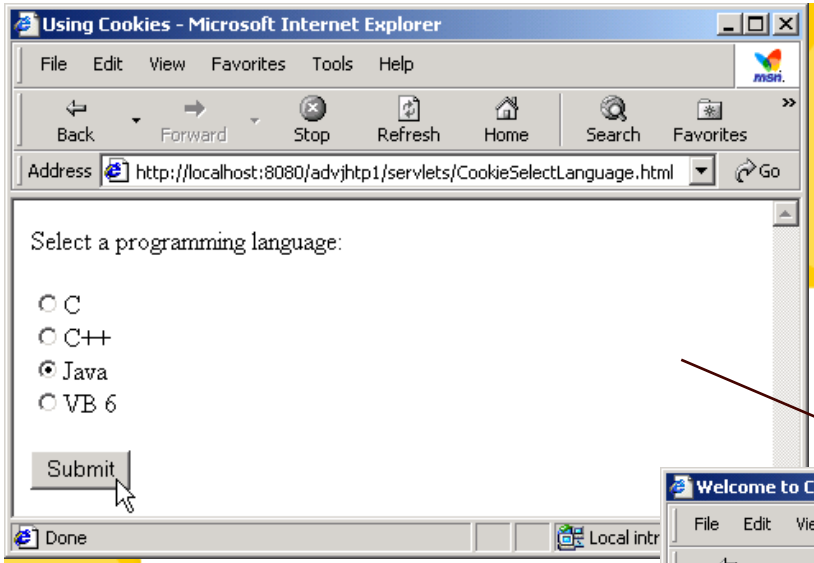
- Vengono memorizzati sul computer dell'utente (client) per utilizzi successivi
- Dati di tipo testo, spediti da servlet
- Età massima di un cookie
- Cancellati automaticamente quando scadono

- Esempio: servlet **CookieServlet**
  - Gestisce sia richieste di **get** che di **post**

Esempio

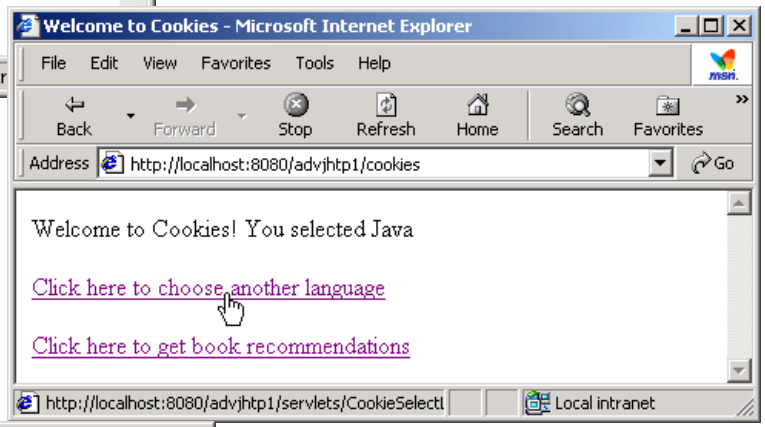
- Vogliamo scrivere una servlet che gestisca scelte che l'utente opera attraverso successive interazioni con l'applicativo.
  - L'utente sceglie un linguaggio di programmazione alla volta
  - Dopo una serie di scelte da parte dell'utente la servlet propone una serie di libri sugli argomenti selezionati.
  - N.B.: La servlet "ricorda" le scelte dell'utente nonostante l'interazione avvenga attraverso un protocollo stateless (http)

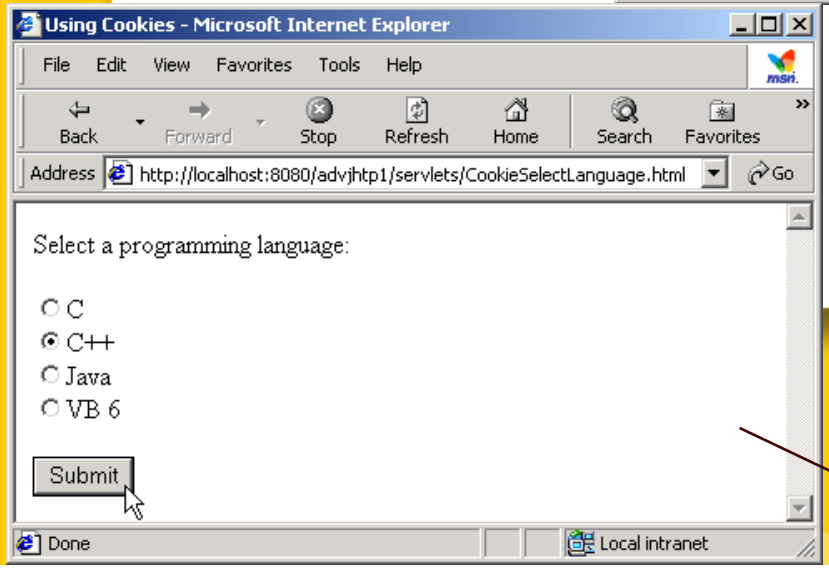
  - gestione della sessione di navigazione attraverso il mantenimento di strutture dati persistenti (dal lato del client, ovvero cookie)



L'uso del radio button obbliga l'utente ad effettuare una sola scelta alla volta!

POST

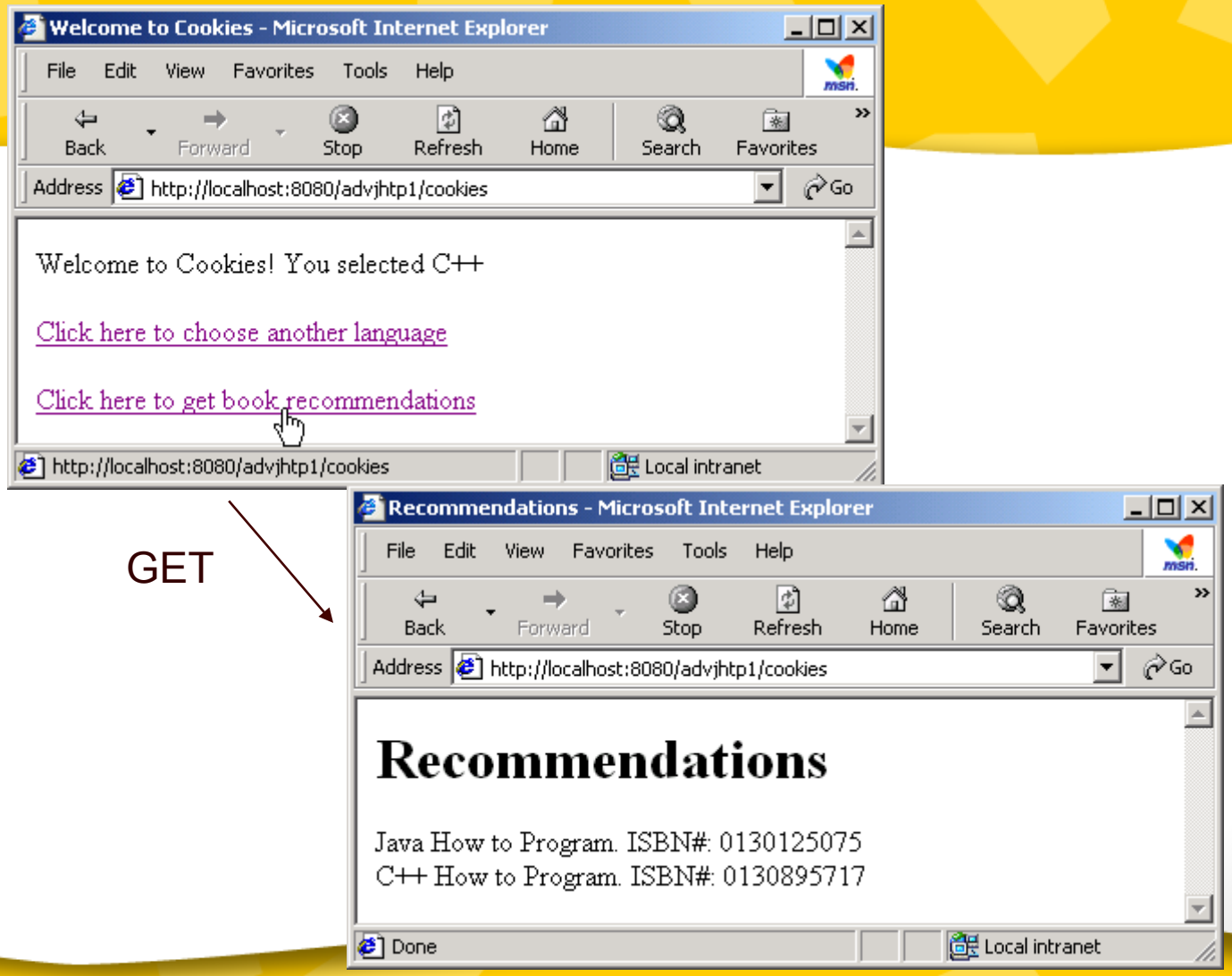




Cosa vogliamo ottenere?

POST

Cosa vogliamo ottenere?



Inviamo le selezioni con una richiesta di POST e le preleviamo con richieste di GET

```
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
  <title>Using Cookies</title>
</head>

<body>
  <form action = "/DirectoryDiSaluto/cookies" method = "post">

    <p>Select a programming language:</p>
    <p>
      <input type = "radio" name = "language"
        value = "C" />C <br />

      <input type = "radio" name = "language"
        value = "C++" />C++ <br />

      <!-- this radio button checked by default -->
      <input type = "radio" name = "language"
        value = "Java" checked = "checked" />Java<br />

      <input type = "radio" name = "language"
        value = "VB6" />VB 6
    </p>

    <p><input type = "submit" value = "Submit" /></p>

  </form>
</body>
</html>
```

```
1  CookieServlet.java
2  // Using cookies to store data on the client computer.
3
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8  import java.util.*;
9
10 public class CookieServlet extends HttpServlet {
11    private final Map books = new HashMap();
12
13    // initialize Map books
14    public void init()
15    {
16       books.put( "C", "0130895725" );
17       books.put( "C++", "0130895717" );
18       books.put( "Java", "0130125075" );
19       books.put( "VB6", "0134569555" );
20    }
21
22    // receive language selection and send cookie containing
23    // recommended book to the client
24    protected void doPost( HttpServletRequest request,
25       HttpServletResponse response )
26          throws ServletException, IOException
27    {
28       String language = request.getParameter( "language" );
29       String isbn = books.get( language ).toString();
30       Cookie cookie = new Cookie( language, isbn );
31
32       response.addCookie( cookie );  // must precede getWriter
33       response.setContentType( "text/html" );
34       PrintWriter out = response.getWriter();
35
```

Chiave

Valore

Il metodo **init** popola la collezione books con 4 coppie chiave/ valore.

Uso il metodo **getParameter** per ottenere il valore selezionato dall'utente

Ricerco nella collezione books l'elemento corrispondente al linguaggio scelto, per ottenere l'ISBN

Creo un nuovo oggetto **Cookie**, usando le due stringhe di testo **language** e **isbn** come nome e valore rispettivamente

Invio il cookie al client attraverso l'oggetto **response** con il metodo **addCookie**

```
36      // send XHTML page to client
37
38      // start XHTML document
39      out.println( "<?xml version = \"1.0\"?>" );
40
41      out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
42        "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
43        "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
44
45      out.println(
46        "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
47
48      // head section of document
49      out.println( "<head>" );
50      out.println( "<title>Welcome to Cookies</title>" );
51      out.println( "</head>" );
52
53      // body section of document
54      out.println( "<body>" );
55      out.println( "<p>Welcome to Cookies! You selected " +
56        language + "</p>" );
57
58      out.println( "<p><a href = " +
59        "\"/DirectoryDiSaluto/CookieSelectLanguage.html\">" +
60        "Click here to choose another language</a></p>" );
61
62      out.println( "<p><a href = \"/DirectoryDiSaluto/cookies\">" +
63        "Click here to get book recommendations</a></p>" );
64      out.println( "</body>" );
65
66      // end XHTML document
67      out.println( "</html>" );
68      out.close();    // close stream
69   }
70
```

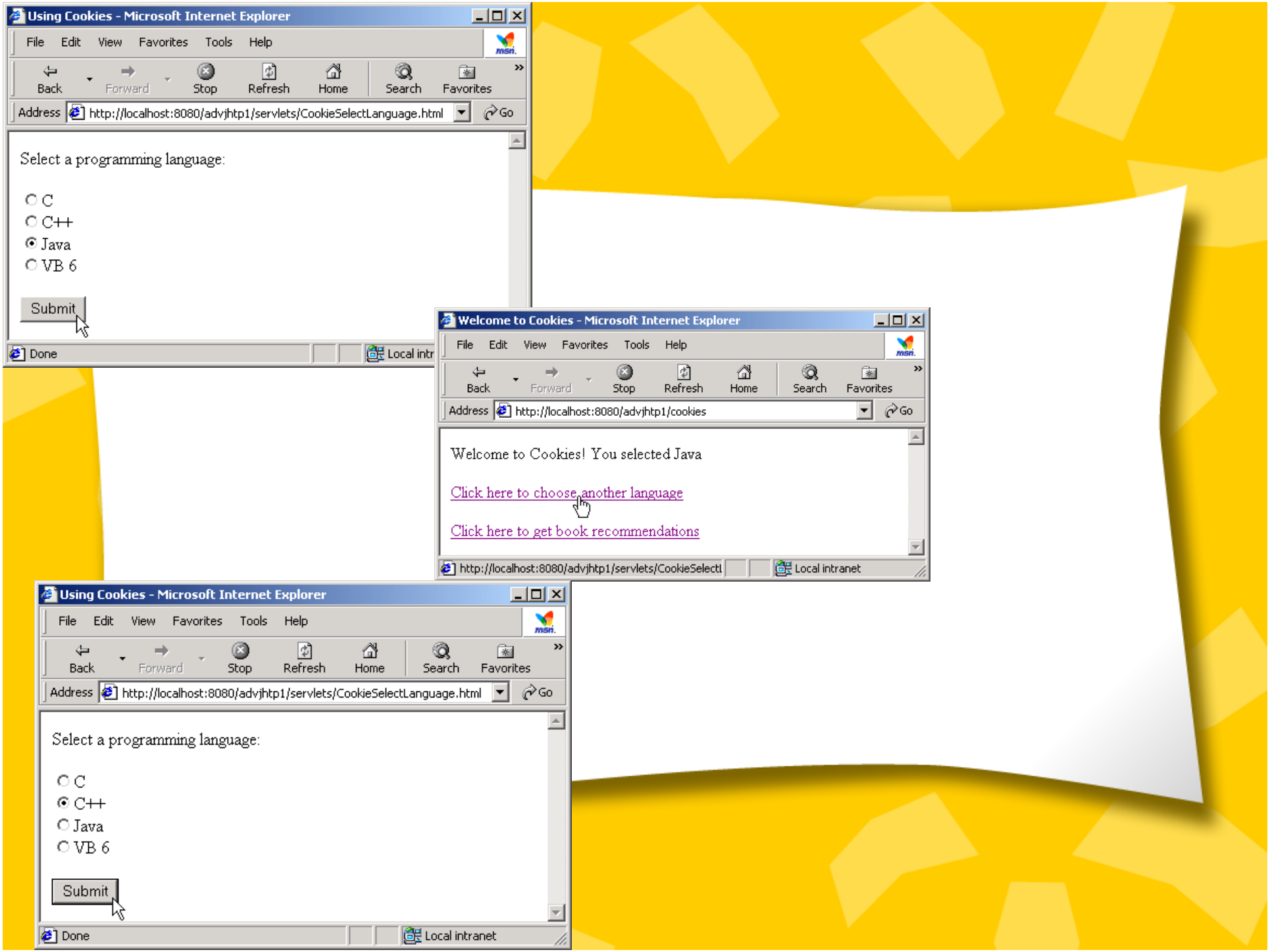
```
71  // read cookies from client and create XHTML document
72  // containing recommended books
73  protected void doGet( HttpServletRequest request,
74     HttpServletResponse response )
75        throws ServletException, IOException
76  {
77     Cookie cookies[] = request.getCookies();  // get cookies
78
79     response.setContentType( "text/html" );
80     PrintWriter out = response.getWriter();
81
82     // start XHTML document
83     out.println( "<?xml version = \"1.0\"?>" );
84
85     out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
86        "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
87        "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
88
89     out.println(
90        "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
91
92     // head section of document
93     out.println( "<head>" );
94     out.println( "<title>Recommendations</title>" );
95     out.println( "</head>" );
96
97     // body section of document
98     out.println( "<body>" );
99
100    // if there are any cookies, recommend a book for each ISBN
101    if ( cookies != null && cookies.length != 0 ) {
102       out.println( "<h1>Recommendations</h1>" );
103       out.println( "<p>" );
104
```

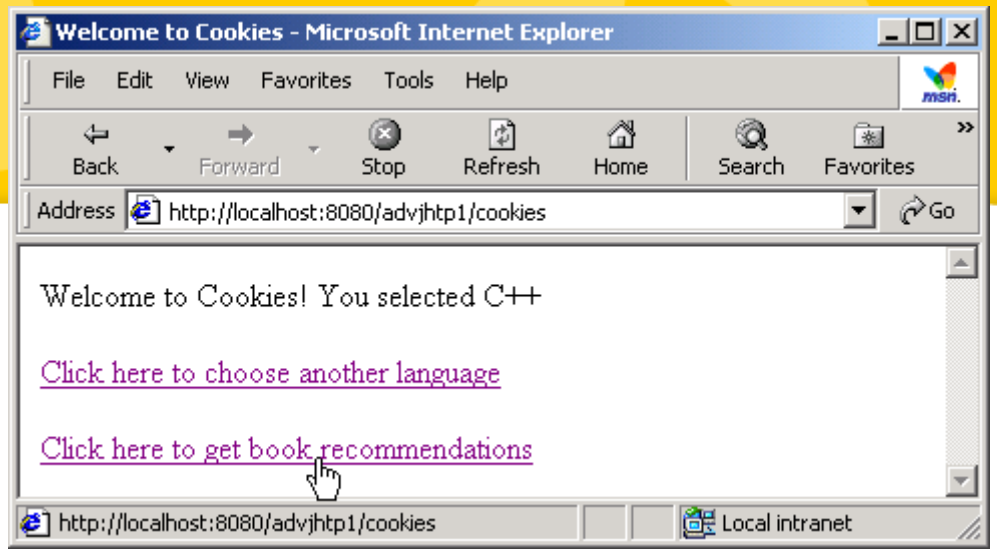
Legge i cookie presenti sul client utilizzando il metodo **getCookies** dell'oggetto **Request**, che restituisce un array di oggetti **Cookie**

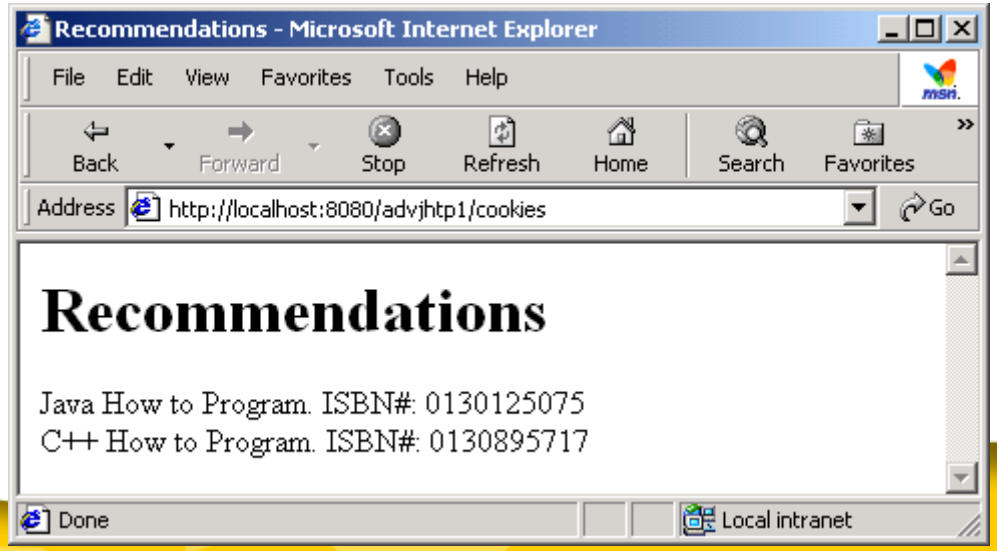
Nota:

Aggiungiamo i cookie all'oggetto risposta e li preleviamo dall'oggetto richiesta

```
105         // get the name of each cookie
106         for ( int i = 0; i < cookies.length; i++ )
107            out.println( cookies[ i ].getName() +
108               " How to Program. ISBN#: " +
109               cookies[ i ].getValue() + "<br />" );
110
111         out.println( "</p>" );
112      }
113      else {   // there were no cookies
114         out.println( "<h1>No Recommendations</h1>" );
115         out.println( "<p>You did not select a language.</p>" );
116      }
117
118      out.println( "</body>" );
119
120      // end XHTML document
121      out.println( "</html>" );
122      out.close();    // close stream
123   }
124 }
```


Using Cookies - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/advjhtp1/servlets/CookieSelectLanguage.html
Go
Select a programming language:
C
C++
Java
VB 6
Submit
Done
Local intr
Welcome to Cookies - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/advjhtp1/cookies
Go
Welcome to Cookies! You selected Java
Click here to choose another language
Click here to get book recommendations
http://localhost:8080/advjhtp1/servlets/CookieSelectl
Local intranet
Using Cookies - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/advjhtp1/servlets/CookieSelectLanguage.html
Go
Select a programming language:
C
C++
Java
VB 6
Submit
Done
Local intranet


Welcome to Cookies - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/advjhtp1/cookies
Go
Welcome to Cookies! You selected C++
Click here to choose another language
Click here to get book recommendations
http://localhost:8080/advjhtp1/cookies
Local intranet


Recommendations - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/advjhtp1/cookies
Go
Recommendations
Java How to Program. ISBN#: 0130125075
C++ How to Program. ISBN#: 0130895717
Done
Local intranet

I cookie (modifica del file web.xml)

Descriptor element	Value
servlet *element*	
servlet-name	**cookies** (ma va bene anche **"Paperino"**)
description	Using cookies to maintain state information.
servlet-class	**CookieServlet**
servlet-mapping *element*	
servlet-name	**cookies** (ma va bene anche **"Paperino"**)
url-pattern	**/cookies**

I cookie (Cont.)

Method	Description
`getComment()`	Returns a **`String`** describing the purpose of the cookie (**`null`** if no comment has been set with **`setComment`**).
`getDomain()`	Returns a **`String`** containing the cookie's domain. This determines which servers can receive the cookie. By default, cookies are sent to the server that originally sent the cookie to the client.
`getMaxAge()`	Returns an **`int`** representing the maximum age of the cookie in seconds.
`getName()`	Returns a **`String`** containing the name of the cookie as set by the constructor.
`getPath()`	Returns a **`String`** containing the URL prefix for the cookie. Cookies can be "targeted" to specific URLs that include directories on the Web server. By default, a cookie is returned to services operating in the same directory as the service that sent the cookie or a subdirectory of that directory.
`getSecure()`	Returns a **`boolean`** value indicating if the cookie should be transmitted using a secure protocol (**`true`**).
`getValue()`	Returns a **`String`** containing the value of the cookie as set with **`setValue`** or the constructor.
`getVersion()`	Returns an **`int`** containing the version of the cookie protocol used to create the cookie. A value of 0 (the default) indicates the original cookie protocol as defined by Netscape. A value of 1 indicates the current version, which is based on *Request for Comments (RFC) 2109*.

I cookie (Cont.)

`setComment( String )`	The comment describing the purpose of the cookie that is presented by the browser to the user. (Some browsers allow the user to accept cookies on a per -cookie basis.)
`setDomain(String )`	This determines which servers can receive the cookie. By default, cookies are sent to the server that originally sent the cookie to the client. The domain is specified in the form `".deitel.com"` , indicating that all servers ending with `.deitel.com` can receive this cookie.
`setMaxAge(int )`	Sets the max imum age of the cookie in seconds.
`setPath( String )`	Sets the "target" URL prefix indicating the directories on the server that lead to the services that can receive this cookie.
`setSecure(boolean )`	A `true` value indicates that the cookie should only be sent using a secure protocol.
`setValue(String )`	Sets the value of a cookie.
`setVersion( int)`	Sets the cookie protocol for this cookie.

Argomenti prossime lezioni

1. Ricapitolazione sulla redirezione tramite metodo sendRedirect dell'interfaccia HttpServletResponse
2. Concetto di sessione di navigazione
3. Uso di cookie per la gestione di una sessione
4. Uso dell'interfaccia HttpSession
5. URL rewriting nella gestione della sessione
6. Redirezione tramite il metodo forward dell'interfaccia RequestDispatcher

Gestire la sessione con oggetti persistenti sul server (`HttpSession`)

- Interfaccia **`HttpSession`**
- Trovate informazioni all'indirizzo:

  http://127.0.0.1:8080/tomcat-docs/servletapi/index.html

- Un oggetto che implementi l'interfaccia HttpSession identifica un utente/browser attraverso diverse richieste e permette di memorizzare informazioni acquisite durante la sessione di navigazione.
- Tale oggetto rappresenta la sessione:
  - Persiste per uno specifico periodo di tempo.
  - Corrisponde ad un solo utente, che può visitare il sito anche più volte.

Perché usare HttpSession

- Non potremmo memorizzare le informazioni che ci servono direttamente attraverso gli attributi della servlet?

- Più client possono accedere alla stessa servlet in parallelo!

Metodi da conoscere per usare le sessioni (1/3)

- Metodi dell'oggetto HttpServletRequest
  - HttpSession **getSession** (boolean **create**)
    Restituisce l'oggetto HttpSession associato con la richiesta. Se non esiste ancora nessun oggetto HttpSession, ne viene creato uno nel caso **create** valga **true**.
- Metodi dell'oggetto HttpSession
  - public void **setAttribute** (java.lang.String name, **java.lang.Object value**)
    - Associa un oggetto alla presente sessione, utilizzando il nome specificato. Se esiste un collegamento con un oggetto con lo stesso nome, questo viene rimpiazzato.
  - java.lang.Object **getAttribute**(java.lang.String name)
    - Restituisce l'oggetto associato con il nome dato in input, oppure null se non esiste nessun oggetto con quel nome

Metodi da conoscere per usare le sessioni (2/3)

- Metodi dell'oggetto HttpSession
  - java.lang.String **getId**()
    Restituisce una stringa contenente l'identificativo unico della sessione
  - boolean **isNew**()
    Restituisce true se la sessione con il client è stata creata in seguito alla richiesta corrente
  - long **getCreationTime**()
    Restituisce l'istante di creazione della sessione misurato in millisecondi a partire dalle 00:00 del 1 Gennaio 1970, GMT
  - long **getLastAccessedTime**()
    Restituisce l'istante dell'ultima richiesta associata alla sessione misurato in millisecondi a partire dalle 00:00 del 1 Gennaio 1970, GMT

Metodi da conoscere per usare le sessioni (3/3)

- … metodi dell'oggetto HttpSession
  - int **getMaxInactiveInterval**()
    Restituisce il massimo intervallo di tempo, in secondi, in cui il container può mantenere la sessione aperta, tra due accessi consecutivi da parte del client.
  - java.util.Enumeration **getAttributeNames**()
    Restituisce un oggetto Enumeration (un elenco) di stringhe contenenti i nomi di tutti gli oggetti associati alla sessione

Esercizio: uso di HttpSession per memorizzare le scelte dell'utente

- Memorizziamo in un oggetto che implementa l'interfaccia **`HttpSession`** le informazioni che nell'esercizio precedente mettevamo nei cookie.
  - Possiamo memorizzare coppie nome – valore

    dove il valore può essere un oggetto (non solo una stringa come nei cookie)
- Servlet **`SessionServlet`**
  - Usa l'oggetto **`HttpSession`**
  - Gestisce sia richieste di **`GET`** che richieste di **`POST`**

E’ utile sapere per svolgere questo esercizio… (cont.)

- Interface `Enumeration`
  - Consente l’elencazione di una serie di elementi, uno alla volta
  - Due metodi:
    - java.lang.Object nextElement()

      restituisce il prossimo elemento dell’elenco
    - boolean hasMoreElements()

      restituisce true se l’elenco contiene ancora elementi

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- SessionSelectLanguage.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9    <title>Using Sessions</title>
10 </head>
11
12 <body>
13   <form action = "/DDS/sessions" method = "post">
14
15     <p>Select a programming language:</p>
16     <p>
17       <input type = "radio" name = "language"
18         value = "C" />C <br />
19
20       <input type = "radio" name = "language"
21         value = "C++" />C++ <br />
22
23       <!-- this radio button checked by default -->
24       <input type = "radio" name = "language"
25         value = "Java" checked = "checked" />Java<br />
26
27       <input type = "radio" name = "language"
28         value = "VB6" />VB 6
29     </p>
30
31     <p><input type = "submit" value = "Submit" /></p>
32
33   </form>
34 </body>
35 </html>
```

Stesso form dell'esercizio sui cookie

```
1  // SessionServlet.java
2  // Using HttpSession to maintain client state information.
3
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8  import java.util.*;
9
10 public class SessionServlet extends HttpServlet {
11    private final Map books = new HashMap();
12
13    // initialize Map books
14    public void init()
15    {
16       books.put( "C", "0130895725" );
17       books.put( "C++", "0130895717" );
18       books.put( "Java", "0130125075" );
19       books.put( "VB6", "0134569555" );
20    }
21
22    // receive language selection and create HttpSession object
23    // containing recommended book for the client
24    protected void doPost( HttpServletRequest request,
25       HttpServletResponse response )
26          throws ServletException, IOException
27    {
28       String language = request.getParameter( "language" );
29
30       // Get the user's session object.
31       // Create a session (true) if one does not exist.
32       HttpSession session = request.getSession( true );
33
34       // add a value for user's choice to session
35       session.setAttribute( language, books.get( language ) );
```

Chiave

Valore

Usa il metodo **getSession** dell'interfaccia **HttpServletRequest** per ottenere l'oggetto **HttpSession** o crearlo (true)

Usa **setAttribute** per inserire il nome del linguaggio **language** e il corrispondente numero ISBN nell'oggetto **HttpSession** object.

```
36
37    response.setContentType( "text/html" );
38    PrintWriter out = response.getWriter();
39
40    // send XHTML page to client
41
42    // start XHTML document
43    out.println( "<?xml version = \"1.0\"?>" );
44
45    out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
46       "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
47       "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
48
49    out.println(
50       "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
51
52    // head section of document
53    out.println( "<head>" );
54    out.println( "<title>Welcome to Sessions</title>" );
55    out.println( "</head>" );
56
57    // body section of document
58    out.println( "<body>" );
59    out.println( "<p>Welcome to Sessions! You selected " +
60       language + ".</p>" );
61
62    // display information about the session
63    out.println( "<p>Your unique session ID is: " +
64       session.getId() + "<br />" );
65
66    out.println(
67       "This " + ( session.isNew() ? "is" : "is not" ) +
68       " a new session<br />" );
69
```

Usa il metodo **getID** di **HttpSession** per conoscere l'ID di sessione.

Decide se si tratta di una nuova sessione usando il metodo **isNew**.

```
70      out.println( "The session was created at: " +
71        new Date( session.getCreationTime() ) + "<br />" );
72
73      out.println( "You last accessed the session at: " +
74        new Date( session.getLastAccessedTime() ) + "<br />" );
75
76      out.println( "The maximum inactive interval is: " +
77        session.getMaxInactiveInterval() + " seconds</p>" );
78
79      out.println( "<p><a href = " +
80        "\"servlets/SessionSelectLanguage.html\">" +
81        "Click here to choose another language</a></p>" );
82
83      out.println( "<p><a href = \"sessions\">" +
84        "Click here to get book recommendations</a></p>" );
85      out.println( "</body>" );
86
87      // end XHTML document
88      out.println( "</html>" );
89      out.close();    // close stream
90   }
91
92   // read session attributes and create XHTML document
93   // containing recommended books
94   protected void doGet( HttpServletRequest request,
95      HttpServletResponse response )
96         throws ServletException, IOException
97   {
98      // Get the user's session object.
99      // Do not create a session (false) if one does not exist.
100     HttpSession session = request.getSession( false );
101
102     // get names of session object's values
103     Enumeration valueNames;
104
```

Usa il metodo **getMaxInactiveInterval** per conoscere il massimo intervallo di tempo in cui la sessione può restare inattiva prima che il container la scarti.

ATTENZIONE all'uso dei percorsi relativi!!!

Il percorso di partenza è quello definito nell'url pattern e usato per invocare la servlet.

Provare a cambiare l'url pattern, dovrete ridefinire anche questi link

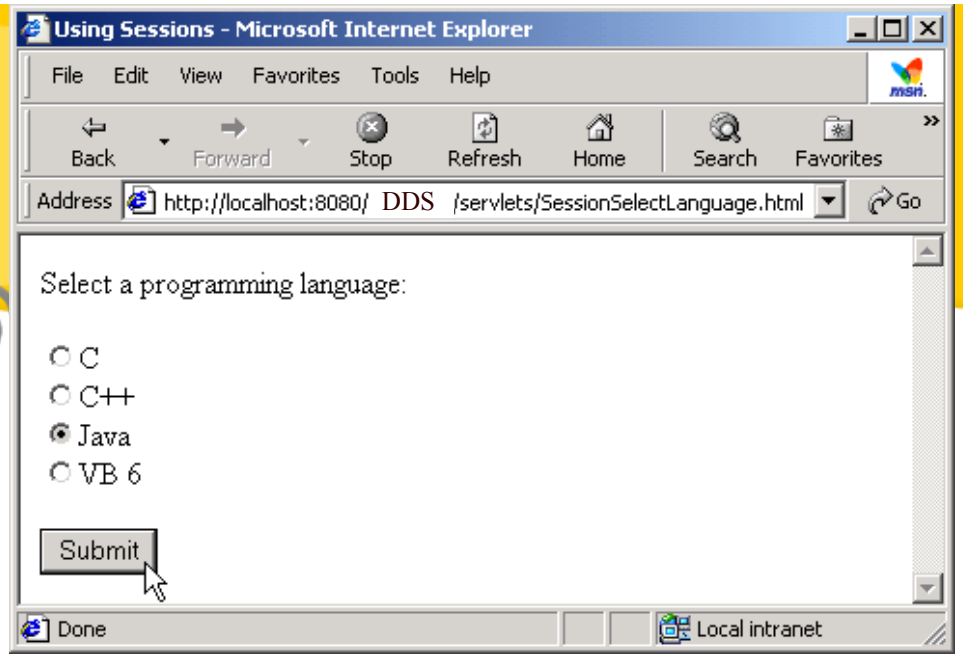
Ottiene l'oggetto **HttpSession** correlato alla sessione corrente, attraverso il metodo **getSession**.

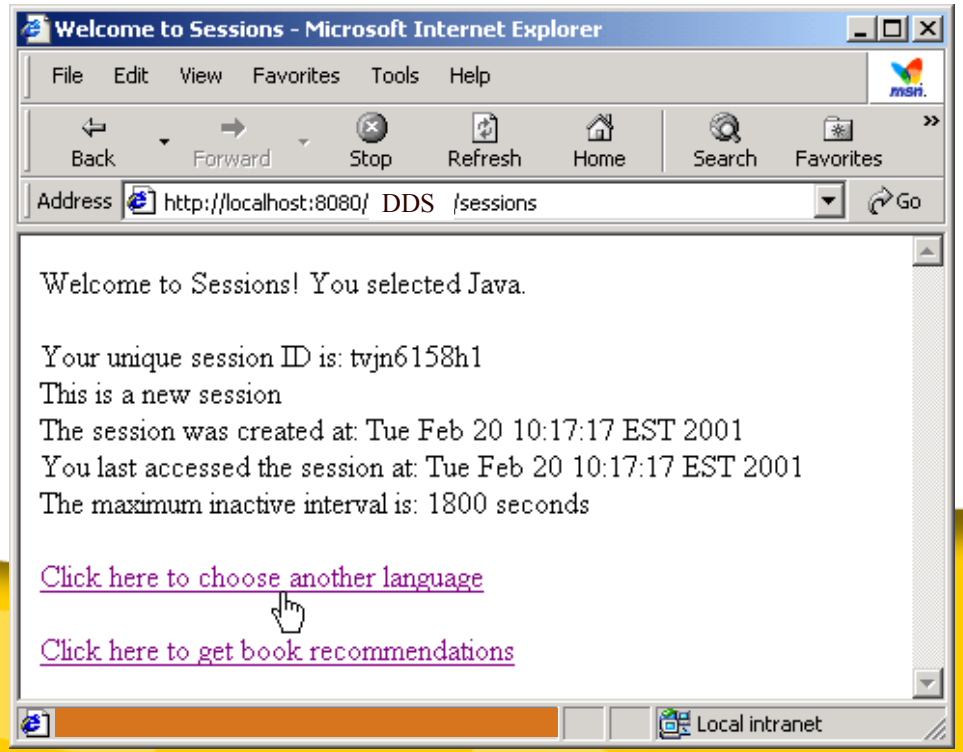
```
105     if ( session != null )
106        valueNames = session.getAttributeNames();
107     else
108        valueNames = null;
109
110     PrintWriter out = response.getWriter();
111     response.setContentType( "text/html" );
112
113     // start XHTML document
114     out.println( "<?xml version = \"1.0\"?>" );
115
116     out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
117        "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
118        "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
119
120     out.println(
121        "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
122
123     // head section of document
124     out.println( "<head>" );
125     out.println( "<title>Recommendations</title>" );
126     out.println( "</head>" );
127
128     // body section of document
129     out.println( "<body>" );
130
131     if ( valueNames != null &&
132          valueNames.hasMoreElements() ) {
133        out.println( "<h1>Recommendations</h1>" );
134        out.println( "<p>" );
135
136        String name, value;
137
```

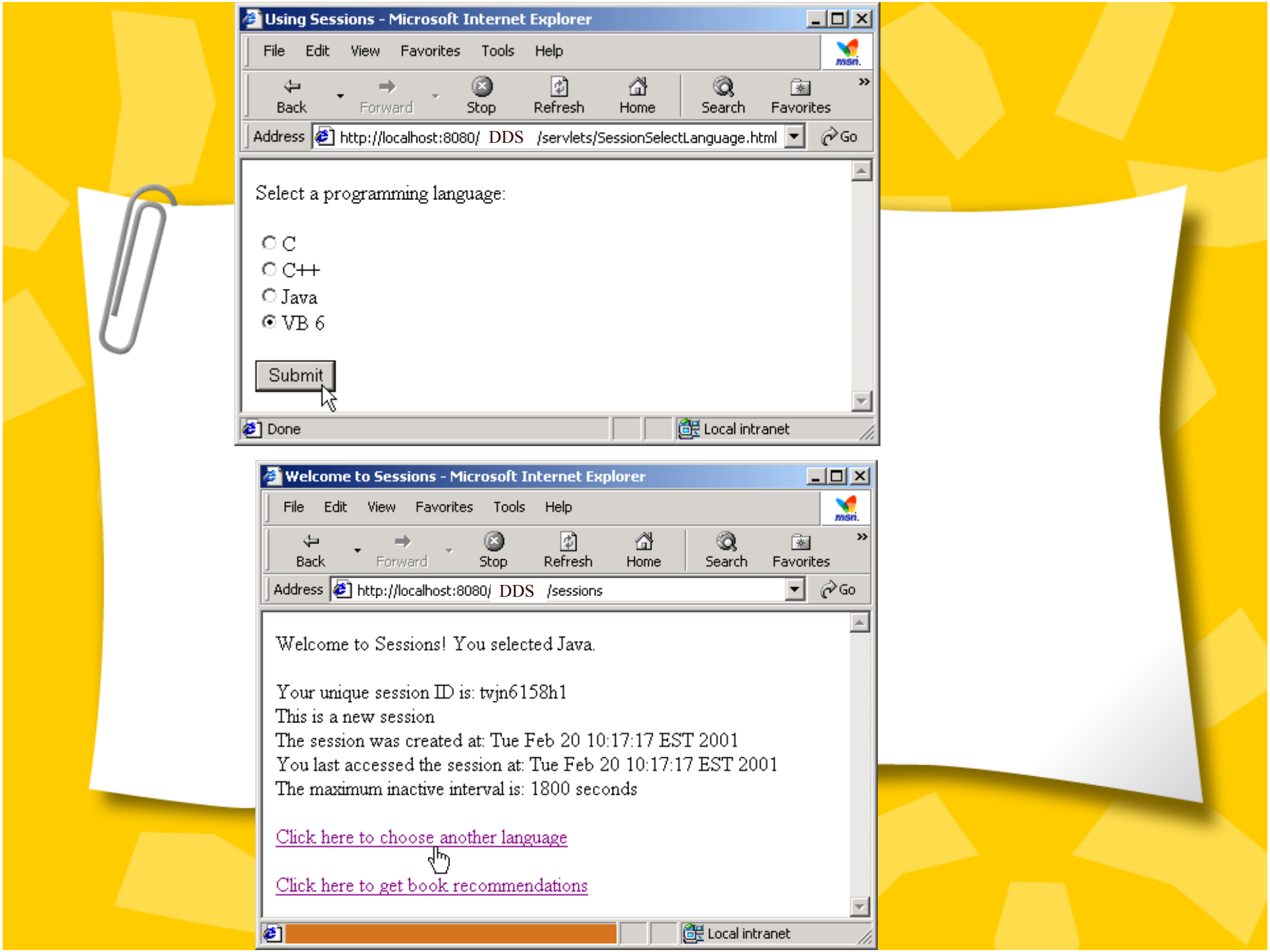
Uses **HttpSession** method **getAttributeNames** to retrieve an **Enumeration** of the attribute names.

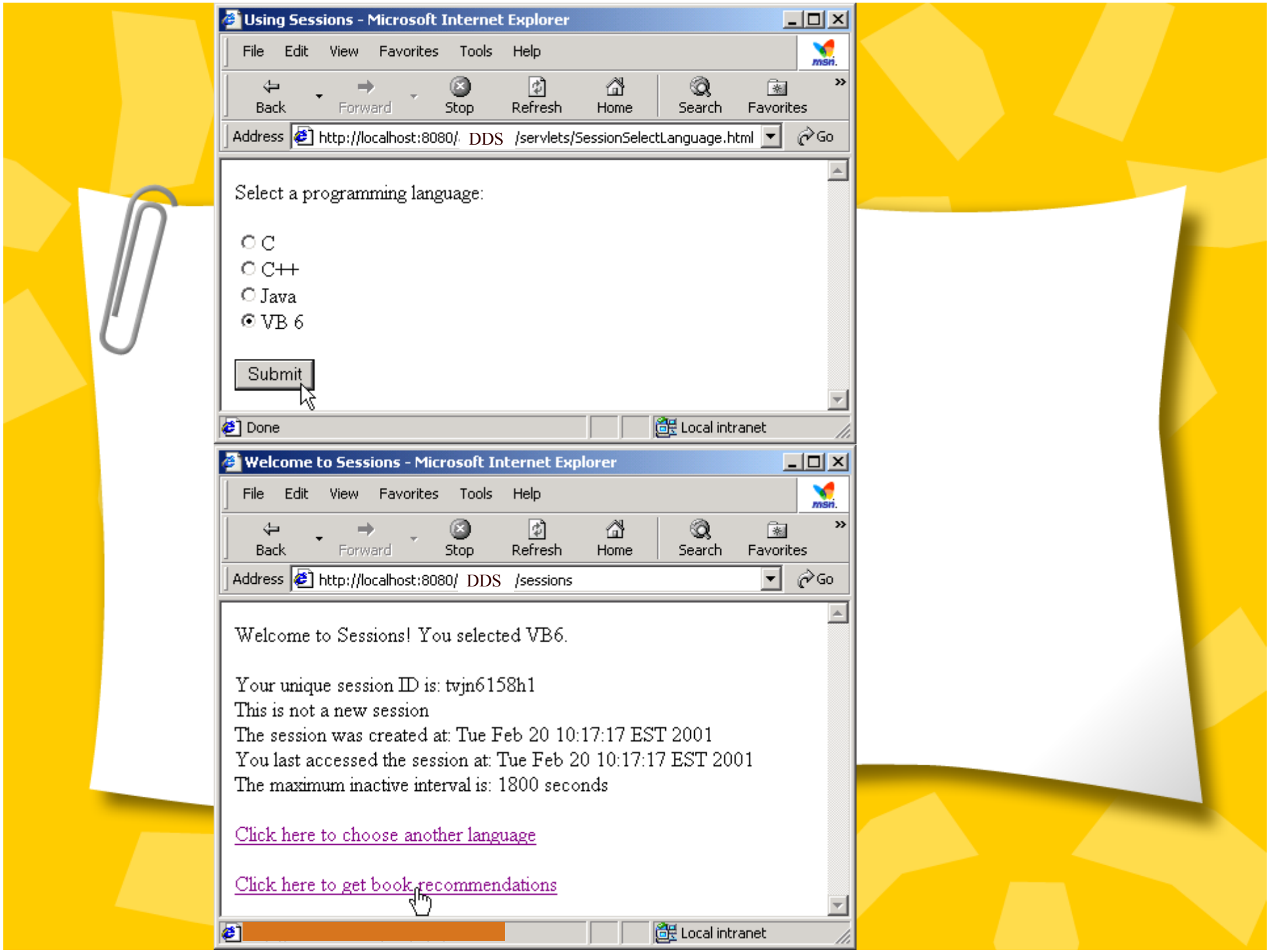
```
138         // get value for each name in valueNames
139         while ( valueNames.hasMoreElements() ) {
140            name = valueNames.nextElement().toString();
141            value = session.getAttribute( name ).toString();
142
143            out.println( name + " How to Program. " +
144               "ISBN#: " + value + "<br />" );
145         }
146
147         out.println( "</p>" );
148      }
149      else {
150         out.println( "<h1>No Recommendations</h1>" );
151         out.println( "<p>You did not select a language.</p>" );
152      }
153
154      out.println( "</body>" );
155
156      // end XHTML document
157      out.println( "</html>" );
158      out.close();    // close stream
159   }
160 }
```

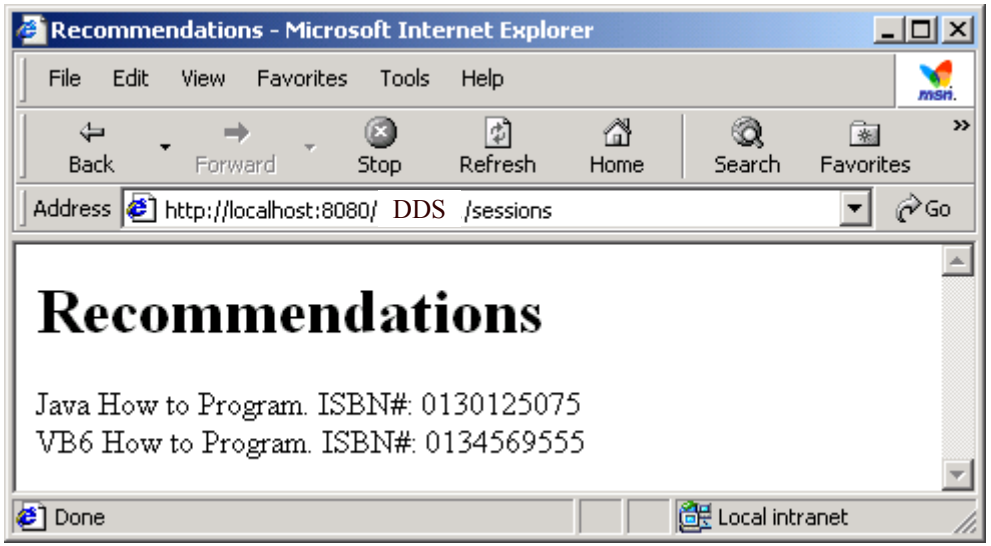
Invokes method **getAttribute** of **HttpSession** to retrieve the ISBN of a book from the **HttpSession** object.


Using Sessions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/ DDS /servlets/SessionSelectLanguage.html
Go
Select a programming language:
C
C++
Java
VB 6
Submit
Done
Local intranet


Welcome to Sessions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/ DDS /sessions
Go
Welcome to Sessions! You selected Java.
Your unique session ID is: tvjn6158h1
This is a new session
The session was created at: Tue Feb 20 10:17:17 EST 2001
You last accessed the session at: Tue Feb 20 10:17:17 EST 2001
The maximum inactive interval is: 1800 seconds
Click here to choose another language
Click here to get book recommendations
Local intranet


Using Sessions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/ DDS /servlets/SessionSelectLanguage.html
Go
Select a programming language:
C
C++
Java
VB 6
Submit
Done
Local intranet
Welcome to Sessions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/ DDS /sessions
Go
Welcome to Sessions! You selected Java.
Your unique session ID is: tvjn6158h1
This is a new session
The session was created at: Tue Feb 20 10:17:17 EST 2001
You last accessed the session at: Tue Feb 20 10:17:17 EST 2001
The maximum inactive interval is: 1800 seconds
Click here to choose another language
Click here to get book recommendations
Local intranet


Using Sessions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/ DDS /servlets/SessionSelectLanguage.html
Go
Select a programming language:
C
C++
Java
VB 6
Submit
Done
Local intranet
Welcome to Sessions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/ DDS /sessions
Go
Welcome to Sessions! You selected VB6.
Your unique session ID is: twjn6158h1
This is not a new session
The session was created at: Tue Feb 20 10:17:17 EST 2001
You last accessed the session at: Tue Feb 20 10:17:17 EST 2001
The maximum inactive interval is: 1800 seconds
Click here to choose another language
Click here to get book recommendations
Local intranet


Recommendations - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/ DDS /sessions
Go
Recommendations
Java How to Program. ISBN#: 0130125075
VB6 How to Program. ISBN#: 0134569555
Done
Local intranet

Session Tracking with `HttpSession` (Cont.)

Descriptor element	Value
servlet *element*	
servlet-name	**sessions**
description	Using sessions to maintain state information.
servlet-class	**SessionServlet**
servlet-mapping *element*	
servlet-name	**sessions**
url-pattern	**/sessions**

Chiusura della sessione di navigazione

- Mentre per i cookie non è previsto un metodo esplicito per la cancellazione dei cookie,
  - Prelevare il cookie
  - Configurare il suo tempo di vita a zero (metodo setMaxAge(int) della classe Cookie)
  - Inviare di nuovo il cookie al client

  la cancellazione della sessione deve fare uso del metodo invalidate() dell'interfaccia HttpSession

Riflessioni…

- Q: Come può il server riconoscere le richieste di una stessa sessione e associarle correttamente all'oggetto persistente con il contesto di visibilità voluto?

Riflessioni…

- Q: Come può il server riconoscere le richieste di una stessa sessione e associarle correttamente all'oggetto persistente con il contesto di visibilità voluto?
- R: Viene usato un cookie!!!

Un COOKIEEEEE ????

- Q1: che senso ha usare la sessione se poi questa dipende dal cookie?
- R1: i dati sensibili restano sul server, il cookie creato non è trasferibile da un computer ad un altro in quanto contiene un identificativo generato dinamicamente.

Riflessioni…

- Q2: se il browser ha i cookie disabilitati come faccio a gestire la sessione sul server?
- R2: Si deve fare in modo che tutti i percorsi utilizzati dal browser appendano all'url l'identificativo di sessione (sarà sempre il server a scrivere dinamicamente questi url con una tecnica detta di URL rewriting). Questo si ottiene con il metodo dell'interfaccia HttpServletResponse:

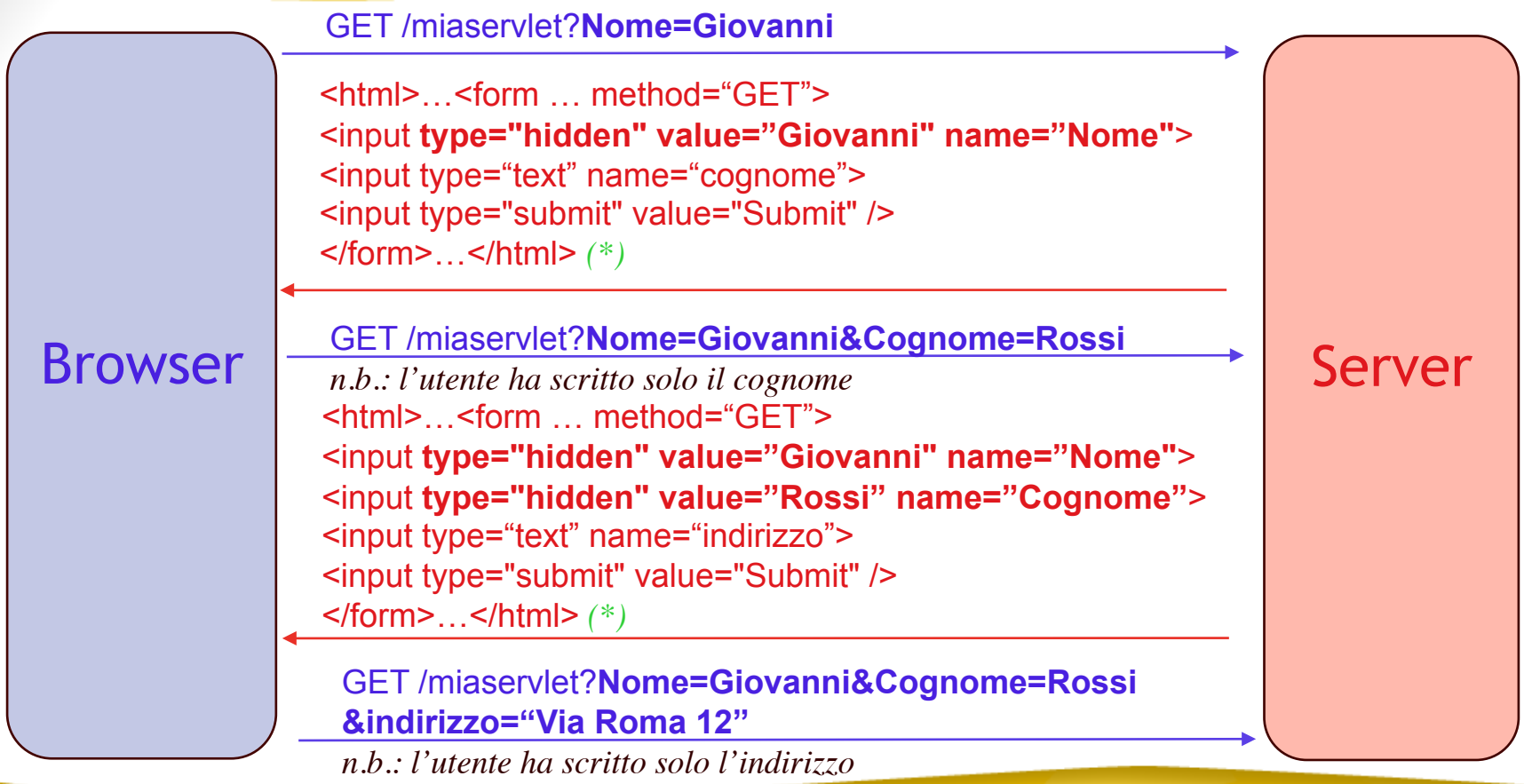
java.lang.String **encodeURL**(java.lang.String url)

Dalla documentazione:

Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.

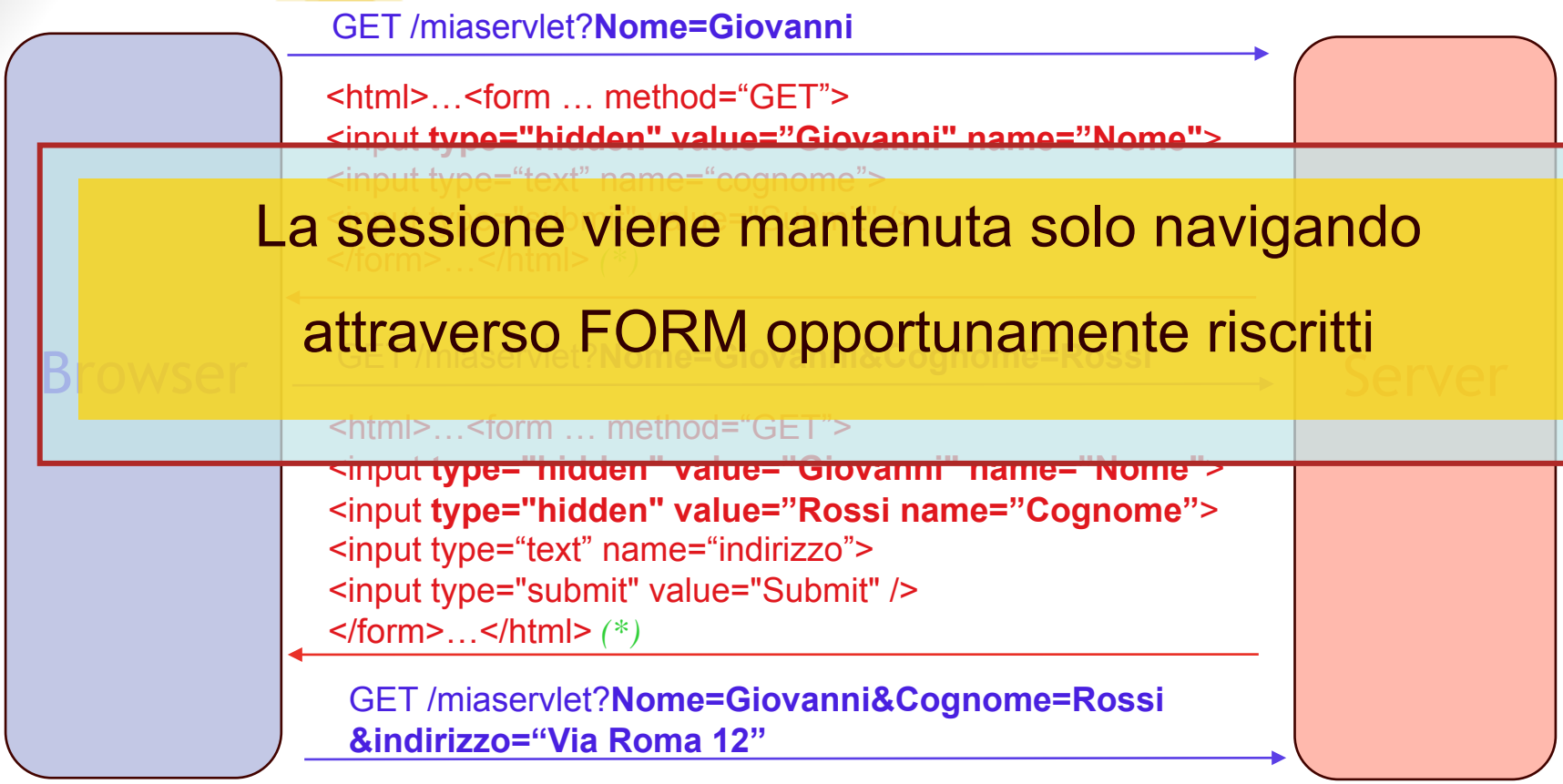
Riassunto: gestione della sessione tramite parametri hidden

Riassunto: gestione della sessione tramite parametri hidden



(*) *Il testo della pagina html viene generato dinamicamente dalla servlet in funzione dei parametri ricevuti nella richiesta, sia che fossero hidden sia che fossero visibili nel form*

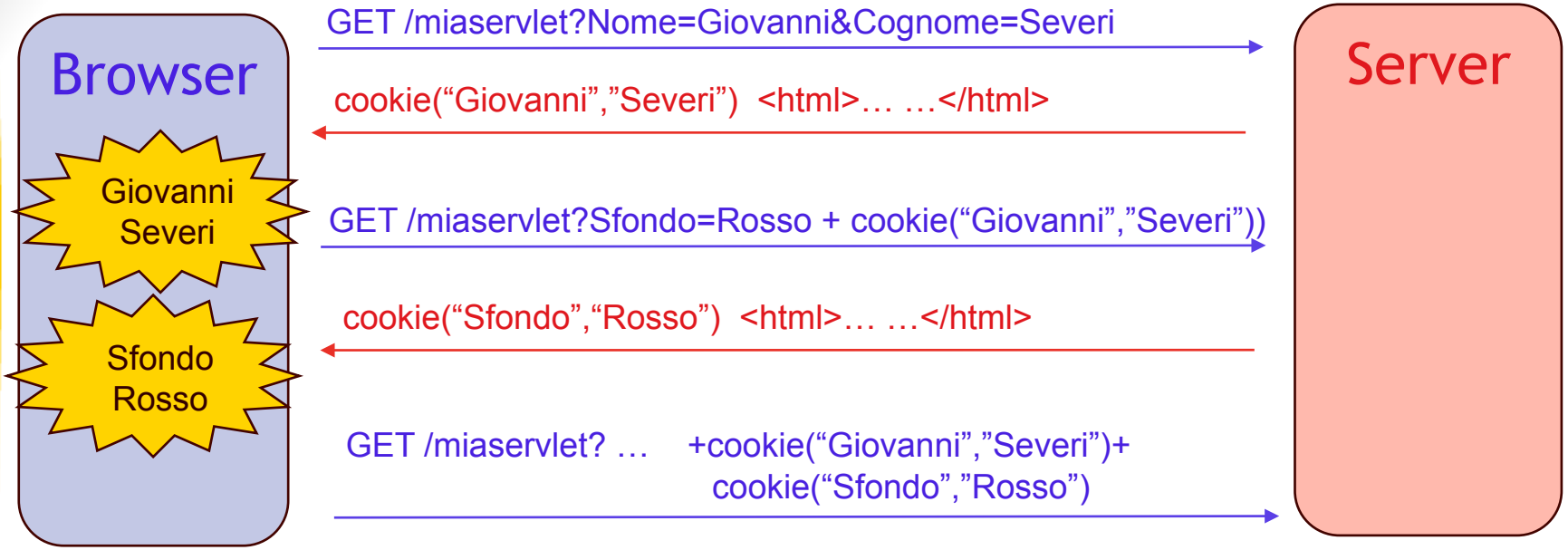
Riassunto: gestione della sessione tramite parametri hidden



(*) Il testo della pagina html viene generato dinamicamente dalla servlet in funzione dei parametri ricevuti nella richiesta, sia che fossero hidden sia che fossero visibili nel form

Riassunto: gestione della sessione tramite cookie

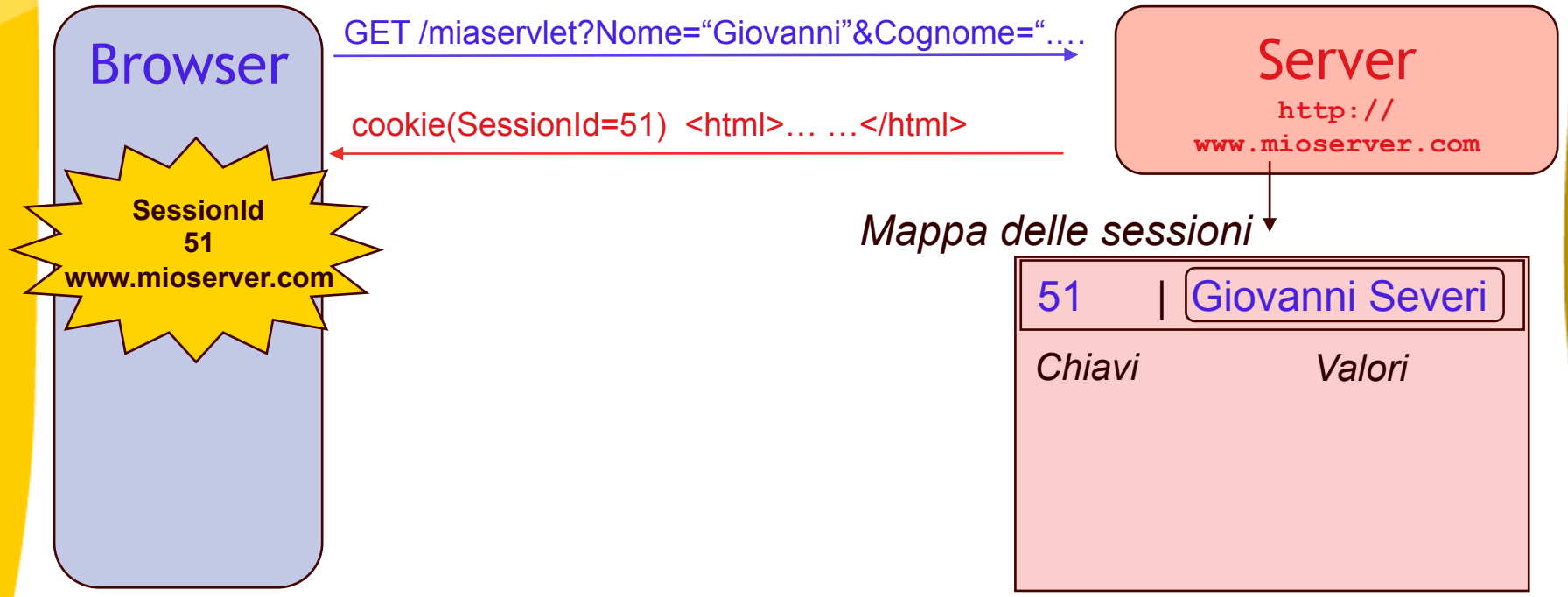
Riassunto: gestione della sessione tramite cookie



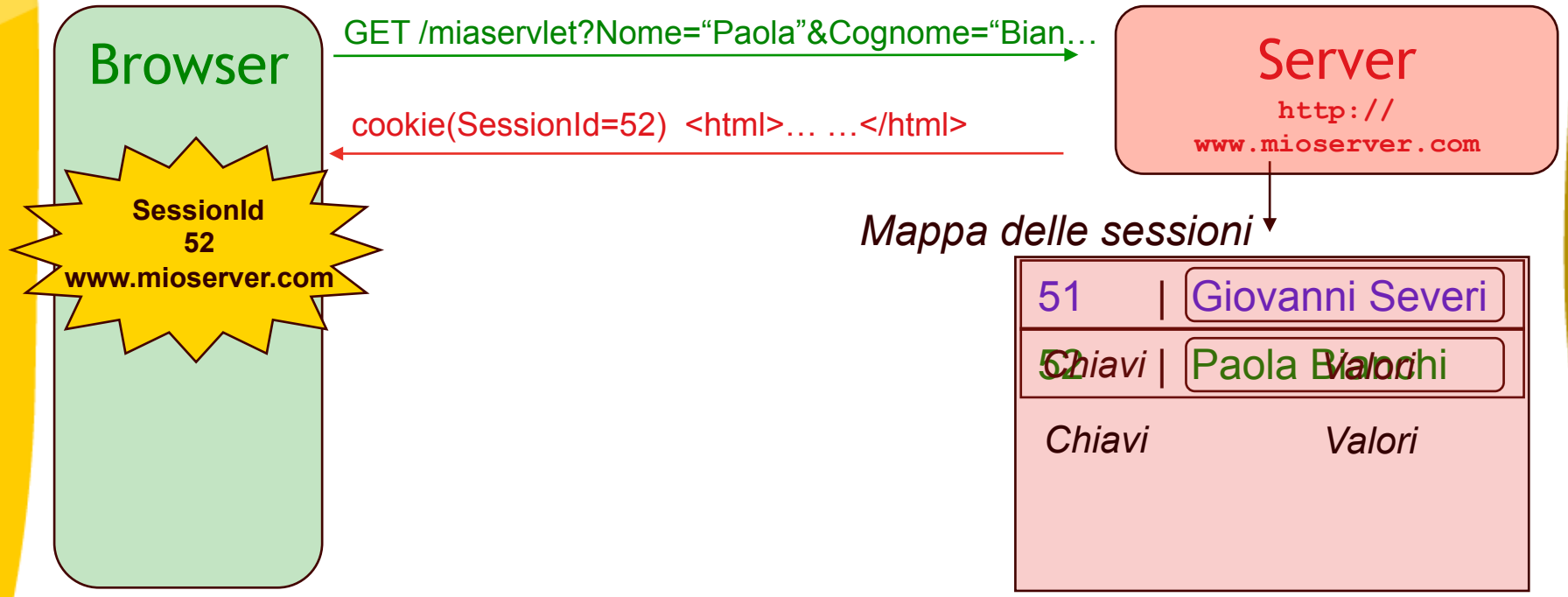
Gestione della sessione di navigazione tramite oggetti persistenti sul server

- Uso di oggetti che implementano l'interfaccia HttpSession

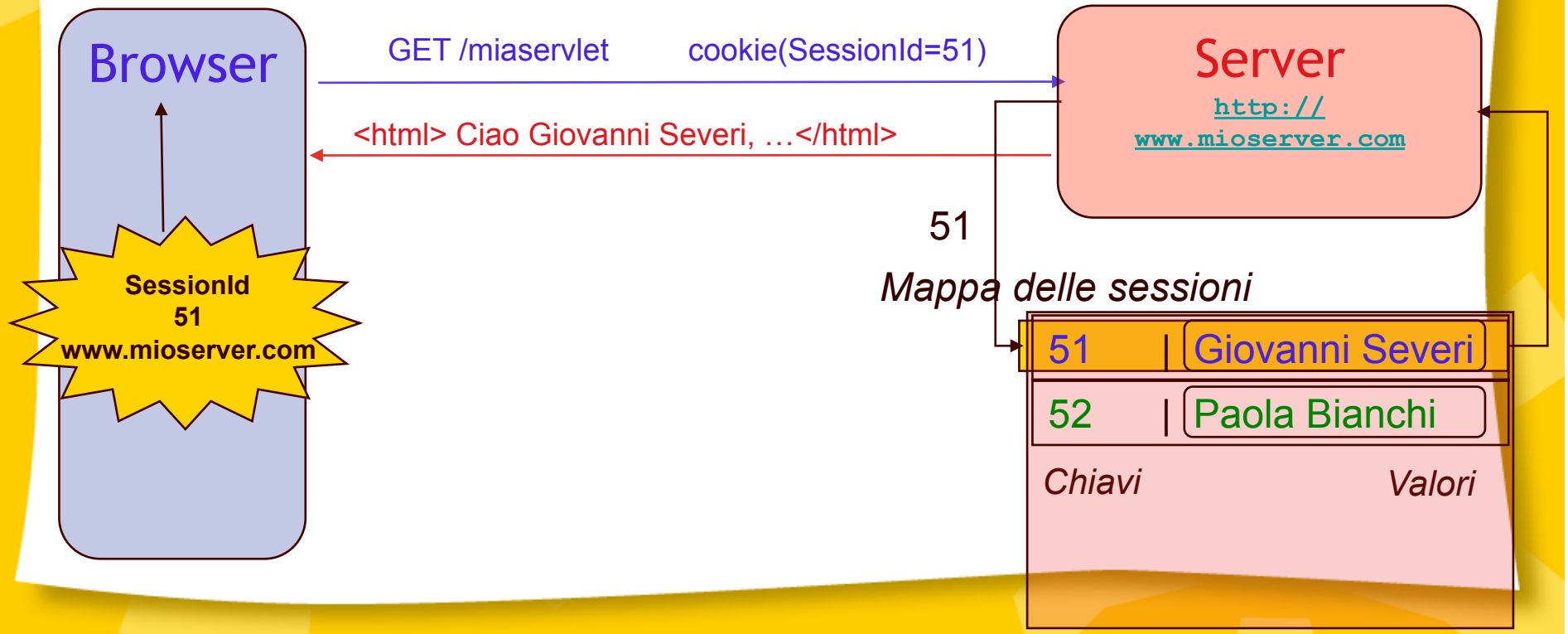
Gestione della sessione tramite oggetto persistente sul server (interfaccia **HttpSession**) (1/3)



Gestione della sessione tramite oggetto persistente sul server (interfaccia **HttpSession**) (2/3)



Gestione della sessione tramite oggetto persistente sul server (interfaccia **HttpSession**) (3/3)



Chiusura della sessione di navigazione

- Come detto, la cancellazione della sessione deve fare uso del metodo invalidate() dell'interfaccia HttpSession
- I cookie usati per la gestione della sessione sono cookie di sessione: scadono alla chiusura del browser

Uso dei cookie di sessione

- In pratica il server gestisce una **mappa**: ogni entry rappresenta una sessione di lavoro distinta
  - La chiave della mappa è un identificatore per la sessione
  - Il valore della mappa è un oggetto che contiene informazioni associate a quella specifica sessione
- In tutte le risposte al client il server aggiunge automaticamente e in maniera trasparente un cookie che contiene l'identificatore di sessione
- Attraverso questo identificatore, ad ogni richiesta il server è in grado di recuperare l'oggetto con le informazioni associate alla sessione

Supporto del container alla gestione della sessione di navigazione (1/2)

Tomcat (così come gli altri servlet container) mette a disposizione del programmatore una particolare infrastruttura e le API per una **gestione trasparente** di questo meccanismo:

- Nel processare una richiesta, il programmatore deve semplicemente chiedere all'oggetto `HttpRequest` l'oggetto associato alla sessione
- Il contenitore, in maniera trasparente, estrae dalla richiesta l'identificatore di sessione e lo usa per recuperare dalla mappa delle sessioni l'oggetto associato a quell'identificatore

Supporto del container alla gestione della sessione di navigazione (2/2)

- Se nella richiesta non viene trovato nessun identificatore allora, ove necessario (argomento true nel metodo getSession()), il container procede alla creazione della nuova sessione.
  - Vengono creati:
    - un nuovo identificatore,
    - un nuovo oggetto sessione,
    - una relativa entry nella mappa delle sessioni,
    - un cookie di sessione che viene agganciato alla risposta.
- Se il browser ha i cookie disabilitati come faccio a gestire la sessione sul server?

Argomenti prossime lezioni

1. Ricapitolazione sulla redirezione tramite metodo sendRedirect dell'interfaccia HttpServletResponse
2. Concetto di sessione di navigazione
3. Uso di cookie per la gestione di una sessione
4. Uso dell'interfaccia HttpSession
5. URL rewriting nella gestione della sessione
6. Redirezione tramite il metodo forward dell'interfaccia RequestDispatcher

URL REWRITING nella gestione delle sessioni

- Se il browser ha i cookie disabilitati come faccio a gestire la sessione sul server?
- Si deve fare in modo che **tutti i percorsi utilizzati dal browser appendano all'url l'identificativo di sessione** (tecnica detta di **URL rewriting**).
  - N.B. il programmatore non conosce l'ID di sessione che verrà usato

Per una gestione semplice e trasparente delle operazioni di URL rewriting si ricorre al metodo dell'interfaccia HttpServletResponse:

java.lang.String **encodeURL**(java.lang.String url)

Dalla documentazione:

Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.

URL rewriting

Si deve poter rispondere a due questioni fondamentali:

1. Quali sono le URL che l'utente utilizzerà in futuro?
2. Come possiamo costringere l'utente ad appendere a queste URL l'identificativo di sessione?

Non possiamo sapere con certezza che richieste verranno effettuate dall'utente, ma possiamo controllare quelle effettuate attraverso gli hyperlink presenti nella pagina.

URL rewriting

- Se il client rifiuta i cookie, è possibile chiedere al contenitore di appendere l'identificatore della sessione agli URL degli hyperlink presenti nel codice HTML della risposta
  - Pagine di risposta generate da sessioni diverse conterranno URL diversi (con diversi identificativi di sessione appesi all'url)
  - Se la navigazione dell'utente procederà attraverso gli hyperlink la sessione verrà mantenuta: il client riproporrà al server l'identificatore della sessione nelle successive richieste HTTP
  - Identificazione trasparente della sessione

URL rewriting tramite il metodo encodeURL(…)

- Attenzione: il contenitore **riscrive gli URL solo se lo sviluppatore lo richiede esplicitamente**.
- Per realizzare questa operazione si usa il metodo **`String encodeURL(String url)`** di **`HttpServletResponse`**
  - il parametro **`url`** rappresenta l'URL non riscritto
  - il risultato del metodo è l'URL riscritto dal contenitore con appeso l'ID di sessione
- Se il programmatore richiede la riscrittura degli URL il comportamento del container è quello di commutare automaticamente tra le due modalità:
  - Se il browser del client accetta i cookie la sessione viene gestita solo con i cookie
  - Se i cookie vengono rifiutati, viene attivata la riscrittura degli URL

URL rewriting

- Poiché la riscrittura dell'URL avviene in modo trasparente da parte del contenitore
  - Lo sviluppatore deve solo usare **`encodeURL()`**
- Il contenitore si preoccupa di adottare la riscrittura degli url **solo** nel caso in cui i cookie vengono rifiutati
- E' quindi conveniente utilizzare sempre **`encodeURL()`** per rendere più robusta la gestione delle sessioni

Domande:

- Appurato che il meccanismo di gestione della sessione sul server basato su trasmissione di cookie potrebbe non funzionare, allora:
- 1. Perché non adottare sempre e solo l'url rewriting?
- 2. Perché non adottare entrambi i metodi contemporaneamente e non alternativamente?

Domanda 1: solo URL rewriting?

- Perché usare i cookie se a volte i browser li rifiutano? Non sarebbe stato meglio progettare container che utilizzassero sempre e solo l'URL rewriting?

  L'URL rewriting consente la gestione della sessione solo se l'utente naviga attraverso i link della pagina di risposta.

  La sessione viene persa se:

  - l'utente usa dei bookmark
  - l'utente usa il tasto backward raggiungendo una pagina richiesta precedentemente senza identificativo riscritto
  - l'utente scrive l'url sulla barra degli indirizzi del browser

Domanda 2: URL rewriting AND cookie?

- Perché il metodo encodeURL() non funziona semplicemente aggiungendo sempre l'id di sessione invece che farlo solo se il browser dell'utente non accetta i cookie?
- L'encoding dell'URL costituisce un carico sul server per riscrivere le stringhe
- Carico sulla rete perché ciascuna risposta conterrà diversi url riscritti (notare che nel caso di gestione tramite cookie il server invia il cookie di sessione solo al momento della sua creazione).

Logica di funzionamento del metodo encodeURL()

- L'interfaccia HttpServletResponse fornisce questo metodo:

  java.lang.String **encodeURL**(java.lang.String url)

Dalla documentazione:

> Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged

Logica di funzionamento del metodo encodeURL()

- Se il browser dell'utente accetta i cookie, il metodo lascia le URL inalterate
- Se il browser dell'utente NON accetta i cookie, il metodo effettua la riscrittura dell'URL passato come argomento
- COME FA QUESTO METODO A CAPIRE SE IL BROWSER DELL'UTENTE ACCETTA I COOKIE???

  Se la richiesta non contiene già dei cookie non ha modo di saperlo.
  Torniamo allora al primo esempio …

Logica di funzionamento del metodo encodeURL()

Prima che il server si accorga se il browser dell'utente accetta i cookie o no, le URL devono essere già state riscritte, altrimenti la sessione viene persa!

Logica di funzionamento del metodo encodeURL(): prima richiesta di una sessione

- All'atto della **creazione** dell'oggetto sessione, il **cookie** di sessione viene **sempre automaticamente aggiunto** all'oggetto rappresentativo della risposta
- La prima volta che viene utilizzato il metodo encodeURL nel corso di una sessione, il container può non sapere se il browser accetti i cookie o no.
- Al primo utilizzo del metodo encodeURL vengono inviati sia il cookie di sessione che le URL riscritte (solo quelle per cui lo sviluppatore avrà richiesto la riscrittura).

Logica di funzionamento del metodo encodeURL(): richieste successive alla prima di una sessione

- All'arrivo di richieste successive a quella che ha generato la sessione corrente, viene controllato se l'ID di sessione è stato ottenuto 1) anche (o solo) tramite un cookie o 2) non è stato ottenuto da un cookie.
  - Nel primo caso non viene effettuato encoding,
  - Nel secondo caso l'URL viene riscritta con appeso l'ID di sessione.

Supporto dell'interfaccia HttpServletRequest alle operazioni di encoding dell'URL

- L'interfaccia HttpSerlvetRequest fornisce i seguenti metodi:
  - boolean **isRequestedSessionIdFromCookie()**

    Checks whether the requested session ID came in as a cookie.

  - boolean **isRequestedSessionIdFromURL()**

    Checks whether the requested session ID came in as part of the request URL.

Argomenti prossime lezioni

1. Ricapitolazione sulla redirezione tramite metodo sendRedirect dell'interfaccia HttpServletResponse
2. Concetto di sessione di navigazione
3. Uso di cookie per la gestione di una sessione
4. Uso dell'interfaccia HttpSession
5. URL rewriting nella gestione della sessione
6. Redirezione tramite il metodo forward dell'interfaccia RequestDispatcher

Redirezione di richieste

Configurazione di una servlet

- Il container utilizza un oggetto corrispondente all'interfaccia ServletConfig per passare informazioni alla servlet nel momento della sua creazione
- Si può ottenere tale oggetto per una specifica servlet con il metodo Servlet.getServletConfig()
- **Metodi:**
  - java.lang.String **getInitParameter**(java.lang.String name)
    Fornisce una stringa corrispondente al valore del parametro indicato.
  - java.util.Enumeration **getInitParameterNames**()
    Fornisce un elenco di nomi di parametri di inizializzazione
  - java.lang.String **getServletName**()
    Fornisce il nome dell'istanza corrente della servlet
  - ServletContext **getServletContext**()
    Fornisce il riferimento al contesto operativo in cui viene eseguita la servlet chiamante (segue).

Contesto di una servlet (1)

- L'oggetto ServletContext è contenuto nell'oggetto ServletConfig creato e associato ad una servlet al momento della sua creazione.
- Viene definito attraverso l'interfaccia

  Interface ServletContext
- Definisce un insieme di metodi che una servlet usa per comunicare con il proprio container (il motore servlet)
- Esiste un solo contesto per ciascuna web application
- Nel servlet container esistono uno o più contesti servlet e ogni servlet deve essere contenuta in un contesto

Contesto di una servlet (2)

- Il contesto delle servlet è un contenitore di oggetti condivisi e può essere usato per comunicazioni tra servlet di una stessa web application

- Esempio: per trasferire una richiesta da una servlet ad un'altra dello stesso contesto si può usare il metodo di ServletContext:
  getRequestDispatcher(java.lang.String path)
  che fornisce un oggetto RequestDispatcher associato al percorso path

Interface RequestDispatcher (1)

- Si tratta di un oggetto che riceve richieste da un client e le inoltra a qualsiasi risorsa (servlet, pagine HTML o JSP) sul server.
- Un oggetto che implementa l'interfaccia RequestDispatcher viene richiamato attraverso il metodo
  - ServletContext.getRequestDispatcher("<URL>");

Interface RequestDispatcher: forward()

- public void **forward**(ServletRequest request, ServletResponse response)
- Inoltra una richiesta da una servlet ad un'altra risorsa (servlet o pagina JSP o HTML) sullo stesso server.
- Se l'oggetto RequestDispatcher è stato ottenuto attraverso una chiamata del metodo getRequestDispatcher(<URL>), il percorso di destinazione dell'oggetto ServletRequest è stato riconfigurato con l'<URL> specificato.
  - Il metodo forward deve essere chiamato prima che l'oggetto risposta venga inviato altrimenti si genera una IllegalStateException.
  - **Parametri:**
    - richiesta – un oggetto ServletRequest: la richiesta ricevuta dal client
    - risposta – un oggetto ServletResponse: la risposta che deve pervenire al client
- Questo metodo consente di effettuare un'elaborazione preliminare della richiesta da parte di una risorsa e demandare l'elaborazione definitiva della risposta ad un'altra risorsa.

Interface RequestDispatcher: include()

- public void **include**(ServletRequest request, ServletResponse response)
  - Include il contenuto di una risorsa (servlet o pagina JSP o HTML) nella risposta.
  - L'oggetto ServletResponse è lo stesso utilizzato dalla risorsa chiamante e i percorsi non vengono riconfigurati perché il client riceve la risposta dalla servlet che ha eseguito il metodo include.
  - **Parameters:**
    - richiesta – un oggetto ServletRequest: la richiesta ricevuta dal client
    - risposta – un oggetto ServletResponse: la risposta che deve pervenire al client

Inoltrare una richiesta da una servlet ad un'altra risorsa (servlet, jsp, html)

- L'oggetto RequestDispatcher fornisce un metodo alternativo al metodo ServletResponse.sendRedirect(<URL>)

- 1. Si invoca il metodo **getRequestDispatcher** di **ServletContext**
  - Si fornisce la URL relativa al server o alla applicazione web (NO PERCORSI ASSOLUTI!)

```
String url = "/welcome1";
RequestDispatcher dispatcher =
  getServletContext().getRequestDispatcher(url);
```

Inoltrare una richiesta... metodo alternativo

- 2. Si invoca il metodo **`forward`** per trasferire il controllo completo alla pagina di destinazione
  - non è prevista nessuna comunicazione con il client, al contrario di quanto avviene con **`response.sendRedirect`**)

- 2 bis. Si invoca il metodo **`include`** per inserire l'output della pagina di destinazione e continuare l'elaborazione

Inoltrare una richiesta: esempio

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
  String operation = request.getParameter("destinazione");
  if (operation = = null) {
    operation = "unknown";
  }
  if (operation.equals("destinazione1")) {
    gotoPage("/operations/paginadidestinazione.jsp",
             request, response);
  } else if (operation.equals("destinazione2")) {
    gotoPage("/operations/servletdidestinazione",
             request, response);
  } else {
    gotoPage("/operations/servletdierrore",
             request, response);
  }
}
```

Inoltrare una richiesta: esempio (cont.)

```
private void gotoPage(String address,
                      HttpServletRequest request,
                      HttpServletResponse response)
    throws ServletException, IOException {
  RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher(address);
  dispatcher.forward(request, response);
}
```

Inoltrare una richiesta... (continua)

- Il metodo **`forward(req, resp)`** della classe **`RequestDispatcher`**
  - La locazione deve essere all'interno della applicazione web, non può essere un URL esterno
  - La redirezione non coinvolge il client. Avviene in modo trasparente al client
  - Può essere utilizzata sia per richieste di GET che per richieste di POST
  - E' più efficiente del metodo **`sendRedirect()`** e va preferito a quest'ultimo ove possibile

Attenzione alle richieste di POST...

- Al contrario delle richieste di GET, non possono essere inoltrate a normali pagine HTML.
- Se avete l'esigenza di inoltrare una richiesta di POST ad una pagina HTML statica rinominatela con estensione **`*.jsp`**
  - nomefile.html non può gestire richieste di POST
  - nomefile.jsp restituisce la stessa risposta sia alla richiesta di GET che alla richiesta di POST

Come fornire dati alla pagina/servlet di destinazione

- **Se** :
  - la richiesta può essere inoltrata a più pagine di destinazione
  - richiede l'elaborazione di dati contenuti nell'oggetto **`HttpServletRequest`**
- conviene spostare l'elaborazione dei dati nella servlet da cui la richiesta ha origine e passare alla pagina/servlet di destinazione i dati già elaborati

Come fornire dati alla pagina di destinazione

- i dati preventivamente elaborati dalla servlet di origine possono essere inclusi come attributi dell'oggetto **`HttpServletRequest`**
- **`request.setAttribute("key1", value1);`**
- La pagina di destinazione può prelevare questi dati
- **`Type1 value1 = (Type1)request.getAttribute("key1");`**

Pagine JSP

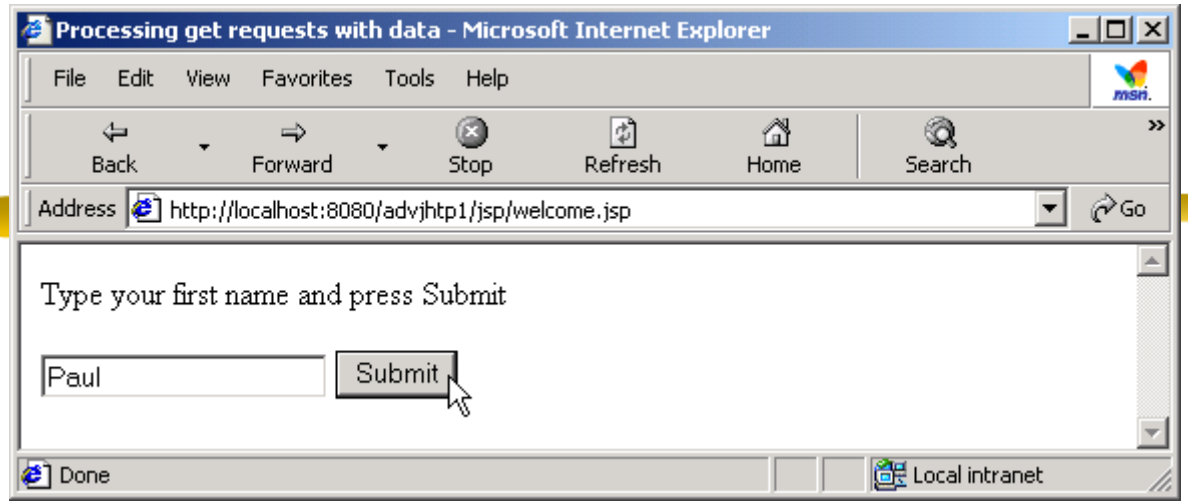
Un esempio, prima di cominciare la trattazione teorica

```
4
5  <!-- welcome.jsp -->
6  <!-- JSP that processes a "get" request containing data. -->
7
8  <html>
9
10    <!-- head section of document -->
11    <head>
12       <title>Processing "get" requests with data</title>
13    </head>
14
15    <!-- body section of document -->
16    <body>
17       <% // begin scriptlet
18
19          String name = request.getParameter( "firstName" );
20
21          if ( name != null ) {
22
23       %> <%-- end scriptlet to insert fixed template data --%>
24
25             <h1>
26                Hello <%= name %>, <br />
27                Welcome to JavaServer Pages!
28             </h1>
29
30       <% // continue scriptlet
31
32          }  // end if
```

Uso di scriptlet per inserire codice java

Uso dell'oggetto implicito **`request`** per ottenere il valore di un parametro

```
33          else {
34
35       %> <%-- end scriptlet to insert fixed template data --%>
36
37             <form action = "welcome.jsp" method = "get">
38                <p>Type your first name and press Submit</p>
39
40                <p><input type = "text" name = "firstName" />
41                   <input type = "submit" value = "Submit" />
42                </p>
43             </form>
44
45       <% // continue scriptlet
46
47          }  // end else
48
49       %> <%-- end scriptlet --%>
50    </body>
51
52 </html>  <!-- end XHTML document -->
```


Processing get requests with data - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/advjhtp1/jsp/welcome.jsp
Go
Type your first name and press Submit
Paul
Submit
Done
Local intranet


Processing get requests with data - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/advjhtp1/jsp/welcome.jsp?firstName=Paul
Go
Hello Paul,
Welcome to JavaServer Pages!
Done
Local intranet

Java Server Pages (JSP)

- Java Server Pages
  - Costituiscono un'estensione della tecnologia delle servlet
- Classi e interfacce specifiche per la definizione di pagine JSP:
  - Package **`javax.servlet.jsp`**
  - Package **`javax.servlet.jsp.tagext`**

Java Server Pages (JSP)

- Il JSP container (al momento della prima invocazione della pagina):
  - Legge la pagina JSP
  - Scrive una servlet (corrispondente alla pagina letta)
  - La compila (default usa javac, ma configurabile)
  - La invoca secondo il ciclo di vita della servlet stessa (inizializzazione, servizio)
- Alle invocazioni successive il container fa riferimento alla servlet già caricata in memoria e inizializzata, pertanto esegue solo il metodo di servizio.

Java Server Pages (JSP)

- Come per le servlet, le pagine JSP utilizzano un oggetto:
  - request (rappresentativo della richiesta HTTP pervenuta)
  - response (rappresentativo della risposta HTTP da inviare)
  - hanno inoltre accesso a tutti i dati della richiesta, del contesto e dell'applicazione web.

Java Server Pages - componenti

- Elementi che costituiscono una pagina JSP:
  - Direttive, cioè istruzioni dirette al servlet/JSP container che specificano come gestire la JSP
  - Azioni
    - Scriptlet (e varianti, come espressioni, dichiarazioni ecc.)
    - Azioni standard
    - Tag personalizzati

Java Server Pages - direttive

- Direttive
  - Istruzioni dirette al JSP container
    - i.e. programma che gestisce le pagine JSP fino alla loro esecuzione
  - Consentono al programmatore di specificare
    - Impostazioni e opzioni della pagina
    - Contenuti esterni da includere o package da importare
    - Librerie di tag personalizzati utilizzabili nella pagina

Java Server Pages - direttive

- Sintassi:
  - <%@ direttiva {attr="valore"}%>
    - Es: <%@ page language="java"%>
      specifica il linguaggio di scripting
      (la spec. JSP 1.1 riconosce solo Java)
    - Es: <%@ include file="relativeURLspec"%>
      specifica il percorso relativo (URL) di un file che deve essere incluso

Java Server Pages - direttiva page

Direttiva	Descrizione
Direttive **page**	
import="importlist"	Fornisce un elenco di package da importare. Utile per non dover scrivere tutto il percorso dei package (nomi di classi pienamente qualificati). Per impostazione predefinita vengono importati: *java.lang.*, javax.servlet.*, javax.servlet.jsp.** e *javax.servlet.http.**
session="true|false"	Se *true* la pagina ha accesso alla variabile implicita *session*, che fa riferimento alla sessione attuale. Per impostazione predefinita è *true*.
isThreadSafe="true|false"	Se *true*, il container può inviare alla pagina nuove richieste prima che vengano completate le richieste in corso. Se questo attributo è su *false* le richieste verranno inviate progressivamente, con possibili conseguenze a livello di prestazioni. L'impostazione predefinita è *true*.
errorPage="error_url"	Se su questa pagina si verifica un'eccezione non catturata, il container passerà alla pagina qui indicata.

Java Server Pages - azioni

- Azioni
  - Sono codificate in un linguaggio di programmazione (JSP 1.1 consente soltanto Java)
  - Specificate sotto forma di
    - scriptlet
      - » codice puro <% sorgente scriptlet %>
      - » Varianti: espressioni <%= espressione %>, dichiarazioni <%! Dichiarazione %>, commento <%-- commento --%>
    - azione standard

      <jsp:actionName attributo="valore">body</jsp:faiqualcosa>
    - tag personalizzati

      <tagPrefix:tagName attributo="valore">body</tag>

JavaServer Pages - scriptlet

- Scriptlet
  - Sono blocchi di codice eseguiti nel contesto della pagina
  - Consentono l'inserimento di codice Java all'interno della pagina JSP
  - Realizzano l'elaborazione della richiesta
    - Interagiscono con gli elementi della pagina e altre componenti per creare pagine dinamiche

JSP: azioni standard e tag personalizzati

- Azioni standard: tag JSP dal comportamento predefinito, comportano l'esecuzione di codice, parametrizzato in base agli attributi del tag.

- Librerie di tag personalizzati
  - Meccanismo di estensione dell'insieme dei tag JSP predefiniti
  - Consente la definizione di nuovi tag da parte del programmatore
    - Nuovi tag possono incapsulare complesse funzionalità

Traduzione della pagina JSP in una servlet

- Il codice contenuto nella pagina JSP
  - Costituirà un blocco di codice all'interno della definizione di un servlet
    - Metodo di inizializzazione _jspInit()
    - Il metodo **`_jspService()`** conterrà il codice degli scriptlet e una sequenza di istruzioni di write("…") per tutti i tag HTML presenti nella pagina .jsp
    - Metodo di distruzione _jspDestroy()

- Ciclo di vita della JSP simile a quello di una servlet.
  - La prima volta che viene invocata la pagina, il container la traduce in una servlet, che viene compilata e mandata in esecuzione (esecuzione del metodo _jspInit).
  - Il container invoca il metodo _jspService ad ogni richiesta successiva della stessa pagina JSP.

Errori JSP

- Errori al momento della traduzione
  - Si verificano nel momento in cui viene generata la servlet corrispondente alla JSP
- Errori al momento della richiesta
  - Si verificano durante l'elaborazione della richiesta

JSP o Servlet?

- JSP
  - Hanno l'aspetto e la struttura di pagine XHTML
    - Contengono markup HTML o XHTML
  - Vengono utilizzate quando la maggior parte del contenuto che deve essere visualizzato segue una struttura fissata.
    - In generale una piccola parte del contenuto deve essere generata dinamicamente
- Servlet
  - Utilizzate invece quando solo una piccola porzione del contenuto deve seguire una struttura fissata
    - La maggior parte del contenuto deve essere generata dinamicamente

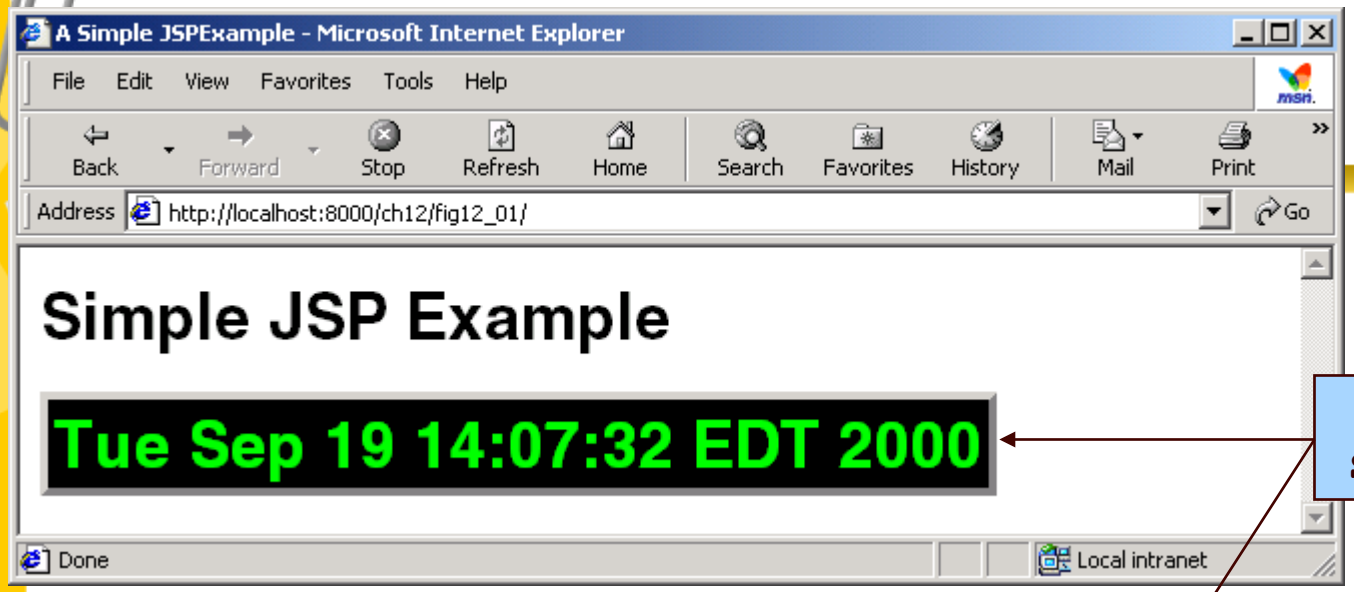
Esempio di pagina JSP

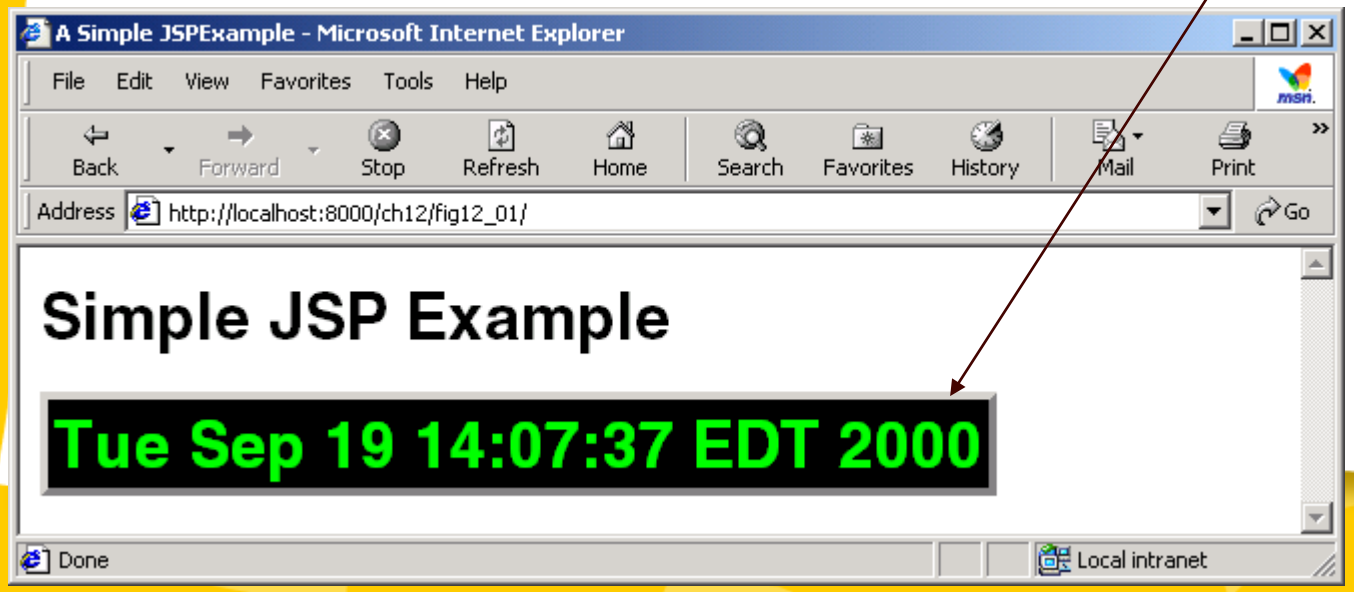
```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5
6  <html xmlns = "http://www.w3.org/1999/xhtml">
7
8
9    <head>
10     <meta http-equiv = "refresh" content = "60" />
11
12     <title>A Simple JSP Example</title>
13
14     <style type = "text/css">
15       .big { font-family: helvetica, arial, sans-serif;
16            font-weight: bold;
17            font-size: 2em; }
18     </style>
19   </head>
20
21   <body>
22     <p class = "big">Simple JSP Example</p>
23
24     <table style = "border: 6px outset;">
25       <tr>
26         <td style = "background-color: black;">
27           <p class = "big" style = "color: cyan;">
28
29             <!-- JSP expression to insert date/time -->
30             <%= new java.util.Date() %>
31
32           </p>
33         </td>
34       </tr>
35     </table>
```

`meta` element refresh: ricarica la pagina ogni **`60`** seconds

Crea un oggetto **`Date`** che viene implicitamente convertito in un oggetto **`String:`** si tratta di un'espressione

```
36    </body>
37
38  </html>
```





Rappresentazione tramite **`String`** dell'oggetto **`Date`**

Aree di visibilità (scope) in una pagina JSP

- Le pagine JSP possono accedere ad oggetti definiti in diverse aree di visibilità (scope):
  - Applicazione
    - Oggetti associati al **contesto servlet** della JSP;
    - Per recuperare tali oggetti si ricorre al metodo javax.servlet.ServletContext.**getAttribute**
  - Pagina
    - Oggetti che sono visibili solo al codice presente sulla stessa pagina
    - Per accedervi si usa javax.servlet.jsp.PageContext.**getAttribute**
    - Una volta completata la richiesta della pagina il container elimina il riferimento a tali oggetti

Aree di visibilità (continua)

- Richiesta
  - Visibilità uguale alla richiesta
  - Vi si accede con il metodo javax.servlet.ServletRequest.**getAttribute**
  - Viene eliminato il riferimento a tali oggetti quando viene inviata la risposta
- Sessione
  - Oggetti associati ad una sessione utente
  - Vi si accede con il metodo javax.servlet.http.**HTTPSession.getAttribute**
  - I riferimenti a tali oggetti vengono eliminati quando la sessione termina (per volere del client o per timeout)

Oggetti impliciti

Implicit Object	Description
Application Scope	
`application`	This **`javax.servlet.ServletContext`** object represents the container in which the JSP executes.
Page Scope	
`config`	This **`javax.servlet.ServletConfig`** object represents the JSP configuration options. As with servlets, configuration options can be specified in a Web application descriptor.
`exception`	This **`java.lang.Throwable`** object represents the exception that is passed to the JSP error page. This object is available only in a JSP error page.
`out`	This **`javax.servlet.jsp.JspWriter`** object writes text as part of the response to a request. This object is used implicitly with JSP expressions and actions that insert string content in a response.
`page`	This **`java.lang.Object`** object represents the **`this`** reference for the current JSP instance.
`pageContext`	This **`javax.servlet.jsp.PageContext`** object hides the implementation details of the underlying servlet and JSP container and provides JSP programmers with access to the implicit objects discussed in this table.

Oggetti impliciti (cont.)

Implicit Object	Description
`response`	This object represents the response to the client. The object normally is an instance of a class that implements **`HttpServletResponse`** (package **`javax.servlet.http`**). If a protocol other than HTTP is used, this object is an instance of a class that implements **`javax.servlet.ServletResponse`** .
Request Scope	
`request`	This object represents the client request. The object normally is an instance of a class that implements **`HttpServletRequest`** (package **`javax.servlet.http`**). If a protocol other than HTTP is used, this object is an instance of a subclass of **`javax.servlet.ServletRequest`** .
Session Scope	
`session`	This **`javax.servlet.http.HttpSession`** object represents the client session information if such a session has been created. This object is available only in pages that participate in a session.

Componenti di scripting JSP

- Come specificare componenti JSP
  - Scriptlets (delimitate da **<%** and **%>**)
  - Commenti (delimitati da **<%--** and **--%>**)
  - Espressioni (delimitati da **<%=** and **%>**)
  - Dichiarazioni (delimitati da **<%!** and **%>**)

Scriptlet

- Delimitate da <% e %>
- Blocchi di codice Java
- Inserite nel metodo **_jspService** al momento della traduzione

- Scriptlet, espressioni e codice XHTML possono essere intercalati per creare diverse risposte sulla base di informazioni incluse nella richiesta

Componenti di scripting JSP (seq. di escape)

Literal	Escape sequence	Description
`<%`	`<\%`	The character sequence `<%` normally indicates the beginning of a scriptlet. The `<\%` escape sequence places the literal characters `<%` in the response to the client.
`%>`	`%\>`	The character sequence `%>` normally indicates the end of a scriptlet. The `%\>` escape sequence places the literal characters `%>` in the response to the client.

`'`
`"`
`\`
	`\'`
	`\"`
	`\\`
		As with string literals in a Java program, the escape sequences for characters `'`, `"` and `\` allow these characters to appear in attribute values. Remember that the literal text in a JSP becomes string literals in the servlet that represents the translated JSP .

Commenti

- Sono supportati tre tipi di commento:
  - Commento JSP
  - Commento XHTML
  - Commento del linguaggio di scripting

Commento JSP

- Commento JSP
  - <%-- --%>
    - Non si usa all'interno di scriptlet
    - Non è visibile al client

Commento XHTML

- Commento XHTML
  - <!-- -->
    - Non si usa all'interno di scriptlet
    - E' visibile al client

Commento del linguaggio di scripting

- Commento del linguaggio di scripting
  - Single-line //, oppure Multi-line /* */
    - Si usa esclusivamente all'interno di scriptlet
    - E' visibile al client?

Espressioni

- Sono delimitate da <%= e %>
- Contengono espressioni Java che vengono valutate quando il client richiede la pagina che le contiene
- Il container converte il risultato di un'espressione in un oggetto **String** e lo invia in output come parte della risposta

Java Server Pages (JSP)

- Java Server Pages
  - Costituiscono un'estensione della tecnologia delle servlet
- Classi e interfacce specifiche per la definizione di pagine JSP:
  - Package **`javax.servlet.jsp`**
  - Package **`javax.servlet.jsp.tagext`**

Java Server Pages (JSP)

- Il JSP container (al momento della prima invocazione della pagina):
  - Legge la pagina JSP
  - Scrive una servlet (corrispondente alla pagina letta)
  - La compila (default usa javac, ma configurabile)
  - La invoca secondo il ciclo di vita della servlet stessa (inizializzazione, servizio)
- Alle invocazioni successive il container fa riferimento alla servlet già caricata in memoria e inizializzata, pertanto esegue solo il metodo di servizio.

Java Server Pages (JSP)

- Come per le servlet, le pagine JSP utilizzano un oggetto:
  - request (rappresentativo della richiesta HTTP pervenuta)
  - response (rappresentativo della risposta HTTP da inviare)
  - hanno inoltre accesso a tutti i dati della richiesta, del contesto e dell'applicazione web.

Java Server Pages - componenti

- Elementi che costituiscono una pagina JSP:
  - Direttive, cioè istruzioni dirette al servlet/JSP container che specificano come gestire la JSP
  - Azioni
    - Scriptlet (e varianti, come espressioni, dichiarazioni ecc.)
    - Azioni standard
    - Tag personalizzati

Java Server Pages - direttive

- Direttive
  - Istruzioni dirette al JSP container
    - i.e. programma che gestisce le pagine JSP fino alla loro esecuzione
  - Consentono al programmatore di specificare
    - Impostazioni e opzioni della pagina
    - Contenuti esterni da includere o package da importare
    - Librerie di tag personalizzati utilizzabili nella pagina

Java Server Pages - direttive

- Sintassi:
  - <%@ direttiva {attr="valore"}%>
    - Es: <%@ page language="java"%>
      specifica il linguaggio di scripting
      (la spec. JSP 1.1 riconosce solo Java)
    - Es: <%@ include file="relativeURLspec"%>
      specifica il percorso relativo (URL) di un file che deve essere incluso

Java Server Pages - direttiva page

Direttiva	Descrizione
Direttive **page**	
import="importlist"	Fornisce un elenco di package da importare. Utile per non dover scrivere tutto il percorso dei package (nomi di classi pienamente qualificati). Per impostazione predefinita vengono importati: *java.lang.*, javax.servlet.*, javax.servlet.jsp.** e *javax.servlet.http.**
session="true|false"	Se *true* la pagina ha accesso alla variabile implicita *session*, che fa riferimento alla sessione attuale. Per impostazione predefinita è *true*.
isThreadSafe="true|false"	Se *true*, il container può inviare alla pagina nuove richieste prima che vengano completate le richieste in corso. Se questo attributo è su *false* le richieste verranno inviate progressivamente, con possibili conseguenze a livello di prestazioni. L'impostazione predefinita è *true*.
errorPage="error_url"	Se su questa pagina si verifica un'eccezione non catturata, il container passerà alla pagina qui indicata.

Java Server Pages - azioni

- Azioni
  - Sono codificate in un linguaggio di programmazione (JSP 1.1 consente soltanto Java)
  - Specificate sotto forma di
    - scriptlet
      - » codice puro <% sorgente scriptlet %>
      - » Varianti: espressioni <%= espressione %>, dichiarazioni <%! Dichiarazione %>, commento <%-- commento --%>
    - azione standard

      <jsp:actionName attributo="valore">body</jsp:faiqualcosa>
    - tag personalizzati

      <tagPrefix:tagName attributo="valore">body</tag>

JavaServer Pages - scriptlet

- Scriptlet
  - Sono blocchi di codice eseguiti nel contesto della pagina
  - Consentono l'inserimento di codice Java all'interno della pagina JSP
  - Realizzano l'elaborazione della richiesta
    - Interagiscono con gli elementi della pagina e altre componenti per creare pagine dinamiche

JSP: azioni standard e tag personalizzati

- Azioni standard: tag JSP dal comportamento predefinito, comportano l'esecuzione di codice, parametrizzato in base agli attributi del tag.

- Librerie di tag personalizzati
  - Meccanismo di estensione dell'insieme dei tag JSP predefiniti
  - Consente la definizione di nuovi tag da parte del programmatore
    - Nuovi tag possono incapsulare complesse funzionalità

Traduzione della pagina JSP in una servlet

- Il codice contenuto nella pagina JSP
  - Costituirà un blocco di codice all'interno della definizione di un servlet
    - Metodo di inizializzazione _jspInit()
    - Il metodo **`_jspService()`** conterrà il codice degli scriptlet e una sequenza di istruzioni di write("…") per tutti i tag HTML presenti nella pagina .jsp
    - Metodo di distruzione _jspDestroy()

- Ciclo di vita della JSP simile a quello di una servlet.
  - La prima volta che viene invocata la pagina, il container la traduce in una servlet, che viene compilata e mandata in esecuzione (esecuzione del metodo _jspInit).
  - Il container invoca il metodo _jspService ad ogni richiesta successiva della stessa pagina JSP.

Errori JSP

- Errori al momento della traduzione
  - Si verificano nel momento in cui viene generata la servlet corrispondente alla JSP
- Errori al momento della richiesta
  - Si verificano durante l'elaborazione della richiesta

JSP o Servlet?

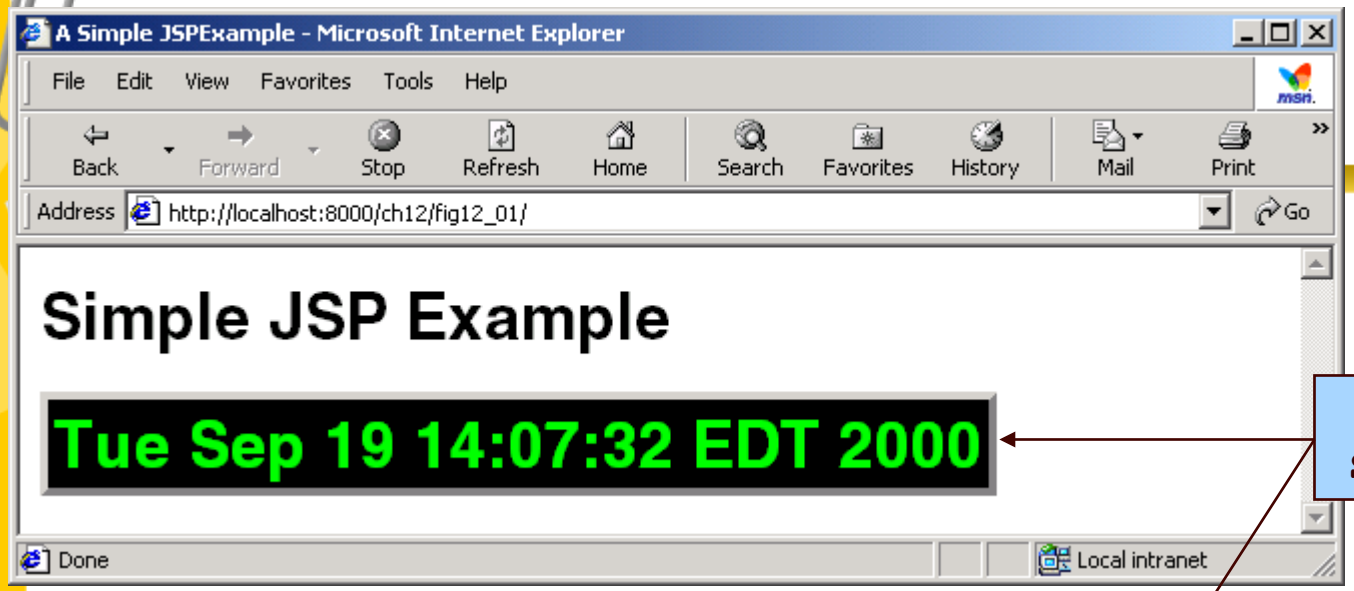
- JSP
  - Hanno l'aspetto e la struttura di pagine XHTML
    - Contengono markup HTML o XHTML
  - Vengono utilizzate quando la maggior parte del contenuto che deve essere visualizzato segue una struttura fissata.
    - In generale una piccola parte del contenuto deve essere generata dinamicamente
- Servlet
  - Utilizzate invece quando solo una piccola porzione del contenuto deve seguire una struttura fissata
    - La maggior parte del contenuto deve essere generata dinamicamente

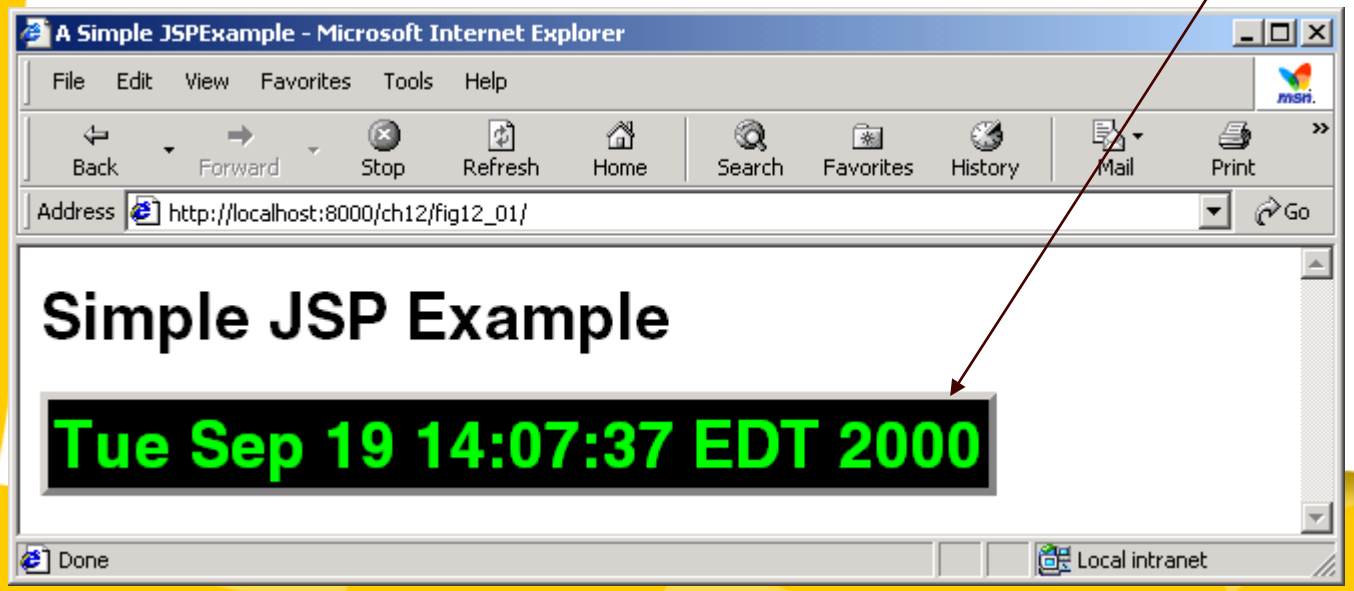
Uso di espressioni in una pagina JSP - esempio

```
4
5
6   <html xmlns = "http://www.w3.org/1999/xhtml">
7
8
9     <head>
10       <meta http-equiv = "refresh" content = "60" />
11
12       <title>A Simple JSP Example</title>
13
14       <style type = "text/css">
15          .big { font-family: helvetica, arial, sans-serif;
16                 font-weight: bold;
17                 font-size: 2em; }
18       </style>
19     </head>
20
21     <body>
22       <p class = "big">Simple JSP Example</p>
23
24       <table style = "border: 6px outset;">
25          <tr>
26             <td style = "background-color: black;">
27                <p class = "big" style = "color: cyan;">
28
29                   <!-- JSP expression to insert date/time -->
30                   <%= new java.util.Date() %>
31
32                </p>
33             </td>
34          </tr>
66       </table>
36     </body>
37
38  </html>
```

`meta` element refresh: ricarica la pagina ogni **`60`** seconds

Crea un oggetto **`Date`** che viene implicitamente convertito in un oggetto **`String`**: si tratta di un'espressione





Rappresentazione tramite **`String`** dell'oggetto **`Date`**

Aree di visibilità (scope) in una pagina JSP

- Le pagine JSP possono accedere ad oggetti definiti in diverse aree di visibilità (scope):
  - Applicazione
    - Oggetti associati al **contesto servlet** della JSP;
    - Per recuperare tali oggetti si ricorre al metodo javax.servlet.ServletContext.**getAttribute()**
  - Pagina
    - Oggetti che sono visibili solo al codice presente sulla stessa pagina
    - Per accedervi si usa javax.servlet.jsp.PageContext.**getAttribute()**
    - Una volta completata la richiesta della pagina il container elimina il riferimento a tali oggetti

Aree di visibilità (continua)

- Richiesta
  - Visibilità uguale alla richiesta
  - Vi si accede con il metodo javax.servlet.ServletRequest.**getAttribute()**
  - Viene eliminato il riferimento a tali oggetti quando viene inviata la risposta
- Sessione
  - Oggetti associati ad una sessione utente
  - Vi si accede con il metodo javax.servlet.http.**HTTPSession.getAttribute ()**
  - I riferimenti a tali oggetti vengono eliminati quando la sessione termina (per volere del client o per timeout)

Oggetti impliciti

Implicit Object	Description
Application Scope	
application	This **javax.servlet.ServletContext** object represents the container in which the JSP executes.
Page Scope	
config	This **javax.servlet.ServletConfig** object represents the JSP configuration options. As with servlets, configuration options can be specified in a Web application descriptor.
exception	This **java.lang.Throwable** object represents the exception that is passed to the JSP error page. This object is available only in a JSP error page.
out	This **javax.servlet.jsp.JspWriter** object writes text as part of the response to a request. This object is used implicitly with JSP expressions and actions that insert string content in a response.
page	This **java.lang.Object** object represents the **this** reference for the current JSP instance.
pageContext	This **javax.servlet.jsp.PageContext** object hides the implementation details of the underlying servlet and JSP container and provides JSP programmers with access to the implicit objects discussed in this table.

Oggetti impliciti (cont.)

Implicit Object	Description
`response`	This object represents the response to the client. The object normally is an instance of a class that implements **`HttpServletResponse`** (package **`javax.servlet.http`**). If a protocol other than HTTP is used, this object is an instance of a class that implements **`javax.servlet.ServletResponse`**.
Request Scope	
`request`	This object represents the client request. The object normally is an instance of a class that implements **`HttpServletRequest`** (package **`javax.servlet.http`**). If a protocol other than HTTP is used, this object is an instance of a subclass of **`javax.servlet.ServletRequest`**.
Session Scope	
`session`	This **`javax.servlet.http.HttpSession`** object represents the client session information if such a session has been created. This object is available only in pages that participate in a session.

Oggetto Page

- L'oggetto page rappresenta l'istanza corrente della servlet corrispondente alla pagina JSP
- Ha come tipo l'interfaccia HTTPJspPage che discende da JSP page, la quale a sua volta estende Servlet
- Può quindi essere quindi utilizzato per accedere a tutti i metodi definiti nelle servlet

```
<%@ page info="Esempio di uso page." %>
<p>Page info: <%=page.getServletInfo() %> </p>
```

JSP

```
<p>Page info: Esempio di uso di page</p>
```

HTML

Oggetto PageContext

- Oggetto che fornisce informazioni sul contesto di esecuzione della JSP
- Rappresenta l'insieme degli oggetti impliciti di una JSP
- Consente l'accesso a tutti gli oggetti impliciti e ai loro attributi attraverso i corrispondenti metodi *get*:

  getPage()
  getRequest()
  getResponse()
  getSession()
  getServletContext()
  getException() ecc.

- Poco usato per lo scripting, utile per costruire custom tags

Oggetto Application

- Oggetto che fornisce informazioni sul contesto di esecuzione della JSP (è il ServletContext)
- Rappresenta la web application a cui la JSP appartiene
- Consente di interagire con l'ambiente di esecuzione:
  - garantisce l'accesso a risorse server-side
  - permette accesso ai parametri di inizializzazione relativi all'applicazione
  - consente di gestire gli attributi di un'applicazione

Oggetto Exception

- Oggetto connesso alla gestione degli errori
- Rappresenta l'eccezione che non viene gestita da nessun blocco catch
- Non è automaticamente disponibile in tutte le pagine ma solo nelle Error Page (quelle dichiarate con l'attributo errorPage impostato a true)

```
<%@ page isErrorPage="true" %>
<h1>Attenzione!</h1>
E' stato rilevato il seguente errore:<br/>
<b><%= exception %></b><br/>
<% exception.printStackTrace(out); %>
```

Componenti di scripting JSP

- Come specificare componenti JSP
  - Scriptlets (delimitate da **<%** and **%>**)
  - Commenti (delimitati da **<%--** and **--%>**)
  - Espressioni (delimitati da **<%=** and **%>**)
  - Dichiarazioni (delimitati da **<%!** and **%>**)

Scriptlet

- Delimitate da <% e %>
- Blocchi di codice Java
- Inserite nel metodo **_jspService** al momento della traduzione

- Scriptlet, espressioni e codice XHTML possono essere intercalati per creare diverse risposte sulla base di informazioni incluse nella richiesta

Componenti di scripting JSP (seq. di escape)

Literal	Escape sequence	Description
`<%`	`<\%`	The character sequence `<%` normally indicates the beginning of a scriptlet. The `<\%` escape sequence places the literal characters `<%` in the response to the client.
`%>`	`%\>`	The character sequence `%>` normally indicates the end of a scriptlet. The `%\>` escape sequence places the literal characters `%>` in the response to the client.

`'`
`"`
`\`
	`\'`
	`\"`
	`\\`
		As with string literals in a Java program, the escape sequences for characters `'`, `"` and `\` allow these characters to appear in attribute values. Remember that the literal text in a JSP becomes string literals in the servlet that represents the translated JSP .

Commenti

- Sono supportati tre tipi di commento:
  - Commento JSP
  - Commento XHTML
  - Commento del linguaggio di scripting

Commento JSP

- Commento JSP
  - <%-- --%>
    - Non si usa all'interno di scriptlet
    - Non è visibile al client

Commento XHTML

- Commento XHTML
  - <!-- -->
    - Non si usa all'interno di scriptlet
    - E' visibile al client

Commento del linguaggio di scripting

- Commento del linguaggio di scripting
  - Single-line //, oppure Multi-line /* */
    - Si usa esclusivamente all'interno di scriptlet
    - E' visibile al client?

Espressioni

- Sono delimitate da <%= e %>
- Contengono espressioni Java che vengono valutate quando il client richiede la pagina che le contiene
- Il container converte il risultato di un'espressione in un oggetto **String** e lo invia in output come parte della risposta (metodo _jspService())

Dichiarazioni

- Delimitate da <%! e %>
- Consentono la definizione di variabili e metodi attraverso la sintassi di Java
- Le variabili diventano attributi della classe servlet che rappresenta la pagina JSP
- I metodi così dichiarati corrisponderanno ai metodi della classe servlet che rappresenta la pagina JSP
- La stessa variabile senza <%! diventa var locale di _jspService()

Esempio sulla dichiarazione

- Scrivere una pagina.jsp contenente una variabile **contatore** e delle istruzioni di incremento del suo valore.
- Ci sono differenze se la variabile viene utilizzata nei seguenti modi:
  - <% int counter=0; %>
  - <%! int counter=0; %> ?

Nel primo caso scrive sempre:
1, 2

Nel secondo ?

```
1 <html>
2 <head>
3  <title>
4  Contatore dichiarato come scriptlet
5  </title>
6
7 </head>
8 <body>
9 <%@ page language="java" %>
10 <% int counter=0; %>     oppure     <%! int counter=0; %>
11 <% counter++; %>
12 <p>Il contatore vale <%= counter %>.</p>
13 <% counter++; %>
14 <p>Il contatore vale <%= counter %>.</p>
15
16
17 </body>
18 </html>
```

Azione standard `<jsp:include>`

Reminder: redirezione attraverso direttiva include
<%@ include file="relativeURLspec"%>
specifica il percorso relativo (URL) di un file che deve essere incluso

- **`<jsp:include …>`**
  - Consente l'inclusione di contenuto **dinamico** in una pagina JSP
  - Più flessibile della direttiva **`include`**
    - Richiede maggiore overhead quando il contenuto della pagina cambia frequentemente
    - La direttiva include il codice al momento della traduzione (l'inclusione corrisponde ad una serie di print), mentre l'azione standard include il codice solo al momento dell'esecuzione

Azione <jsp:include>

Attribute	Description
page	Specifies the relative URI path of the resource to include. The resource must be part of the same Web application.
flush	Specifies whether the buffer should be flushed before the **include** is performed. In JSP 1.1, this attribute is required to be **true**.

```
<!-- banner.html              -->
<!-- banner to include in another document -->
<div style = "width: 580px">
  <p>
    Corso di
    Laboratorio <br /> Internet and
    World Wide Web Programming Training <br />
    On-Site Seminars Delivered Worldwide
  </p>

  <p>
    <a href = "mailto:novella@di.uniroma1.it">
      novella@di.uniroma1.it</a><br />

    978.579.9911<br />
    490B Boston Post Road, Suite 200,
    Sudbury, MA 01776
  </p>
</div>
```

```
<!-- toc.html                    -->
<!-- contents to include in another document -->

<p><a href = "http://www.uniroma1.it/books/index.html">
  Publications/BookStore
</a></p>

<p><a href = "http://www.uniroma1.it/whatsnew.html">
  What's New
</a></p>

<p><a href = "http://www.uniroma1.it/books/downloads.html">
  Downloads/Resources
</a></p>

<p><a href = "http://www.uniroma1.it/faq/index.html">
  FAQ (Frequently Asked Questions)
</a></p>

<p><a href = "http://www.uniroma1.it/intro.html">
  Who we are
</a></p>

<p><a href = "http://www.uniroma1.it/index.html">
  Home Page
</a></p>

<p>Send questions or comments about this site to
  <a href = "mailto:novella@di.uniroma1.it">
    novella@di.uniroma1.it
  </a><br />


</p>
```

```jsp
<!-- Fig. 10.9: clock2.jsp                    -->
<!-- date and time to include in another document -->

<table>
  <tr>
    <td style = "background-color: black;">
      <p class = "big" style = "color: cyan; font-size: 3em;
        font-weight: bold;">

        <%-- script to determine client local and --%>
        <%-- format date accordingly             --%>
        <%
          // get client locale
          java.util.Locale locale = request.getLocale();

          // get DateFormat for client's Locale
          java.text.DateFormat dateFormat =
            java.text.DateFormat.getDateTimeInstance(
              java.text.DateFormat.LONG,
              java.text.DateFormat.LONG, locale );

        %>  <%-- end script --%>

        <%-- output date --%>
        <%= dateFormat.format( new java.util.Date() ) %>
      </p>
    </td>
  </tr>
</table>
```

Use **`Locale`** to format **`Data`** with specified **`DataFormat`**

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- include.jsp -->

<html xmlns = "http://www.w3.org/1999/xhtml">

  <head>
    <title>Using jsp:include</title>

    <style type = "text/css">
      body {
        font-family: tahoma, helvetica, arial, sans-serif;
      }

      table, tr, td {
        font-size: .9em;
        border: 3px groove;
        padding: 5px;
        background-color: #dddddd;
      }
    </style>
  </head>

  <body>
    <table>
      <tr>
        <td style = "width: 160px; text-align: center">
          <img src = "images/logotiny.png"
            width = "140" height = "93"
          />
        </td>

```

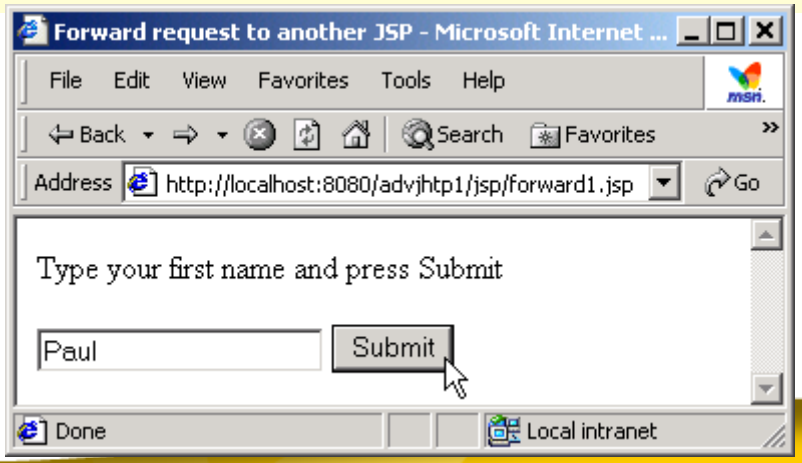
```
35          <td>
36
37            <%-- include banner.html in this JSP --%>
38            <jsp:include page = "banner.html"
39              flush = "true" />
40
41          </td>
42        </tr>
43
44        <tr>
45          <td style = "width: 160px">
46
47            <%-- include toc.html in this JSP --%>
48            <jsp:include page = "toc.html" flush = "true" />
49
50          </td>
51
52          <td style = "vertical-align: top">
53
54            <%-- include clock2.jsp in this JSP --%>
55            <jsp:include page = "clock2.jsp"
56              flush = "true" />
57
58          </td>
59        </tr>
60      </table>
61    </body>
62  </html>
```

Azione `<jsp:forward>`

- **`<jsp:forward>`**
  - Consente ad una pagina JSP di inoltrare la richiesta ad altre risorse
- L'azione **`<jsp:param>`** (annidata nel forward) specifica coppie nome/valore di dati da allegare ad altre azioni

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.11: forward1.jsp -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8
9  <head>
10   <title>Forward request to another JSP</title>
11 </head>
12
13 <body>
14   <% // begin scriptlet
15
16     String name = request.getParameter( "firstName" );
17
18     if ( name != null ) {
19
20   %> <%-- end scriptlet to insert fixed template data --%>
21
22       <jsp:forward page = "forward2.jsp">
23         <jsp:param name = "date"
24           value = "<%= new java.util.Date() %>" />
25       </jsp:forward>
26
27   <% // continue scriptlet
28
29     }  // end if
30     else {
31
32   %> <%-- end scriptlet to insert fixed template data --%>
33
```

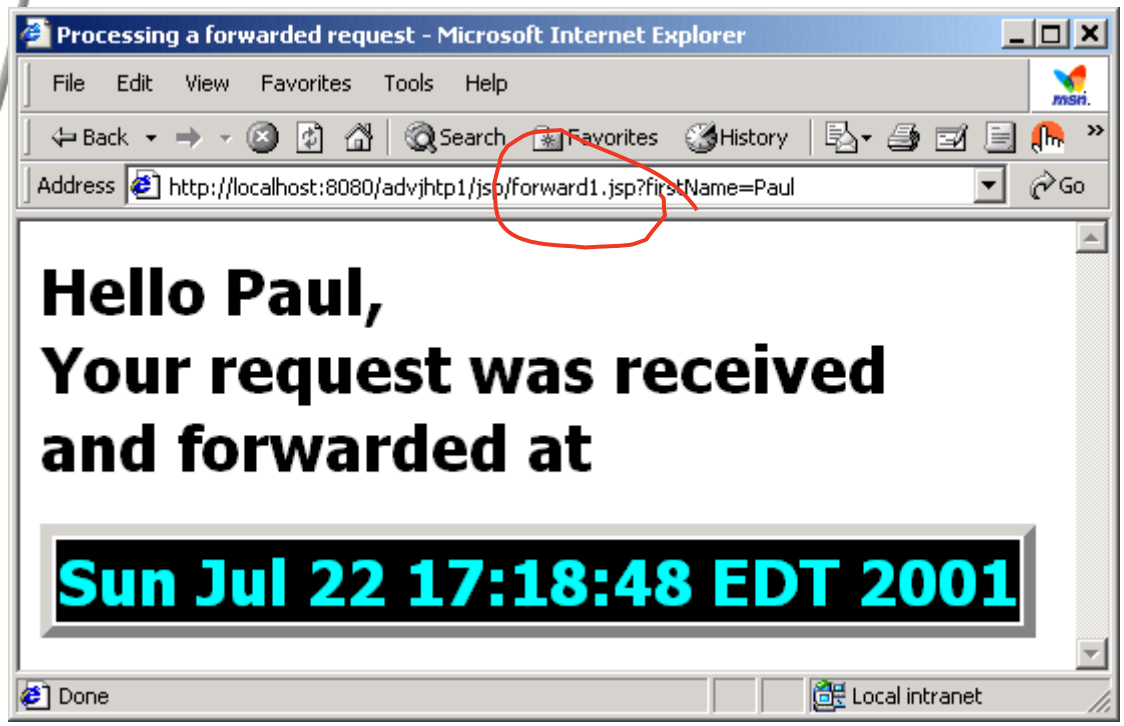
```
34        <form action = "forward1.jsp" method = "get">
35          <p>Type your first name and press Submit</p>
36
37          <p><input type = "text" name = "firstName" />
38            <input type = "submit" value = "Submit" />
39          </p>
40        </form>
41
42    <%  // continue scriptlet
43
44      }  // end else
45
46    %> <%-- end scriptlet --%>
47  </body>
48
49  </html>  <!-- end XHTML document -->
```



```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- forward2.jsp -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml"v
8
9  <head>
10   <title>Processing a forwarded request</title>
11
12   <style type = "text/css">
13     .big {
14       font-family: tahoma, helvetica, arial, sans-serif;
15       font-weight: bold;
16       font-size: 2em;
17     }
18   </style>
19 </head>
20
21 <body>
22   <p class = "big">
23     Hello <%= request.getParameter( "firstName" ) %>, <br />
24     Your request was received <br /> and forwarded at
25   </p>
26
27   <table style = "border: 6px outset;">
28     <tr>
29       <td style = "background-color: black;">
30         <p class = "big" style = "color: cyan;">
31           <%= request.getParameter( "date" ) %>
32         </p>
33       </td>
34     </tr>
35   </table>
```

Receive request from **`forward1.jsp`**, then get **`firstName`** parameter from request

Get **`data`** parameter from request



Redirezione
INTERNA

JSP e Bean

- Azione Standard **`<jsp:useBean>`**
  - Il codice contenuto negli scriptlet Java può esser inserito in una classe apposita invocata al momento del bisogno
  - Necessaria la conformità agli standard fissati per i JavaBean (architettura a componenti)

```
// classe JellyBean.java

package beans; //NOTARE CHE IL BEAN VIENE MESSO IN UN PACKAGE

public class JellyBean {
  private String color;

  public JellyBean()  {
  }

  public String getColor() {
                return color;
  }

  public void setColor(String newColor) {
                color=newColor;
  }
}
```

JavaBeans

- Ai fini dello sviluppo di servlet e JSP contano i seguenti aspetti della specifica:
  - I JavaBean non devono avere nessuna proprietà pubblica (attributi di classe privati).
  - Tutte le proprietà di un JavaBean che devono essere esposte dovranno avere metodi pubblici **get** e **set** secondo necessità.
    - Tali metodi dovranno avere nome **getVariableName** e **setVariableName**
  - Tutti i JavaBean devono avere un metodo costruttore senza argomenti

Utilizzo di un JavaBean in una JSP

- Affinchè sia visibile alla web application il bean deve essere incluso nella sotto-directory `classes` di `WEB-INF` con tutto il percorso del suo package

- Una volta creato il JavaBean, per poterlo utilizzare in una JSP vengono definiti 3 tag:
  - Per individuare o creare un JavaBean nell'area di visibilità specificata
  - Per definire una proprietà di un JavaBean (set)
  - Per leggere una proprietà di un JavaBean (get)

Caricamento di un JavaBean

- Il tag <jsp:useBean> ha la seguente sintassi:
  - <jsp:useBean id="nome" scope="page|request|session|application" class="nomeClasse" />
    - id è l'etichetta che viene assegnata al bean all'interno dell'applicazione (nome dell'istanza cui si fa riferimento)
    - scope definisce le modalità con cui l'istanza del Bean deve essere ricercata
- Esempio:
  - <jsp:useBean id="jb" scope="application" class="beans.JellyBean" />

Attributi dell'azione <jsp:useBean>

Attribute	Description
id	The name used to manipulate the Java object with actions **<jsp:setProperty>** and **<jsp:getProperty>** . A variable of this name is also declared for use in JSP scripting elements. The name specified here is case sensitive.
scope	The sc ope in which the Java object is accessible —**page**, **request**, **session** or **application** . The default scope is **page**.
class	The fully qualified class name of the Java object.
type	The type of the JavaBean. This can be the same type as the **class** attribute, a superclass of that type or an interface implemented by that type. The default value is the same as for attribute **class** . A **ClassCastEx ception** occurs if the Java object is not of the type specified with attribute **type**.

Attributes of the **<jsp:useBean>** action.

Impostazione di una proprietà JavaBean

- Una volta creato il Bean è possibile definirne le proprietà con il tag **jsp:setProperty**
  - <jsp:setProperty name="beanName" property="propertyName" value="propertyValue" />
    (assegnazione di un valore)
  - <jsp:setProperty name="beanName" property="propertyName" param="parameterName" />
    (passaggio di un parametro della richiesta)

Si noti che qui "name" è il nome dell'istanza, ovvero l'"id"
Specificato nell'azione standard <jsp:usebean …>

Impostazione di una proprietà JavaBean

- Modo rapido per definire le proprietà del Bean attraverso dati provenienti da un form:
  - <jsp:setProperty name="beanName" property="*">
- Aggiungendo "*" il metodo setProperty fa corrispondere tutti i parametri dell'oggetto richiesta ai metodi set del bean e passa i valori tra di essi.
- Definizione automatica delle proprietà possibile solo se si rispettano le convenzioni sui nomi dei metodi set e get

Attributi dell'azione <jsp:setProperty>

Attribute	Description
name	The ID of the JavaBean for which a property (or properties) will be set.
property	The name of the property to set. Specifying "*" for this attribute causes the JSP to match the request parameters to the properties of the bean. For each request parameter that matches (i.e., the name of the request parameter is identical to the bean's property name), the corresponding property in the bean is set to the value of the parameter. If the value of the request parameter is "", the proper ty value in the bean remains unchanged.
param	If request parameter names do not match bean property names, this attribute can be used to specify which request parameter should be used to obtain the value for a specific bean property. This attribute is op tional. If this attribute is omitted, the request parameter names must match bean property names.
value	The value to assign to a bean property. The value typically is the result of a JSP expression. This attribute is particularly useful for setting bean properties that cannot be set using request parameters. This attribute is optional. If this attribute is omitted, the JavaBean property must be of a data type that can be set using request parameters.

Lettura delle proprietà di un bean

- Una volta creato il Bean è possibile leggerne le proprietà con il tag **jsp:getProperty**
  - <jsp:getProperty name="beanName" property="propertyName" />

Esempio: definizione dell'attributo di un bean per l'accesso da una pagina jsp

```
<!– Pagina di scrittura delle proprietà: scrivi.jsp -->

<html>
<head>
  <title> Impostazione del colore di JellyBean </title>
</head>
<&@ page language="java" %>
<jsp:useBean id="jb" scope="session" class="beans.JellyBean" />
<body>
<form action = "leggi.jsp" method = "get">


        <strong>Scegli il colore del bean:</strong><br />
        <label>Rosso
          <input name = "newColor" type = "radio"
            value = "red" checked = "checked" />
        </label>
```

Esempio: definizione dell'attributo di un bean per l'accesso da una pagina jsp

```
        <label>Verde
          <input name = "newColor" type = "radio"
            value = "green" /></label>

        <label>giallo
          <input name = "newColor" type = "radio"
            value = "yellow" /></label>



          <input type = "submit" value = "Submit" />
</form>
</body>
```

Come ottenere una proprietà di un JavaBean

```
<!– leggi proprietà: leggi.jsp →

<html>
  <head>
    <title> Ottenere il colore di JellyBean </title>
  </head>
<%@ page language="java" %>
<jsp:useBean id="jb" scope="session" class="JellyBean" />



<body>
   <jsp:setProperty name="jb" property="color" param="newColor" />
   Il colore del bean è stato impostato al valore:</td>
   <jsp:getProperty name="jb" property="color" /></td>
</body>
</html>
```

Esercizio:

- Scrivere semplici esempi di pagine jsp contenenti:
  - La direttiva “include”
  - Le azioni standard
    - include
    - forward
    - useBean (con diversi contesti di visibilità)
    - setProperty (sia il caso in cui viene passato un valore, sia il caso in cui viene preso il valore di un parametro della richiesta)
    - getProperty
  - Descrivere (non codice) come vengono tradotte dal jsp container (consegnare solo quest’ultima parte)

JSP useBean: contesti di visibilità

- request
  - <jsp:useBean id="..." class="..." scope="request" />
- session
  - <jsp:useBean id="..." class="..." scope="session" />
- application
  - <jsp:useBean id="..." class="..." scope="application" />
- page
  - <jsp:useBean id="..." class="..." scope="page" />
    oppure
    <jsp:useBean id="..." class="..." /> (page è il contesto di default)

Visibilità di un JavaBean

- Si può accedere ad un bean da più pagine JSP, a condizione che
  - Il bean sia stato dichiarato con scope *session* e sia lo stesso utente ad accedere ad entrambe le pagine
    - Se si osserva il codice generato per le JSP si noterà una chiamata al metodo **`pageContext.getSession()`**
  - Oppure che il bean sia stato dichiarato con scope application
    - Tutti gli utenti avranno accesso alla stessa istanza del bean

Come condividere dati tra diverse risorse (1)

- Se :
  - la richiesta può essere inoltrata a più pagine di destinazione
  - richiede l'elaborazione di dati contenuti nell'oggetto **`HttpServletRequest`**
- Può essere conveniente spostare l'elaborazione dei dati nella servlet da cui la richiesta ha origine e passare alla pagina di destinazione i dati già elaborati

Come condividere dati tra diverse risorse (2)

- Primo metodo:
  - i dati preventivamente elaborati dalla servlet di origine possono essere inclusi come attributi dell'oggetto **`HttpServletRequest`**
  - **`request.setAttribute("key1", value1);`**
  - La pagina di destinazione può prelevare questi dati
  - **`Type1 value1 = (Type1)request.getAttribute("key1");`**

Come condividere dati tra diverse risorse (3)

- Secondo metodo:
  - rappresentare i dati come bean e memorizzarli nel contesto di visibilità voluto dal tag **`jsp:useBean`** per condividere i bean.

JSP useBean: contesti di visibilità

- request
  - <jsp:useBean id="..." class="..." scope="request" />
- session
  - <jsp:useBean id="..." class="..." scope="session" />
- application
  - <jsp:useBean id="..." class="..." scope="application" />
- page
  - <jsp:useBean id="..." class="..." scope="page" />
    oppure
    <jsp:useBean id="..." class="..." /> (page è il contesto di default)

Come condividere dati tra diverse risorse (4)

- Contesto di visibilità: richiesta
  - Memorizzare un dato attraverso una servlet che sia poi reperibile da una pagina JSP che elabori la stessa richiesta.
- Sintassi usata dalla servlet per memorizzare il dato

```
SomeClass value = new SomeClass(…);
request.setAttribute("key", value);
// Usa il RequestDispatcher per inoltrare la richiesta
   alla pagina JSP
```

- Sintassi usata dalla pagina JSP per recuperare il dato

```
<jsp:useBean id="key" class="somepackage.SomeClass"
    scope="request" />
```

Come condividere dati tra diverse risorse (5)

- Contesto di visibilità: sessione
  - Memorizzare un dato attraverso una servlet che sia poi reperibile da una pagina JSP all'interno della stessa sessione.
- Sintassi usata dalla servlet per memorizzare il dato

```
SomeClass value = new SomeClass(…);
HttpSession session =
  request.getSession(true);
session.setAttribute("key", value);
```

- Sintassi usata dalla pagina JSP per recuperare il dato

```
<jsp:useBean id="key"
    class="somepackage.SomeClass"
    scope="session" />
```

Come condividere dati tra diverse risorse (6)

- Contesto di visibilità: applicazione
  - Memorizzare un dato attraverso una servlet che sia poi reperibile da una pagina JSP all'interno della stessa applicazione.
- Sintassi usata dalla servlet per memorizzare il dato

```
synchronized(this)* {
  SomeClass value = new SomeClass(…);
  getServletContext().setAttribute("key",
                                   value);
  }
```

- Sintassi usata dalla pagina JSP per recuperare il dato

```
<jsp:useBean id="key"
    class="somepackage.SomeClass"
    scope="application" />
```

*la keyword "synchronized" si usa per un segmento di codice che non si vuole venga eseguito simultaneamente da due o più thread. L'oggetto tra parentesi è quello su cui si mette il lock

Come condividere dati tra diverse risorse (7)

- Sintassi usata dalla pagina JSP per recuperare il dato

```
<jsp:useBean id="key"
    class="somepackage.SomeClass"
    scope="application" />
```

- Alternativamente la pagina di destinazione può utilizzare un elemento di scripting
  - `Type1 value1 = (Type1)application.getAttribute("key1");`

Attenzione all'uso di URL relativi in una pagina JSP se questa è la destinazione di una redirezione

- L'inoltro di una richiesta attraverso il dispatcher è trasparente al client (lo stesso vale quando si usa l'azione standard "forward).
- Gli url relativi della pagina JSP vengono riferiti al percorso della servlet da cui la richiesta è stata inoltrata e non al percorso della pagina JSP di destinazione

URL relativi nelle pagine JSP

- Se nella pagina jsp cui la servlet invia la richiesta, è presente un percorso relativo:

```
<IMG SRC="foo.gif" …>
<LINK REL=STYLESHEET
                HREF="JSP-Styles.css"
                TYPE="text/css">
<A HREF="bar.jsp">…</A>
```

  - Il browser lo interpreterà come relativo all'URL della servlet di provenienza della richiesta
- Soluzione:
  - Definire il percorso relativo al server nelle pagine JSP (percorsi che iniziano con "/<contextroot>/...")
- Fare esercizio di prova

Upload di file da una pagina JSP

- Si vuole scrivere una pagina JSP che permetta all'utente di selezionare un file dal proprio disco e inviarlo al server

Upload di file e invio di messaggi di posta

Jakarta Commons

- Sotto-progetto di Jakarta per la creazione e la manutenzione di package che realizzano funzioni genericamente utili e non collegate ad uno specifico framework o prodotto
- I package del repository di Jakarta Commons sono scritti con il proposito di renderli **riutilizzabili** e di metterli **in comune**

fileupload-1.0.jar (Jakarta Commons)

- Si tratta di un package che fornisce componenti pronte per la realizzazione di upload di file dal client verso il server
- Richiede la presenza del file `fileupload-1.0.jar` nella directory /**WEB-INF/lib** della web application

  - In realtà questo file può risiedere anche nelle directory /shared/lib oppure /common/lib di Tomcat, ma è necessario essere l'amministratore del web server (es: non è sempre possibile se si utilizzano servizi di hosting)
  - Documentazione disponibile:

    http://jakarta.apache.org/commons/fileupload/

fileupload-1.0.jar e RFC 1867

- Fornisce capacità di upload a servlet e web application.
- Le classi del package fileupload permettono la gestione di richieste HTTP conformi al documento RFC 1867 , **"Form-based File Upload in HTML"**.
- Una richiesta HTTP di upload è conforme alla RFC1867 se
  1. Viene inoltrata usando il metodo POST,
  2. Specifica un content type "multipart/form-data"

1. Pagina HTML:
Creazione del form di richiesta (RFC1867)

- Una richiesta HTTP di upload di file, secondo la RFC1867, deve essere inoltrata con il metodo `POST` e codificata come `multipart/form-data`
  - enctype="multipart/form-data“

```
6 <form name="myform" method="post" action="fileuploaddemo.jsp" enctype="multipart/form-data">
7 Specifica il tuo nome: <br>
8
9 <input type="text" name="name" size="15"><br>
10
11 Specifica un file di immagine: <br>
12 <input type="file" name="myimage" ACCEPT="image/gif"><br>
13
14 Specifica un file di testo: <br>
15 <input type="file" name="myfile" ACCEPT="text/html"><br><br>
16
17 <input type="submit" name="submit" value="sottoponi il tuo file">
18 </form>
```

Form per l’inserimento di un nome, un’immagine e un documento di test

2. Creazione della pagina JSP: Controllare il formato della richiesta

- Importare i package richiesti
- Una richiesta di upload contiene una lista ordinata di *items* che sono codificati secondo la RFC 1867
- Il package Commons fileupload fornisce un metodo per verificarlo:

```
<%@ page import="org.apache.commons.fileupload.*,java.util.*,java.io.*" %>
. . . . . .
 <%
 // check that we have a file upload request
 boolean isMultipart = FileUpload.isMultipartContent(request);
```

3. Analizzare la richiesta

- Creazione di un handler per la richiesta di upload:

```
DiskFileUpload upload= new DiskFileUpload();
```

- Analisi degli item della lista

```
List items=upload.parseRequest(request);
```

4. Analisi degli elementi della lista associata alla richiesta

- Si utilizza un oggetto **`Iterator`** per analizzare gli item uno ad uno e valutare se si tratti di file oppure di field del form (con isFormField())
  - **`Tutti gli item della lista items devono implementare l'interfaccia FileItem anche se non rappresentano dei file`**

```
Iterator itr = items.iterator();
while (itr.hasNext()) {
  FileItem item = (FileItem) itr.next();
//controlla se si tratta di un campo del form oppure di un file caricato
  if (item.isFormField()) {
      //prende il nome del campo
      String fieldName=item.getFieldName();
      //se è il campo nome lo si può usare per un messaggio di benvenuto all'utente
      if (fieldName.equals("name")){
        %>
<h1>
 Ciao <%= item.getString() %>
</h1>
<%
      }
   } else  //. . . Siamo nel caso in cui si tratta proprio di un file da caricare sul server
```

5. Scrittura dei file sul server

- Richiedere esplicitamente la scrittura del file su disco, con il percorso voluto

```
} else  //. . . Siamo nel caso in cui si tratta proprio di un file da caricare sul server
{
      //Nota: item.getName() che riporta il full path
      //del file nella macchina del client come nome del file caricato.
      //per superare questo inconveniente utilizziamo fullFile.getName().

      File fullFile = new File(item.getName());
      File savedFile=new File(getServletContext().getRealPath("/"),"definitivi\\"+fullFile.getName());
      item.write(savedFile);
      }
  }
```

6. Per maggiori controlli…

- Si può porre un limite alla dimensione dei file in upload attraverso la funzione upload.setSizeMax
  - Se si imposta tale valore a -1 viene consentito l'upload di file di qualunque dimensione

```
upload.setSizeMax(500000);
```

- Per controllare la dim. massima dei file da caricare in memoria si può imporre un limite massimo oltre il quale i file vengono memorizzati sul disco in una dir temporanea

```
upload.setSizeThreshold(5000);
upload.setRepositoryPath(getServletContext().getRealPath("/")+"tempo");
```

7. Mettere insieme tutti gli elementi

```
1 <html>
2    <head>
3      <title> File html di upload </title>
4   </head>
5 <body>
6 <form name="myform" method="post" action="fileuploaddemo.jsp" enctype="multipart/form-data">
7 Specifica il tuo nome: <br>
8
9 <input type="text" name="name" size="15"><br>
10
11 Specifica l'immagine: <br>
12 <input type="file" name="myimage" ACCEPT="image/gif"><br>
13
14 Specifica il tuo file: <br>
15 <input type="file" name="myfile" ACCEPT="text/html"><br><br>
16
17 <input type="submit" name="submit" value="sottoponi il tuo file">
18
19 </form>
20 </body>
```

`Fileupload.html`

Scrive un form per l'upload di due file e la scrittura di un nome.

```
 1 <html>
 2    <head>
 3      <title> JSP per l'upload dei file del client </title>
 4   </head>
 5 <%@ page language="java" %>
 6 <%@ page import="org.apache.commons.fileupload.*,java.util.*,java.io.*" %>
 7 <body>
 8
 9 <%
10
11 boolean isMultipart = FileUpload.isMultipartContent(request);
12
13
14 DiskFileUpload upload= new DiskFileUpload();
15
16 upload.setSizeThreshold(5000);
17 upload.setRepositoryPath(getServletContext().getRealPath("/")+"tempo");
18 upload.setSizeMax(500000);
19
20
21
22 List items=upload.parseRequest(request);
23
24
25 Iterator itr = items.iterator();
26 while (itr.hasNext()) {
27    FileItem item = (FileItem) itr.next();
28    if (item.isFormField()) {
29         String fieldName=item.getFieldName();
30         if (fieldName.equals("name")){
31           %>
32 <h1>
33  Ciao <%= item.getString() %>
34 </h1>
```

`Fileuploaddemo.jsp`

Esegue l'upload di due file e scrive diversi messaggi all'utente.

```
35
36 <%
37          }
38
39        } else
40        {
41        File fullFile = new File(item.getName());
42 %>
43
44 <br>
45 Il file di cui hai richiesto l'upload è: <%= item.getName() %> .
46 <br><br>
47 Il nome del file senza percorso sul client è il seguente:
48 <%= fullFile.getName() %>
49 <br><br>
50 Lo metteremo invece in questa directory: <%= getServletContext().getRealPath("/")+"definitivi" %>
51 <br>
52
53 <%
54 File savedFile = new File(getServletContext().getRealPath("/"),"definitivi\\"+fullFile.getName());
55        item.write(savedFile);
56        }
57    }
58 %>
59 </body>
```

`Fileuploaddemo.jsp`

Esegue l'upload di due file e scrive diversi messaggi all'utente. (cont.)

Invio di una mail da una pagina JSP

- Si vuole scrivere una pagina JSP che permetta all'utente di inviare una mail attraverso il server

Mandare una e-mail da una pagina JSP

- Esistono numerosi package che fanno al caso nostro
  - Le API JavaMail sono fornite da Sun e sono open source
  - Non fanno parte del JDK
  - Per funzionare richiedono il JAF (JavaBeans Activation Framework)
  - Trovate i file javamail-1_3_1.zip e jaf-1_0_2.zip contenenti i package (jar) che vi servono sul sito del corso

Come installare i package di JavaMail

- Bisogna aggiungere i file mail.jar e activation.jar in una delle directory:
  - ...web-apps/<context-root>/WEB-INF/lib/

    per rendere visibili le classi alla vostra applicazione
  - <CATALINA_HOME>/shared/lib oppure

    <CATALINA_HOME>/common/lib

    per rendere visibili le classi a tutte le applicazioni web

Per testare l'installazione

- Scrivere una semplice direttiva di importazione in una pagina JSP

```
<html>
<head><title>Test di installazione</title></head>
<body>
<%@ page import="javax.mail.*" %>
Se leggi questo messaggio l'installazione di JavaMail <br>
è stata eseguita correttamente <br>
</body>
</html>
```

1. Definire una sessione di e-mail

- La classe `javax.mail.Session` rappresenta una sessione di mail
- Per la creazione di un oggetto della classe **`Session`** è necessario definire un oggetto **`Properties`** di configurazione
- Per l'esempio che segue basta definire una sola proprietà **`"mail.smtp.host"`** che definisce un server di posta SMTP

1. Definire una sessione di e-mail

- Definiamo una sessione di e-mail indicando il nome di un server smtp

```
Properties props = new Properties();
props.put("mail.smtp.host", "mail.fastwebnet.it");
//....altre proprietà se necessario
Session sendMailSession;
sendMailSession=Session.getInstance(props);
```

2. Definizione del messaggio di e-mail

- Classe **`Message`**: si tratta di una classe astratta, che rappresenta tutto ciò che ha a che vedere con il messaggio di e-mail
- JavaMail fornisce una sottoclasse **`MimeMessage`** per rappresentare e-mail MIME
- Per definire un nuovo oggetto **`MimeMessage`** si fa riferimento alla sessione di email corrente:

```
Message newMessage = new MimeMessage(sendMailSession);
```

2. Definizione del messaggio di e-mail

- Una volta creato l'oggetto **MimeMessage**, definire i valori dei suoi attributi (consultare la documentazione per un elenco completo)
- "**Subject**": una stringa

```
newMessage.setSubject("Subject della mail.");
```

- "**Text**": una stringa

```
newMessage.setText("Ciao questa mail proviene da una pagina JSP");
```

- "**From**": oggetto di classe **InternetAddress**

```
InternetAddress from = new InternetAddress("pluto@paperopoli.net");
newMessage.setFrom(from);
```

2. Definizione del messaggio di e-mail

- Per definire il destinatario oltre all'indirizzo serve un parametro ulteriore:
  - RecipientType.TO
  - RecipientType.CC
  - RecipientType.BCC

```
InternetAddress to = new InternetAddress("minnie@paperopoli.com");
newMessage.addRecipient(RecipientType.TO, to);
```

3. Specificare il protocollo da usare

- La classe astratta **`Transport`** serve a definire il protocollo da usare per spedire la mail (fornisce tutti i metodi per utilizzare il protocollo smtp)
- JavaMail ci risparmia l'implementazione dell'intero protocollo SMTP!
- Notare che non è necessario/possibile creare un'istanza di questa classe per spedire una mail. La classe è astratta e il metodo send() ad essa associato è un metodo "static".

Esempio di pagina JSP che spedisce una mail

```
<html><head></head>
<body>
<%@ page import="java.util.*, javax.mail.*, javax.mail.internet.*" %>
<% Properties props = new Properties ();
   props.put("mail.smtp.host", "smtp.fastwebnet.it");
   Session s = Session.getInstance(props, null);
   MimeMessage message= new MimeMessage(s);
   InternetAddress from=new InternetAddress("me@esempio.com");
   message.setFrom(from);
   InternetAddress to=new InternetAddress("you@esempio.com");
   message.addRecipient(Message.RecipientType.TO,to);
   message.setSubject("Test di posta da JavaMail");
   message.setText("Ciao questo messaggio viene da una pagina JSP");
   Transport.send(message);
%>
E' stato spedito un messaggio di email
</body></html>
```

Esempio: creare una pagina JSP per scrivere mail attraverso un form: form.html

```
<html><head><title>Form per spedire una mail</title></head>
<body>
<p><b> Una pagina per spedire le mail </b> </p>
<form action="sendmail.jsp" method="post">
   <table align="center">
      <tr><td>To: </td><td><input name="to" size="50" /></td></tr>
      <tr><td>Subject: </td><td><input name="subject" size="50"
                                 value="Mail scritta dal form" /></td></tr>
      <tr><td colspan="2" align="center">
           <textarea name="text" cols="50" rows="20">
           Ciao da JavaMail!
           </textarea>
      </td></tr>
      <tr><td colspan="2" align="center">
          <input type="submit" value="Send E-Mail"/>
      </td></tr>
   </table>
</body></html>
```

Esempio: creare una pagina JSP per scrivere mail attraverso un form: sendmail.jsp

```
<html><head></head>
<body>
<%@ page import="java.util.*, javax.mail.*, javax.mail.internet.*" %>
<% Properties props = new Properties ();
   props.put("mail.smtp.host", "smtp.fastwebnet.it");
   Session s = Session.getInstance(props, null);
   MimeMessage message= new MimeMessage(s);
   InternetAddress from=new InternetAddress("me@esempio.com");
   message.setFrom(from);
   String toAddress = request.getParameter("to");
   InternetAddress to = new InternetAddress(toAddress);
   message.addRecipient(Message.RecipientType.TO,to);
   String subject=request.getParameter("subject");
   message.setSubject(subject);
   String text= request.getParameter("text");
   message.setText(text);
   Transport.send(message); %>
E' stato spedito un messaggio di email, clicca
<a href="form.html" > qui </a> per mandarne un altro.
</body></html>
```

Come specificare destinatari multipli

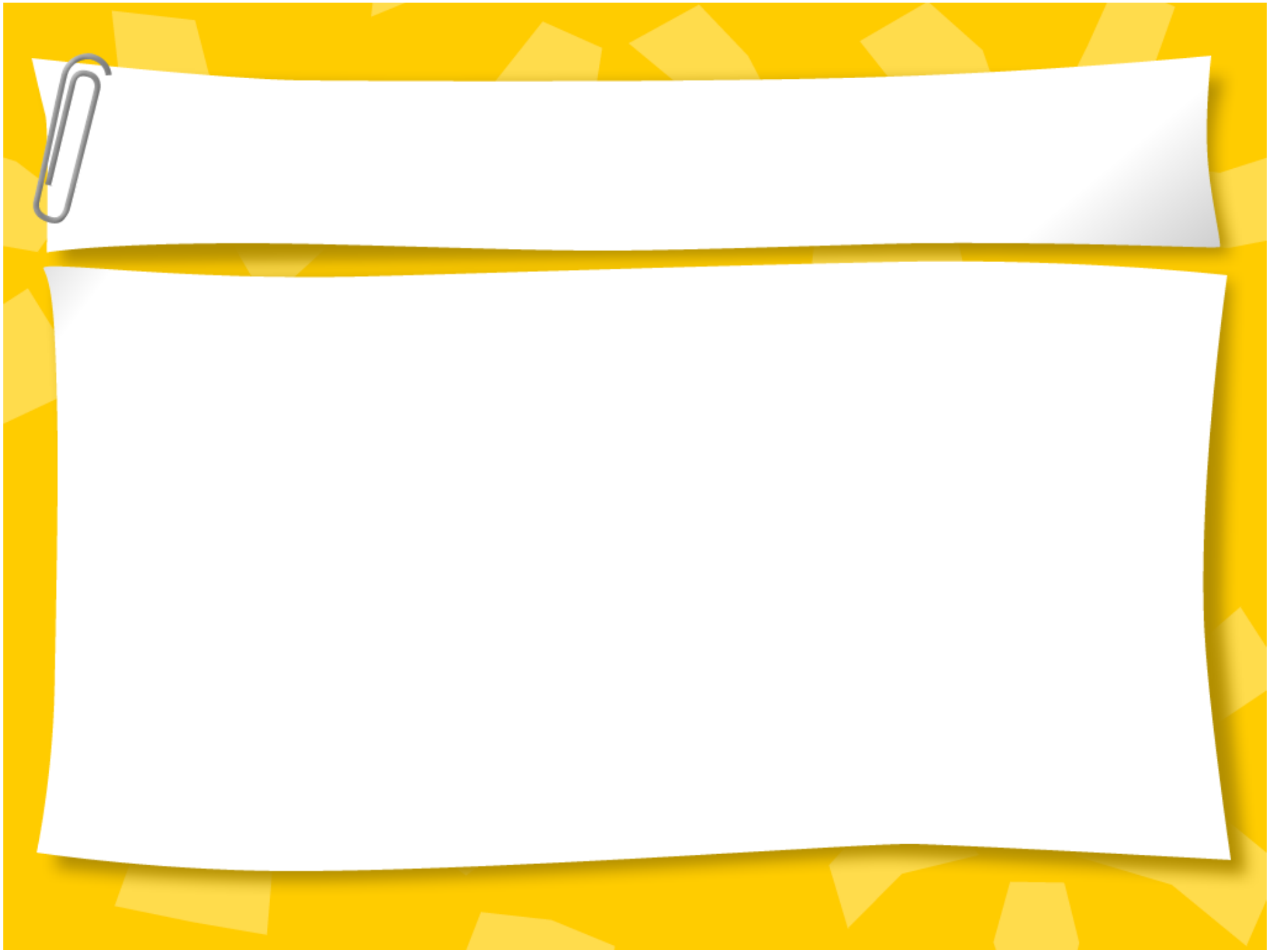
- Si possono ripetere diverse chiamate al metodo **`message.addRecipient()`**

- Si possono usare i metodi
  - setRecipients(Message.RecipientType type, Address[] addresses)
  - addRecipients(Message.RecipientType type, Address[] addresses)

Come specificare destinatari multipli (cont.)

- I metodi
  - setRecipients(Message.RecipientType type,
    String addresses)
  - addRecipients(Message.RecipientType type,
    String addresses)

  prendono come secondo parametro una stringa corrispondente ad una lista di destinatari separati da una virgola.



Tag Personalizzati

Tag personalizzati

- Sono la caratteristica più potente delle pagine jsp
  - Incapsulano funzionalità complesse
  - Permettono agli sviluppatori di software e agli autori di pagine di lavorare in modo autonomo
- Le funzionalità dei tag vengono definite all'interno di classi java che implementano l'interfaccia Tag
  - Package javax.servlet.jsp.tagext
  - (includere il file jsp-api.jar nel CLASSPATH per poter compilare le classi che definiscono i tag)

Definire ed utilizzare tag personalizzati

1. Definire, per ogni tag, una **classe handler** del tag che ne implementi le funzionalità
2. Definire un **descrittore della libreria** di tag (TLD)
3. Scrivere le pagine JSP che fanno uso della libreria dei tag personalizzati
   - Invocare la direttiva per la localizzazione del descrittore della libreria dei tag
   - Utilizzare i tag all'interno della pagina

Tag personalizzati - come si usano

1. Scrivere la classe handler del tag personalizzato
2. Scrivere il descrittore di libreria
3. Scrivere la pagina JSP che utilizza il tag personalizzato

1) La classe tag handler

- E’ la classe che contiene la logica del tag
- E’ una classe Java che implementa l’**interfaccia Tag**
- Quando il container incontra un tag personalizzato
  - Crea l’oggetto tag handler
  - Invoca i metodi dell’interfaccia Tag necessari ad attivare la logica del tag.

L'interfaccia Tag

- Le funzionalità dei tag personalizzati sono definite all'interno di classi java che implementano l'interfaccia Tag
  - In genere si estendono le classi **TagSupport** oppure **BodyTagSupport** che implementano l'interfaccia **Tag**
    - **TagSupport viene utilizzata per tag che non elaborano il contenuto del proprio body**
    - **BodyTagSupport viene utilizzata per tag che elaborano il contenuto del proprio body**
- L'interfaccia Tag fornisce i metodi che vengono invocati dal container durante l'elaborazione del tag

L'interfaccia Tag

- E' definita nel package
  - javax.servlet.jsp.tagext

- Definisce sei metodi
  1. **void setPageContext(PageContext)***
  2. **void setParent(Tag)**
  3. **int doStartTag() throws JspException**
  4. **int doEndTag() throws JspException**
  5. **void release()**
  6. **Tag getParent()**

Vedremo tra poco questi metodi nel dettaglio, per ora procediamo per esempi

I metodi 1-5 vengono invocati dal container nell'ordine indicato quando viene elaborato un tag all'interno di una pagina jsp

* Dalla documentazione: "A PageContext instance provides access to all the namespaces associated with a JSP page, provides access to several page attributes, as well as a layer above the implementation details. Implicit objects are added to the pageContext automatically".

Come definire un tag handler

- I tag handler devono obbligatoriamente essere definiti all'interno di package
- Si definisce l'implementazione dei metodi dell'interfaccia Tag che si ritengono utili all'elaborazione dei tag personalizzati
- Tutte le eccezioni che possono verificarsi durante l'elaborazione di un tag devono essere catturate e deve essere lanciata un'eccezione **JspException**

```java
// WelcomeTagHandler.java
// Custom tag handler that handles a simple tag.
package miei_tag;

// Java core packages
import java.io.*;

// Java extension packages
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class WelcomeTagHandler extends TagSupport {

   // Method called to begin tag processing
   public int doStartTag() throws JspException
   {
      // attempt tag processing
      try {
         // obtain JspWriter to output content
         JspWriter out = pageContext.getOut();

         // output content
         out.print( "Messaggio proveniente dal tag" );
      }

      // rethrow IOException to JSP container as JspException
      catch( IOException ioException ) {
         throw new JspException( ioException.getMessage() );
      }

      return SKIP_BODY;  // ignore the tag's body (alternativa a EVAL_BODY_INCLUDE)
   }
}
```

Class **`WelcomeTagHandler`** implementa l'interfaccia **`Tag`** estendendo la classe **`TagSupport`**

Il JSP container invoca il metodo **`doStartTag`** quando incontra l'apertura di un tag personalizzato

Viene utilizzato l'oggetto **`pageContext`** ereditato da TagSupport, per ottenere l'oggetto **`JspWriter`** necessario per scrivere il testo di output

Tag personalizzati - come si usano

1. Scrivere la classe handler del tag personalizzato
2. Scrivere il descrittore di libreria
3. Scrivere la pagina JSP che utilizza il tag personalizzato

Direttiva taglib

- Questa direttiva è necessaria per poter utilizzare tag personalizzati
- Individua il percorso di un **descrittore della libreria** e un **prefisso** che verrà utilizzato per accedere agli elementi della libreria

Attribute	Description
uri	Specifica il percorso relativo o assoluto del tag library descriptor (tld).
prefix	Specifica il prefisso richiesto per distinguere i tag personalizzati dai tag built-in. I prefissi **jsp**, **jspx**, **java**, **javax**, **servlet**, **sun** e **sunw** sono riservati.

```
<%@ taglib uri="/WEB-INF/tlds/mialib.tld" prefix="util" %>
```

Definire i tag personalizzati: il TLD

- Un descrittore di libreria di tag (TLD) è un documento XML che definisce una libreria di tag e i tag in essa contenuti

```
<taglib>
   <tlibversion>1.0</tlibversion>
   <jspversion>1.1</jspversion>
   <shortname>Libreria Personale</shortname>
   <info> Una semplice libreria di tag di generica utilità</info>

   <tag>
      <name>welcome</name>
      <tagclass> miei_tag.WelcomeTagHandler </tagclass>
      <bodycontent>empty</bodycontent>
      <info> Inserisce un testo di benvenuto </info>
   </tag>

</taglib>
```

File TLD: descrittore

```
<taglib>
   <tlibversion>1.0</tlibversion>
   <jspversion>1.1</jspversion>
   <shortname>Libreria Personale</shortname>
   <info> Una semplice libreria di tag di generica utilità</info>

  <tag>
     <name>welcome</name>
     <tagclass> miei_tag.WelcomeTagHandler </tagclass>
     <bodycontent>empty</bodycontent>
     <info> Scrive un messaggio di benvenuto</info>
  </tag>

</taglib>
```

- **<tlibversion>** versione della libreria
- **<jspversion>** versione della specifica JSP
- **<shortname>** e **<info>** descrizioni

File TLD: descrittore … (cont.)

- I tag sono definiti attraverso l'elemento **<tag>** che ha a sua volta due elementi obbligatori:
  - **<name>:** Il nome del tag così come viene utilizzato nella pagina JSP
  - **<tagclass>:** La classe Java che implementa la funzionalità del tag (tag handler)
- L'ulteriore elemento **<body-content>** specifica il tipo di contenuto del tag, può essere empty, tagdependent oppure JSP.
- Deve essere presente un elemento <tag> per ogni tag personalizzato della libreria

Tag personalizzati - come si usano

1. Scrivere la classe handler del tag personalizzato
2. Scrivere il descrittore di libreria
3. Scrivere la pagina JSP che utilizza il tag personalizzato

Un esempio di pagina JSP che usa un tag personalizzato

```
<!– tagdibenvenuto.jsp →
<html>
  <head>
  <title> Pagina che visualizza un messaggio di benvenuto</title>
  </head>

  <body>

  <%@ taglib uri=“/WEB-INF/tlds/mialib.tld” prefix=“util” %>

  Questo messaggio: <b><util:welcome/></b> <br>
  è stato prodotto da un tag personalizzato.

  </body>
</html>
```

Localizzazione dei file

- Copiare la classe handler del tag con tutto il suo package nella directory

  \WEB-INF\classes della web application

- Mettere il file TLD nel percorso indicato con l'attributo **uri** della direttiva **taglib**

- Mettere il file tagdibenvenuto.jsp in una qualsiasi cartella della web application

Esempio: custom tag con attributi

- I tag personalizzati possono avere un qualunque numero di attributi, obbligatori o facoltativi specificati come *attributo=valore*
  - Es: <util:iterate times="4">

1. Scrivere il tag con il relativo attributo nel file JSP
2. Aggiungere un tag di attributo al TLD
3. Implementare il metodo setAttributo nell'handler del tag

Custom tag con attributi (cont.)

- Quando il container incontra il tag crea l'oggetto tag handler e invoca i metodi setter necessari per impostare i valori degli attributi
- E' prassi comune implementare anche un metodo getter nell'handler del tag per permettere ai tag annidati di accedere alle proprietà degli altri.

Custom tag con attributi (cont.)

- Il file TLD deve contenere per ogni tag l'elenco degli attributi ad esso relativi, ciascuno specificato attraverso un elemento **<attribute>**
- L'elemento <attribute> contiene a sua volta gli elementi:
  - <name> *(nome dell'attributo)*
  - <required> *(indica se l'attributo è necessario o opzionale)*
  - <rtexprvalue> *(indica se l'attributo deve essere specificato come stringa o se è permessa l'elaborazione a tempo di esecuzione)*

```jsp
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- customTagAttribute.jsp              -->
<!-- JSP that uses a custom tag to output content. -->

<%-- taglib directive --%>
<%@ taglib uri = "mialib.tld" prefix = "util" %>

<html>

   <head>
      <title>Utilizzo di tag personalizzati con attributi</title>
   </head>

   <body>
      <p>Uso di un attributo dichiarato come stringa</p>
      <h1>
         <util:welcome2 firstName = "Paul" />
      </h1>

      <p>Valutazione del valore di un attributo a tempo di esecuzione</p>
      <h1>
         <%-- scriptlet to obtain "name" request parameter --%>
         <%
            String name = request.getParameter( "name" );
         %>

         <util:welcome2 firstName = "<%= name %>" />
      </h1>
   </body>

</html>
```

Pagina JSP che usa un tag con un attributo firstName

Viene usato il tag personalizzato **`welcome2`** per inserire un testo nella JSP concatenando il valore dell'attributo **`firstName`**

Classe handler di un tag con un attributo
firstName

```
1   // Welcome2TagHandler.java
2   // Classe tag handler che gestisce un tag con un attributo
3   package miei_tag;
4
5   // Java core packages
6   import java.io.*;
7
8   // Java extension packages
9   import javax.servlet.jsp.*;
10  import javax.servlet.jsp.tagext.*;
11
12  public class Welcome2TagHandler extends TagSupport {
13     private String firstName = "";
14
15     // Method called to begin tag processing
16     public int doStartTag() throws JspException
17     {
18        // attempt tag processing
19        try {
20           // obtain JspWriter to output content
21           JspWriter out = pageContext.getOut();
22
23           // output content
24           out.print( "Hello " + firstName +
25              ", <br />Welcome to JSP Tag Libraries!" );
26        }
27
28        // rethrow IOException to JSP container as JspException
29        catch( IOException ioException ) {
30           throw new JspException( ioException.getMessage() );
31        }
32
33        return SKIP_BODY;  // ignore the tag's body
34     }
35
```

Definizione dell'attributo **`firstName`**

Uso dell'attributo **`firstName`** per produrre un messaggio di output attraverso il tag personalizzato

```
36      // set firstName attribute to the users first name
37      public void setFirstName( String username )
38      {
39         firstName = username;
40      }
41   }
```

Metodo *setter* per l'attributo **`firstName`**

File TLD per la definizione di un tag con un attributo firstName

```
1  <!-- A tag with an attribute -->
2     <tag>
3        <name>welcome2</name>
4
5        <tagclass>
6           miei_tag.Welcome2TagHandler
7        </tagclass>
8
9        <bodycontent>empty</bodycontent>
10
11       <info>
12          Inserisce un messaggio di benvenuto
13          usando l'attributo "name" per inserire il nome dell'utente
14       </info>
15
16       <attribute>
17          <name>firstName</name>
18          <required>true</required>
19          <rtexprvalue>true</rtexprvalue>
20       </attribute>
21    </tag>
```

Si introduce l'elemento **`attribute`** per definire le modalità di uso dell'attributo **`firstName`**

Esempio 1

- Form contenente diversi campi di testo.
- Se il form non è stato compilato correttamente viene riproposto all'utente e i campi del form già compilati vengono rivisualizzati in modo che l'utente debba immettere solo quelli mancanti.

Esempio 1 (continua)

- Un primo tentativo potrebbe essere:

  <input type="text"

  size=15

  nome="firstName"

  value="<%=request.getParameter('firstName')%>">

- Inconveniente:
  - se nessun parametro di richiesta corrisponde al nome dei campi (es. prima visualizzazione del form) viene visualizzato il valore null
- Soluzione: implementare un tag personalizzato che
  - resituisce il valore del parametro della richiesta, se esiste, e una stringa vuota in caso contrario

Esempio 1: /register.jsp

```
<%@ taglib uri="WEB-INF/tlds/html.tld" prefix="form_util" %>
. . .
<table>
   <tr>
      <td> Nome: </td>
      <td> <input type="text" size=15 name=firstName"
            value="<form_util:requestParameter property='firstName'/>">
      </td>
   </tr>
   <tr>
      <td> Cognome: </td>
      <td> <input type="text" size=15 name=lastName"
            value="<form_util:requestParameter property='lastName'/>">
      </td>
   </tr>
   <tr>
      <td> Email: </td>
      <td> <input type="text" size=25 name=emailAddress"
            value="<form_util:requestParameter property='emailAddress'/>">
      </td>
   </tr>
</table>
. . .
```

Esempio 1: /WEB-INF/tlds/html.tld

```
. . .
<taglib>
   . . .
   <tag>
      <name>requestParameter</name>
      <tagclass>miei_tag.GetRequestParameterTag</tagclass>
      <bodycontent>empty</bodycontent>

      <attribute>
          <name>property</name>
          <required>true</required>
          <rtexprvalue>true</rtexprvalue>
      </attribute>
   </tag>
</taglib>
```

Esempio 1:
/WEB-INF/classes/miei_tag/GetRequestParameterTag.java

```
package miei_tag;
import javax.servlet.ServletRequest;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
public class GetRequestParameterTag extends TagSupport {
  private String property;
  public void setProperty(String valore) {
  this.property=valore;
  }
  public int doStartTag() throws JspException {
    ServletRequest req=pageContext.getRequest();
    String value= req.getParameter(property);
    try {
       pageContext.getOut().print(value==null? "":value);
    }
    catch (java.io.IOException ex) {
       throw new JspException(ex.getMessage());
    }
    return SKIP_BODY;
  }
}
```

Documentazione sui tag personalizzati

- Sul vostro pc all'indirizzo:

http://127.0.0.1:8080/tomcat-docs/jspapi/index.html

Ricapitolando:
per definire una libreria di "custom tag" (tag personalizzati)

- classe Handler di un Tag
  - Codice Java che stabilisce come tradurre il tag in codice
  - Implementa l'interfaccia javax.servlet.jsp.tagext.Tag oppure javax.servlet.jsp.tagext.BodyTag
  - Estende TagSupport or BodyTagSupport
  - Va messa in …/WEB-INF/classes insieme a servlet e beans (in un opportuno package)
- file TLD (= Tag Library Descriptor)
  - File XML che descrive il nome del tag, i suoi attributi, e la classe handler che lo implementa
  - Può essere messo insieme al file JSP che usa la libreria oppure ad un altro URL
- file JSP
  - Importa una libreria di tag (specificando l'URL di un file TLD in una direttiva taglib)
  - Definisce il prefisso da associare ai tag della libreria (specificando il prefisso nella direttiva taglib)
  - Utilizza i tag

Tag personalizzati e contesto della pagina (**PageContext**)

- I tag hanno accesso alle informazioni della pagina tramite l'oggetto **pageContext** (istanza di **PageContext**)
- La classe **PageContext** prevede un insieme di metodi per l'accesso agli oggetti impliciti nell'ambito di una pagina (come richiesta, session, scrittore per produrre l'output sulla pagina ecc.)

L’interfaccia Tag

1. **void setPageContext(PageContext)**
2. **void setParent(Tag)**
3. **int doStartTag() throws JspException**
4. **int doEndTag()**
5. **void release()**

6. **Tag getParent()**

- I metodi del primo gruppo (1-5) vengono chiamati nell’ordine dal contenitore di servlet (per questo si parla di *ciclo di vita di un tag*)

Ciclo di vita di un tag (interf. Tag) (1/3)

1. Il metodo **setPageContext(PageContext)** configura il contesto di pagina associato al tag
2. Il metodo **setParent(Tag)** associa un genitore al tag:
   - Tutti i tag hanno un genitore, che è **null** per i tag di livello superiore ed è il tag contenitore per i tag annidati

```
<esempio:tag_esterno>
        <esempio:tag_intermedio>
                <esempio:tag_interno>
                …
                </esempio:tag_interno>
        </esempio:tag_intermedio>
</esempio:tag_esterno>
```

tag_intermedio è genitore di tag_interno

tag_esterno è genitore di tag_intermedio

Il genitore di tag_esterno è null

- Entrambi questi metodi sono rivolti a coloro che implementano il contenitore delle servlet e non agli sviluppatori JSP

Ciclo di vita di un tag (interf. Tag) (2/3)

3. Se il tag prevede degli attributi, vengono invocati tutti i metodi necessari a configurarne i valori (set)
4. Il metodo **doStartTag()** viene invocato subito dopo i primi due (**setPageContext** e **setParent**) e gli eventuali metodi set
   - Tale metodo restituisce un valore intero che condiziona l'elaborazione del tag
     - **SKIP_BODY**: il corpo del tag non viene considerato
     - **EVAL_BODY_INCLUDE**: il corpo del tag deve essere trascritto invariato

Ciclo di vita di un tag (interf. Tag) (3/3)

4. Il metodo **doEndTag()** viene chiamato in corrispondenza del tag di chiusura
   - Tale metodo restituisce un valore intero che condiziona l'elaborazione della parte di pagina che segue il tag
     - **SKIP_PAGE**: la parte di pagina oltre il tag di chiusura viene ignorata
     - **EVAL_PAGE**: la parte di pagina oltre il tag di chiusura viene considerata
6. Il metodo **release()** rilascia del risorse dell'handler del tag

La classe `TagSupport`

- La classe **`tagSupport`** implementa l'interfaccia Tag e aggiunge alcuni attributi e metodi,

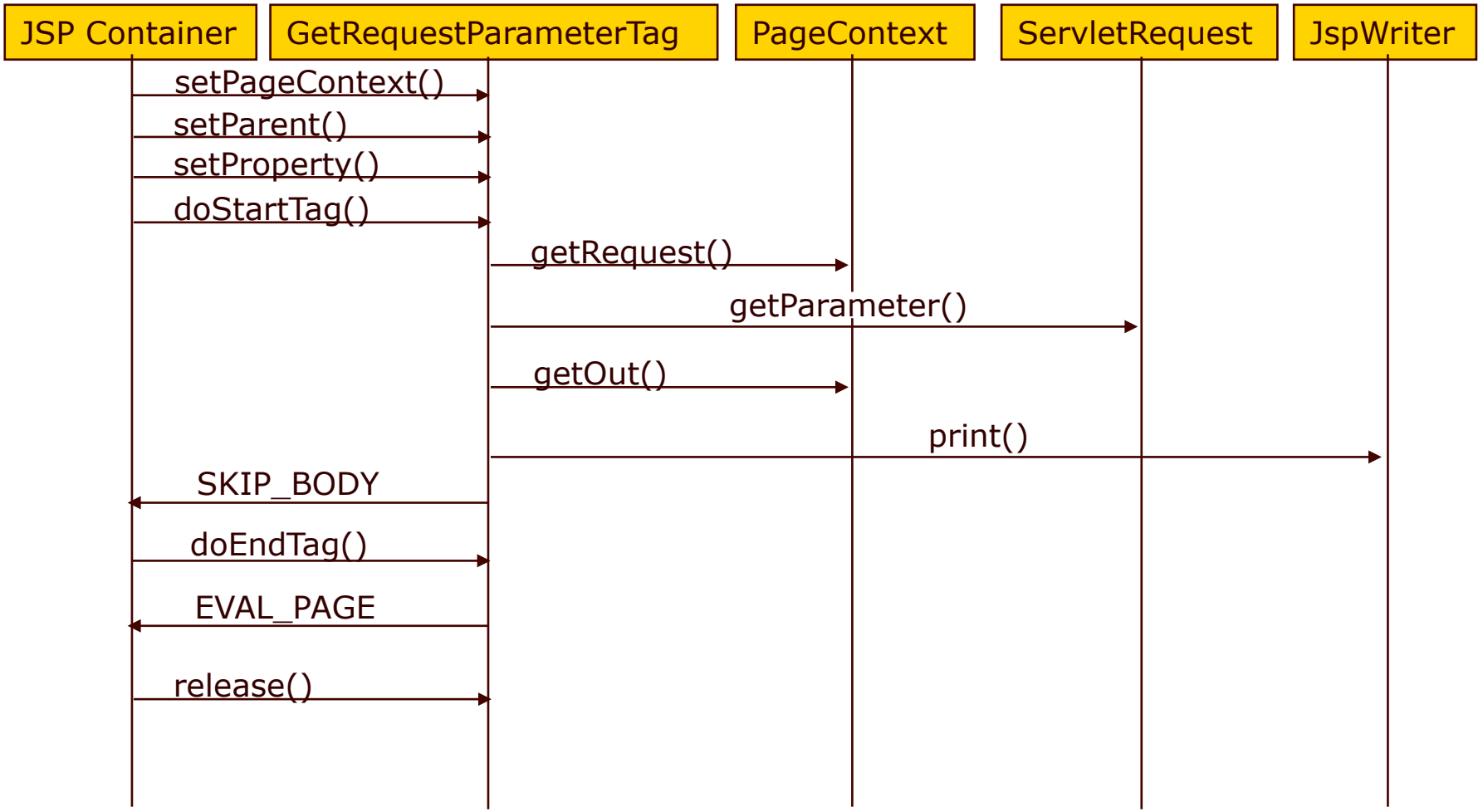
fare riferimento alla documentazione

```
protected String id
protected PageContext pageContext

static Tag findAncestorWithClass(Tag,Class)
Object getValue(String key)
void  setValue(String key, Object value)
void removeValue(String key)
Enumeration getValues()

String getID()
void setID()
```

Ciclo di vita del tag dell'esempio sugli elementi del form (1/2)

```
package miei_tag;
import javax.servlet.ServletRequest;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
public class GetRequestParameterTag extends TagSupport {
  private String property;
  public void setProperty(String valore) {
  this.property=valore;
  }
  public int doStartTag() throws JspException {
    ServletRequest req=pageContext.getRequest();
    String value= req.getParameter(property);
    try {
       pageContext.getOut().print(value==null? "":value);
    }
    catch (java.io.IOException ex) {
       throw new JspException(ex.getMessage());
    }
    return SKIP_BODY;
  }
}
```

Ciclo di vita del tag dell'esempio sugli elementi del form (2/2)



Tag che includono il corpo

- Non richiedono di estendere BodyTagSupport, se il corpo del tag non richiede elaborazione

```
<prefix:tagName>
        JSP Content
</prefix:tagName>
<prefix:tagName att1="val1" ... >
        JSP Content
</prefix:tagName>
```

Tag che includono il corpo: la classe Tag Handler

- **doStartTag**
  - Per includere il corpo restituisce **EVAL_BODY_INCLUDE** invece di **SKIP_BODY**
- **doEndTag**
  - Metodo che definisce azioni da intraprendere dopo l'inclusione del corpo
  - Restituisce **EVAL_PAGE** oppure **SKIP_PAGE** a seconda dei casi

Esempio 2: HeadingTag.java

```
package tags;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;

public class HeadingTag extends TagSupport {
  private String bgColor; // Un attributo obbligatorio
  private String border = null;
  ... //altri attributi

  public void setBgColor(String bgColor) {
    this.bgColor = bgColor;
  }

  public void setBorder(String border) {
    this.border = border;
  }
 ... //altri metodi setter per gli altri attributi
```

Esempio 2: HeadingTag.java (Continua)

```
public int doStartTag() {
    try {
      JspWriter out = pageContext.getOut();
      out.print("<TABLE BORDER=" + border +
                " BGCOLOR=\"" + bgColor + "\"" +
                " ALIGN=\"" + align + "\"");
      if (width != null) {
        out.print(" WIDTH=\"" + width + "\"");
      }
      ...
    } catch(IOException ioe) {
      System.out.println("Error in HeadingTag: " + ioe);
    }
    return(EVAL_BODY_INCLUDE); // Include il corpo del tag
  }
```

Esempio 2: HeadingTag.java (Continua)

```
public int doEndTag() {
    try {
      JspWriter out = pageContext.getOut();
      out.print("</TABLE>");
    } catch(IOException ioe) {
      System.out.println("Error in HeadingTag: " + ioe);
    }
    return(EVAL_PAGE); // Continue with rest of JSP page
  }
```

Tag che includono il corpo: Tag Library Descriptor (TLD)

L'unica novità
(rispetto ai tag che non includono il corpo)
è nell'elemento bodycontent

- Deve essere JSP invece di **empty:**

```
<tag>
  <name>…</name>
  <tagclass>…</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>…</info>
</tag>
```

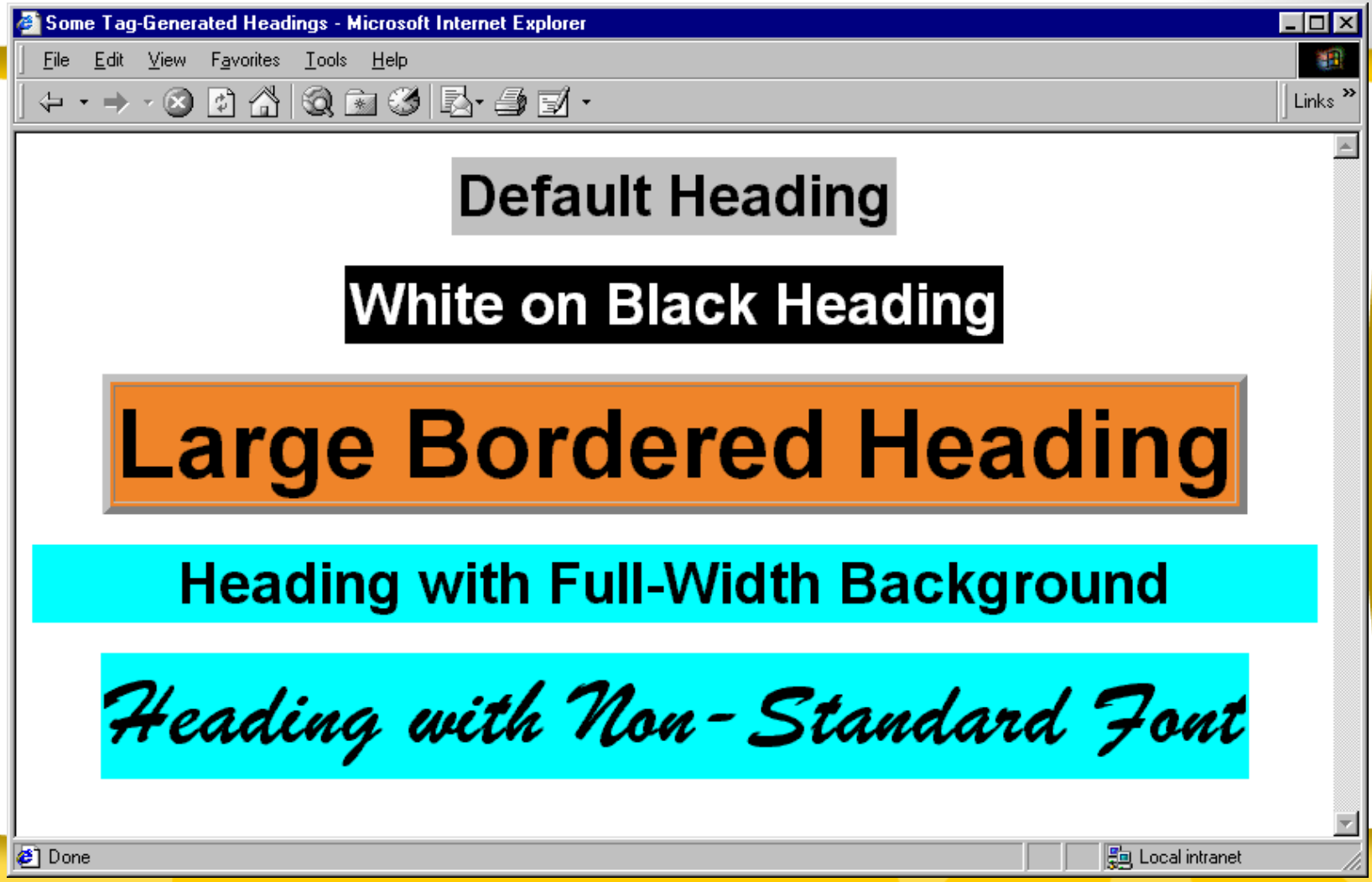
Esempio:
File TLD per il tag della classe HeadingTag

```
...
<taglib>
  <tag>
    <name>heading</name>
    <tagclass>tags.HeadingTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Scrive una tabella di 1 cella per definire un'intestazione</info>
    <attribute>
      <name>bgColor</name>
      <required>true</required> <!-- bgColor obbligatorio -->
    </attribute>
    <attribute>
      <name>border</name>
      <required>false</required> <!-- la dim del bordo della tabella è opzionale -->
    </attribute>
    ...
  </tag>
 </taglib>
```

Uso del tag “heading” in una pagina JSP

```
<%@ taglib uri="libreria.tld" prefix="formato" %>
<formato:heading bgColor="#C0C0C0">
Default Heading
</formato:heading>
<P>
<formato:heading bgColor="BLACK" color="WHITE">
White on Black Heading
</formato:heading>
<P>
<formato:heading bgColor="#EF8429" fontSize="60" border="5">
Large Bordered Heading
</formato:heading>
<P>
<formato:heading bgColor="CYAN" width="100%">
Heading with Full-Width Background
</formato:heading>
...
```

Uso del tag “heading”



Esempio 3: tag per autenticazione

- Definire un bean per una piccola base dati di utenti in cui
  - il singolo utente sia rappresentato da un bean
  - la stessa base dati sia un bean con metodi per l'aggiunta e la ricerca di un utente in un vettore di bean utente
- Definire un meccanismo di sicurezza basato sull'uso di un tag personalizzato

  <security:enforceLogin loginPage="/login.jsp"
  errorPage="/error.jsp">

  il cui funzionamento sia di rimandare alle pagine di login e di errore a seconda dei casi, e che mostri il seguito della pagina solo nel caso in cui l'utente sia stato correttamente autenticato

Rappresentazione del singolo utente

/WEB-INF/classes/beans/User.java

```
package beans;

public class User implements java.io.Serializable {
   private final String userName, password, hint;

   //costruttore
   public User(String userName, String password, String hint) {
            this.userName=userName;
            this.password=password;
            this.hint=hint;
            }

   //metodi getter
   public String getUserName() {return userName;}
   public String getPassword() {return password;}
   public String getHint() {return hint;}

   //metodo che controlla se il bean utente ha il nome uname e una password pwd
   public boolean equals(String uname, String pwd) {
            return (getUserName().equals(uname) && getPassword().equals(pwd));
   }
}
```

Rappresentazione del singolo utente (cont.)

- Gli utenti hanno tre proprietà: nome, password e suggerimento
- Gli attributi del bean utente non possono essere modificati – le proprietà sono impostate dal costruttore (uso della keyword **final**)

- Se un oggetto non può essere modificato dopo la creazione, non possono verificarsi incoerenze dovute all'accesso concorrente di più thread.

Database di accesso

/WEB-INF/classes/beans/LoginDB.java

```
package beans;

import java.util.Iterator;
import java.util.Vector;

public class LoginDB implements java.io.Serializable {
   private Vector users = new Vector();
   private User[] defaultUsers = {
      new User("Picasso","Pablo","Il mio nome"), };
  //costruttore (aggiunge al vettore users tutti gli utenti di default)
  public LoginDB() {
    for (int i=0;i<defaultUsers.length; i++)
      users.add(defaultUsers[i]);
    }
   //metodo adder (aggiunge al vettore users il bean utente con attributi dati)
   public void addUser(String uname, String pwd, String hint) {
         users.add(new User(uname,pwd,hint));
         }
   . . .
```

Database di accesso (cont.)

/WEB-INF/classes/beans/LoginDB.java

```
. . . //continua def della classe LoginDB

   //metodo di ricerca del bean utente identificato da nome e password
   public User getUser(String uname, String pwd) {
         Iterator it = users.iterator();
         User bean;
         synchronized (users) {
                  while (it.hasNext()) {
                     bean = (User)it.next();
                     if (bean.equals(uname,pwd))
                           return bean;
                     }
                  }
          return null;
   }

. . .
```

Database di accesso (cont.)

/WEB-INF/classes/beans/LoginDB.java

```
. . . //continua def della classe LoginDB

   //metodo di ricerca del suggerimento di un bean utente identificato da
   // un certo nome

   public String getHint(String uname) {
         Iterator it = users.iterator();
         User bean;
         synchronized (users) {
                  while (it.hasNext()) {
                     bean = (User) it.next();
                     if (bean.getUserName().equals(uname))
                           return bean.getHint();
                     }
                  }
          return null;
   }

}
```

Una pagina protetta: protectedPage.jsp

```
<html><head><title> Una pagina protetta </title></head>
<%@taglib uri="/WEB-INF/tlds/security.tld" prefix="security" %>
<body>

<security:enforceLogin loginPage="/login.jsp"
                       errorPage="/error.jsp" />

<jsp:useBean id="user" type="beans.User" scope="session" />

Questa è una pagina protetta. Benvenuto <%= user.getUserName() %>
</body>
</html>
```

Tag Library Descriptor: security.tld

```
<taglib><tlibversion>1.0</tlibversion><jspversion>1.1</jspversion>
   <tag>
      <name>enforceLogin</name>
      <tagclass>tags.EnforceLoginTag</tagclass>
      <bodycontent>JSP</bodycontent>
      <attribute>
          <name> loginPage </name>
          <required> true </required>
          <rtexprvalue> true </rtexprvalue>
      </attribute>
      <attribute>
          <name> errorPage </name>
          <required> false </required>
          <rtexprvalue> true </rtexprvalue>
      </attribute>
    </tag>
  <tag> <name>showErrors</name>
          <tagclass>tags.ShowErrorsTag</tagclass>
          <bodycontent>empty</bodycontent>
  </tag>
</taglib>
```

Classe handler del tag `EnforceLogin`

/WEB-INF/classes/tags/EnforceLoginTag.java

```
package tags;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.TagSupport;

public class EnforceLoginTag extends TagSupport {
   private String loginPage, errorPage;
   public void setLoginPage (String loginPage) {
     this.loginPage=loginPage;
     }
   public void setErrorPage (String errorPage) {
     this.errorPage=errorPage;
     }
. . .
```

Classe handler del tag **EnforceLogin**

/WEB-INF/classes/tags/EnforceLoginTag.java

```
. . .//il metodo doEndTag decide se permettere la visualizzazione del resto
     //della pagina
public int doEndTag() throws JspException {
   HttpSession session = pageContext.getSession();
   HttpServletRequest req = (HttpServletRequest)pageContext.getRequest();
   //usa una var protectedPage per memorizzare la pagina richiesta
   //cui fare ritorno dopo l'eventuale redirezione verso la login-page
   String protectedPage = req.getRequestURI();
   if (session.getAttribute("user")==null) {
      session.setAttribute("login-page", loginPage);
      session.setAttribute("error-page", errorPage);
      session.setAttribute("protected-page", protectedPage);
      try {
          pageContext.forward(loginPage);
          return SKIP_PAGE;
          }
      catch (Exception ex) {
            throw new JspException(ex.getMessage());
            }
      }
return EVAL_PAGE; //eseguito se l'attributo user viene trovato nella sessione
}
```

Classe handler del tag **EnforceLogin**

/WEB-INF/classes/tags/EnforceLoginTag.java

```
. . .

        public void release() {
                loginPage=errorPage=null;
                }
}
```

/login.jsp

```
<html><head><title> Login Page </title></head>
<%@taglib uri="/WEB-INF/tlds/security.tld" prefix="security" %>
<body>
  <font size=4 color=red><security:showErrors /> </font>
  <p><font size=5 color=blue">Please login </font> <hr>
  <form action="<%=response.encodeURL("authenticate") %>" method="POST">
  <table>
    <tr>
       <td>Name: </td>
       <td><input type="text" name="userName" /> </td>
    </tr> <tr>
       <td>Password: </td>
       <td><input type="password" name="password" size="8" /> </td>
    </tr> </table>
    <input type="submit" value="login">
   </form> </p>

   Ricorda che un nome valido è: Picasso e password: Pablo
</body></html>
```

/WEB-INF/web.xml

```
<servlet>
       <servlet-name>authenticate</servlet-name>
       <servlet-class> AuthenticateServlet </servlet-class>
</servlet>

<servlet-mapping>
        <servlet-name> authenticate </servlet-name>
        <url-pattern>/authenticate</url-pattern>
</servlet-mapping>
```

/WEB-INF/classes/AuthenticateServlet.java

```
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import beans.LoginDB;
import beans.User;

public class AuthenticateServlet extends HttpServlet {
   private LoginDB loginDB;

   public void init(ServletConfig config) throws ServletException {
      super.init(config);
      loginDB=new LoginDB();
      }
. . .
```

/WEB-INF/classes/AuthenticateServlet.java

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws IOException, SerlvetException {
   HttpSession session=req.getSession();
   String uname = req.getParameter("userName");
   String pwd = req.getParameter("password");
   User user = loginDB.getUser(uname,pwd);
//ricerca nella base dati il bean utente con nome e password del form

   if (user != null) { //authorized
     String protectedPage = (String)session.getAttribute("protected-page");
     session.removeAttribute("login-page");
     session.removeAttribute("error-page");
     session.removeAttribute("protected-page");
     session.removeAttribute("login-error");
     //inserisce il bean utente nella sessione
     session.setAttribute("user",user);
     res.sendRedirect(res.encodeURL(protectedPage));
   }
. . .
```

/WEB-INF/classes/AuthenticateServlet.java

```
. . .

//l'utente con i dati digitati nel form non è stato trovato nella base dati
else {//not authorized
   String loginPage = (String) session.getAttribute("login-page");
   String errorPage = (String) session.getAttribute("error-page");
   String forwardTo = errorPage!=null?errorPage:loginPage;
   session.setAttribute("login-error", "Username and pass are not valid");

   //la richiesta viene rediretta alla pagina di errore se è stata
   //configurata, altrimenti alla pagina di login
   getServletContext().getRequestDispatcher(
           res.encodeURL(forwardTo)).forward(req,res);
    }
  }
}
```

/error.jsp

```
<html><head><title> Login Page </title></head>
<%@taglib uri="/WEB-INF/tlds/security.tld" prefix="security" %>
<body>
  <font size=4 color=red>Login Failed because:</font>
   <security:showErrors/> </font>
   Click <a href="login.jsp"> here </a> to retry login.
</body></html>
```

Classe handler del tag **ShowErrors**

/WEB-INF/classes/tags/ShowErrorsTag.java

```
package tags;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.TagSupport;

public class ShowErrorsTag extends TagSupport {
   public int doStartTag() throws JspException {
      String error = (String)pageContext.getSession().
                                    getAttribute("login-error");
      if (error!=null) {
         try {
              pageContext.getOut().print(error);
             }
         catch (java.io.IOException ex) {
            throw new JspException(ex.getMessage());
         }
      }
   return SKIP_BODY;
   }
}
```

Tag che elaborano il proprio contenuto

Interfaccia `BodyTag` (1/6)

- Gli handler di tag con corpo che implementano l'interfaccia `BodyTag` dispongono di due funzionalità mancanti agli altri tag:
  - Possono contenere codice iterativo
  - Possono manipolare il contenuto del loro corpo

- L'interfaccia **`BodyTag`** estende l'interfaccia `Tag` (estende **`IterationTag`** che estende **`Tag`**) definendo i metodi elencati di seguito:

```
1. void doInitBody()
2. int doAfterBody()
3. void setBodyContent()
```

Interfaccia `BodyTag` (2/6)

- Il metodo `doStartTag()` restituisce un valore intero che condiziona l'elaborazione del tag
  - `SKIP_BODY`: il corpo del tag non deve essere considerato
  - `EVAL_BODY_INCLUDE`: il corpo del tag viene elaborato come nell'interfaccia **`IterationTag`:**
    - **Il corpo viene valutato e passato in output**
    - **Viene invocato il metodo doAfterBody() (eseguito per una o più iterazioni)**
    - **Viene invocato doEndTag()**

Interfaccia **BodyTag** (3/6)

- **EVAL_BODY_BUFFERED**: il corpo del tag viene elaborato e viene creato un oggetto **BodyContent** (sottoclasse di **JspWriter**) utilizzato come oggetto **out**.

- NB: L'oggetto **BodyContent** viene creato esclusivamente se il metodo doStartTag restituisce **EVAL_BODY_BUFFERED**.

Interfaccia **BodyTag** (4/6)

- Il metodo **`setBodyContent()`** configura le proprietà dell'oggetto **`BodyContent`**.
  - Questo metodo non viene invocato per tag vuoti e per i quali il metodo doStartTag() abbia restituito **`SKIP_BODY`** o **`EVAL_BODY_INCLUDE`**.
  - Quando viene invocato, il valore dell'oggetto implicito out viene sostituito nell'oggetto pageContext.
- Il metodo **`doInitBody()`** viene invocato dal container dopo **`setBodyContent`** e prima che il corpo del tag venga valutato per la prima volta.
  - Questo metodo non viene invocato per tag vuoti e per i quali il metodo doStartTag() abbia restituito **`SKIP_BODY`** o **`EVAL_BODY_INCLUDE`**.

Interfaccia `BodyTag` (5/6)

- Il metodo in cui bisogna definire il comportamento per tag che devono modificare/elaborare il corpo è

  `doAfterBody()`

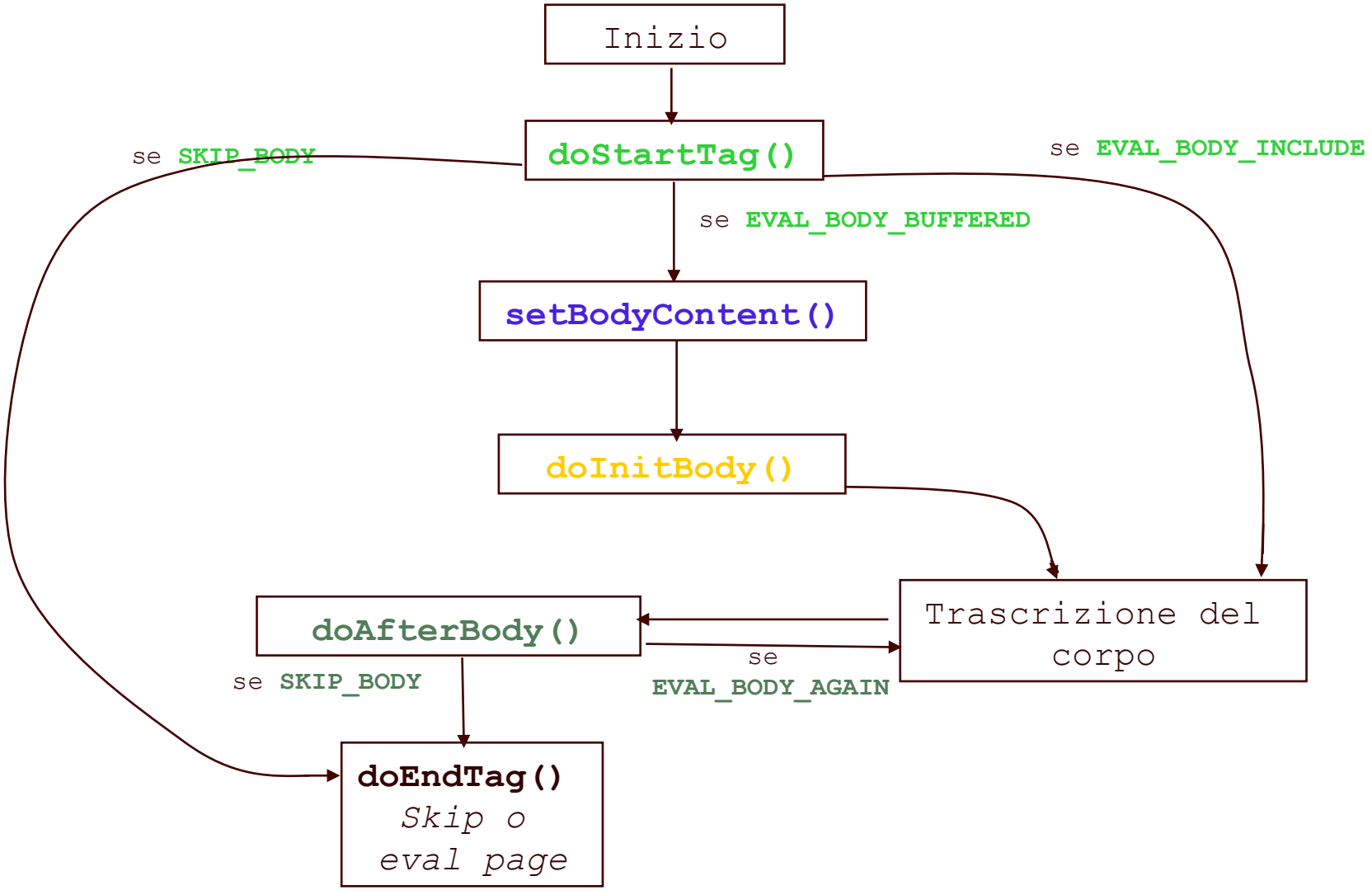
  – Tale metodo restituisce un valore intero che condiziona l'elaborazione del tag

    - `SKIP_BODY`: il corpo del tag non deve essere considerato
    - `EVAL_BODY_AGAIN`: il corpo del tag deve essere valutato nuovamente

Interfaccia `BodyTag` (6/6)

- Il metodo `doEndTag()` viene chiamato come per l'interfaccia **`Tag`** quando il container incontra il tag di chiusura
  - Tale metodo restituisce un valore intero che condiziona l'elaborazione della parte di pagina che segue il tag
    - `SKIP_PAGE`: la parte di pagina oltre il tag di chiusura viene ignorata
    - `EVAL_PAGE`: la parte di pagina oltre il tag di chiusura viene considerata

Ciclo di vita di un tag che implementa l'interfaccia **BodyTag**



Ciclo di vita di un tag che implementa l'interfaccia **BodyTag**

- Il contenitore di servlet richiama i metodi dell'interfaccia **BodyTag** nel seguente modo:

```
if (tag.doStartTag() == EVAL_BODY_BUFFERED) {
    tag.setBodyContent(bodyContent);
    . . .
   tag.doInitBody();}
if ((tag.doStartTag() == EVAL_BODY_INCLUDE)||
    (tag.doStartTag() == EVAL_BODY_BUFFERED))
{
    do {
      // valuta il corpo del tag
   }
   while (tag.doAfterBody() == EVAL_BODY_AGAIN);
}
tag.doEndTag();
```

La classe `BodyTagSupport` (1/2)

- La classe `BodyTagSupport` estende la classe **`TagSupport`** e implementa l'interfaccia **`BodyTag`**
- Questa classe introduce i nuovi metodi
  - **`BodyContent`** `getBodyContent()`

    restituisce il contenuto del corpo di un tag
  - **`JspWriter`** `getPreviousOut()`

    restituisce lo scrittore associato al tag genitore o la variabile implicita **`out`** se il tag è di livello superiore.

La classe `BodyTagSupport` (2/2)

- Per impostazione predefinita le estensioni di **`BodyTagSupport`** valutano il corpo del tag una volta soltanto
- Valori predefiniti restituiti dai metodi **`BodyTagSupport`**
  - **`doStartTag()`:** **`EVAL_BODY_BUFFERED`**
  - **`doAfterBody()`:** **`SKIP_BODY`**
  - **`doEndTag()`:** **`EVAL_PAGE`**

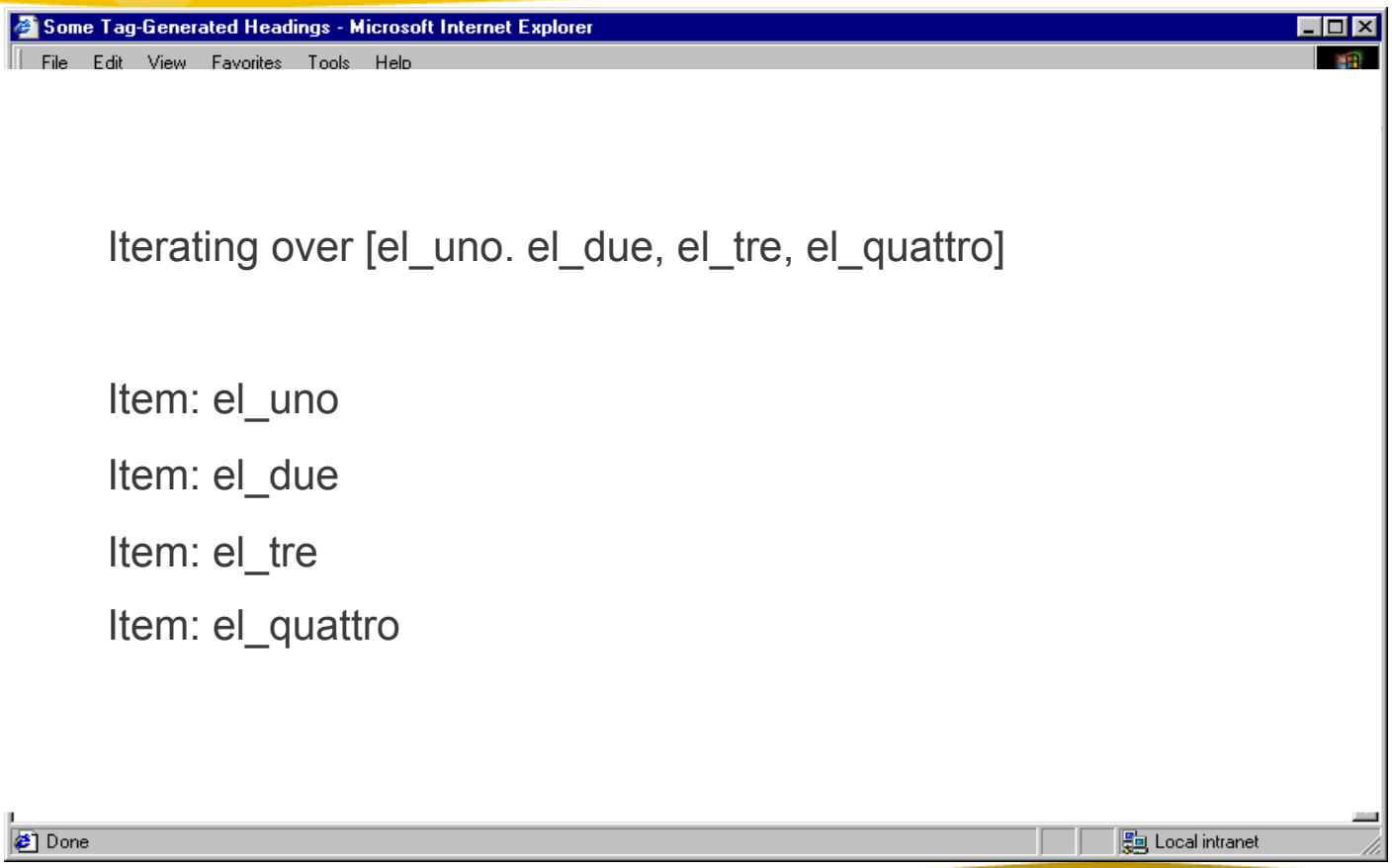
Nota sulla specifica JSP 1.2

- Secondo la nuova specifica è possibile scrivere tag iterativi anche estendendo la classe TagSupport (metodo doAfterBody) ma non viene creato mai l'oggetto BodyContent (si risparmia quando non è strettamente necessario usarlo)
- TagSupport implementa l'interfaccia IterationTag (quest'ultima estende l'interfaccia Tag che invece non consente iterazioni del tag), rendendo inutile l'uso di BodyTag se doStartTag restituisce EVAL_BODY_INCLUDE

Funzionamento del contenuto del corpo

- Il contenuto del corpo è rappresentato dalla classe **`BodyContent`** (scrittore con buffer)
- La classe **`BodyContent`** estende **`JspWriter`** (il tipo della variabile implicita **`out`**)
- I contenitori di servlet mantengono uno stack di oggetti **`BodyContent`** per fare in modo che un tag annidato non sovrascriva il contenuto del corpo di uno dei tag antenati
- Ciascun oggetto **`BodyContent`** conserva un riferimento allo scrittore con buffer del livello inferiore nello stack.
- Tale scrittore è noto come *previous out*, o *scrittore allegato*, ed è disponibile attraverso
  - **`BodyContent.getEnclosingWriter`** oppure
  - **`BodyTagSupport.getPreviousOut`**

Esempio: Iterazione (1/5)



Esempio: Iterazione /test.jsp (2/5)

```
<html><head><title>Un iteratore</title></head>
<%@ taglib uri="/WEB-INF/tlds/iterator.tld" prefix="it" %>
<body>

<% java.util.Vector vector = new java.util.Vector();
  vector.addElement("el_uno"); vector.addElement("el_due");
  vector.addElement("el_tre"); vector.addElement("el_quattro");
%>

Iterating over <%= vector %> ...
<p>
<it:iterate collection="<%= vector %>">
        <jsp:useBean id="item" scope="page" class="java.lang.String"/>
        Item: <%= item %><br>
</it:iterate>
</p>
</body>
</html>
```

Esempio: Iterazione (3/5)

/WEB-INF/classes/tags/IteratorTag.java

```
package tags;

import java.util.Collection;
import java.util.Iterator;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.BodyTagSupport;

public class IteratorTag extends BodyTagSupport{
        private Collection collection;
        private Iterator iterator;

        public void setCollection (Collection collection) {
          this.collection=collection;
        }

        public int doStartTag() throws JspException {
          return collection.size() > 0? EVAL_BODY_BUFFERED : SKIP_BODY;
        }
// . . . segue
```

Esempio: Iterazione (4/5)

IteratorTag.java – segue

```
public void doInitBody() throws JspException {
  iterator=collection.iterator();
  pageContext.setAttribute("item", iterator.next());
}

public int doAfterBody() throws JspException {
   if (iterator.hasNext()) {
   pageContext.setAttribute("item", iterator.next());
   return EVAL_BODY_AGAIN;
   }
   else {
     try {
       getBodyContent().writeOut(getPreviousOut());
     }
     catch (java.io.IOException e) {
       throw new JspException (e.getMessage());
     }
     return SKIP_BODY;
   }
  }
}
```

Esempio: Iterazione (5/5)
/WEB-INF/tlds/iterator.tld

```
<taglib>
 <tlibversion>1.0</tlibversion>
 <jspversion>1.1</jspversion>
  <tag>
                <name> iterate </name>
                <tagclass> tags.IteratorTag </tagclass>
                <bodycontent> JSP </bodycontent>
                <attribute>
                        <name> collection </name>
                        <required> true </required>
                        <rtexprvalue> true </rtexprvalue>
                </attribute>
                <info>
                Scrive iterativamente gli elementi di una collezione                    </info>
        </tag>
</taglib>
```

Protezione delle risorse - Autenticazione

- Un utente tenta di accedere ad una risorsa protetta (es. una pagina jsp).
- Se l'utente è già stato autenticato la risorsa viene resa disponibile, altrimenti viene chiesto all'utente di digitare nome utente e password.
- Se nome e password non possono essere autenticati, viene visualizzato un errore e l'utente ha la possibilità di scrivere nuovamente i dati, altrimenti la risorsa viene resa disponibile.

Autenticazione dichiarativa vs. programmata

- L'approccio dichiarativo all'autenticazione è basato interamente sul servlet container che gestisce i nomi degli utenti, le password e i ruoli
- L'approccio programmato implica invece la gestione diretta della sicurezza da parte di servlet e pagine jsp (come nell'esempio del tag EnforceLogin)

Autenticazione dichiarativa

- Gli aspetti relativi alla sicurezza sono interamente gestiti dal serlvet container.
- Per prevenire accessi non autorizzati
  - Si utilizza il descrittore della web application (*web.xml*) per dichiarare che certe risorse sono riservate a utenti che rivestono certi ruoli.
  - Si definisce un metodo di autenticazione per identificare gli utenti e rispettivi ruoli.
  - Quando viene richiesta una risorsa protetta, il server richiede all'utente username e password, li confronta con un set predefinito e, automaticamente, tiene traccia degli utenti già autenticati. Questo processo è completamente trasparente alle servlet e alle pagine JSP.
- Per preservare la sicurezza dei dati sulla rete
  - Si usa il descrittore della web application per specificare che a certe risorse si può accedere solo attraverso una connessione https. Se gli utenti provano ad usare una connessione HTTP vengono forzati ad usare HTTPS con meccanismi di redirezione.

Autenticazione programmata

- Le risorse protette (servlet e pagine JSP) sono responsabili della gestione della propria sicurezza.
  - Portabilità del codice. Non ci sono elementi della web application che dipendono dal particolare server utilizzato. Non sono necessarie ulteriori specifiche nel descrittore.
- Per prevenire accessi non autorizzati
  - Ciascuna servlet o pagina JSP deve autenticare l'utente o verificare che sia già stato autenticato.
- Per preservare la sicurezza dei dati sulla rete
  - Ogni servlet o pagina JSP deve controllare il protocollo usato
  - Se gli utenti provano ad usare una connessione HTTP la servlet o la pagina JSP devono redirigere le richieste sul protocollo HTTPS.

Tipi di autenticazione dichiarativa

1. Autenticazione BASIC
2. Autenticazione basata su form
3. Autenticazione DIGEST

- La scelta del meccanismo viene specificata nel file web.xml in corrispondenza dei tag
  - **<login-config>** e **<auth-method>**

Principali e ruoli

- L'utente è un *principal*
- I *principal* sono entità denominate, che in genere rappresentano singoli individui o società.
- I principal possono ricoprire uno o più *ruoli (roles)*
  - Es: un cliente può essere nello stesso tempo un dipendente
- Le restrizioni di sicurezza della specifica delle servlet, associano i ruoli con le risorse protette

Restrizioni di sicurezza: /WEB-INF/web.xml

```
<web-app>
    <display-name>Web App x test meccanismi di sicurezza </display-name>
    <description> Test dei meccanismi di sicurezza </description>
    <login-config> … vedremo tra poco …  </login-config>
    <security-constraint>
    <web-resource-collection> <!-- risorse protette -->
        <web-resource-name>Una pagina protetta </web-resource-name>
        <url-pattern>/jsps/pagina-protetta.jsp</url-pattern>
    </web-resource-collection>
    <auth-constraint><!-- ruoli associati alle risorse -->
        <role-name>tomcat</role-name>
        <role-name>role1</role-name>
    </auth-constraint>
    </security-constraint>
</web-app>
```

Elementi di <security-constraint>

Elemento	Tipo*	Descrizione
`web-resource-collection`	+	Un sottoinsieme delle risorse di un'applicazione web al quale vengono applicate le restrizioni di sicurezza
`auth-constraint`	?	Le restrizioni di autorizzazione collocate su una o più raccolte di risorse web (conterrà i tag che identificano i ruoli)
`user-data-constraint`	?	Una specifica sul meccanismo di trasporto dei dati inviati tra un client e un servlet container

* Tipo: += uno o più

?= uno, facoltativo

Elementi <web-resource-collection>

Elemento	Tipo*	Descrizione
`web-resource-name`	`1`	Il nome di una risorsa web
`description`	`?`	La descrizione di una risorsa web
`url-pattern`	`*`	Uno schema url associato ad una risorsa web
`http-method`	`?`	Un metodo http associato ad una risorsa web

* Tipo: 1= uno, obbligatorio

?= zero o più

*=uno o più

Elementi <auth-constraint>

Elemento	Tipo*	Descrizione
description	**?**	Una descrizione della restrizione di un'autorizzazione
role-name	*****	Il ruolo al quale si applica la restrizione

* Tipo: 1= uno, obbligatorio

?= zero o più

*=uno o più

Risorse – Ruoli - Principali

- Le restrizioni specificate nel file web.xml associano i ruoli alle risorse.
- L'associazione dei ruoli ai principali è compito del servlet container
  - (file `<catalina-home>/conf/tomcat-users.xml`).

File di configurazione
<CATALINA-HOME>/conf/tomcat-users.xml

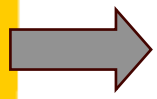
```
<tomcat-users>
 . . .
        <role rolename="cliente"/>
        <user name="rossi" password="tomcat" roles="cliente", "altro" />
 . . .
</tomcat-users>
```

Metodi di `HttpServletRequest`

- `Principal getUserPrincipal()`
  - Restituisce un riferimento a `java.security.Principal`
- `Boolean isUserInRole(String)`
  - Stabilisce se un utente ricopre un ruolo specificato da una stringa
- `String getRemoteUser()`
  - Restituisce il nome utente utilizzato per il login

Metodi di `HttpServletRequest` (cont.)

- Le API della servlet non forniscono i metodi "setter" corrispondenti ai metodi "getter" getUserPrincipal() e getRemoteUser()



Le applicazioni non possono impostare principali e ruoli; questi possono essere impostati solo dai contenitori di servlet

Metodi di `HttpServletRequest`(cont.)

- String `getAuthType`()
  - Restituisce il tipo di autenticazione BASIC, FORM, DIGEST.
- Boolean `isSecure`()
  - Restituisce true se la connessione è http o https
- String `getScheme`()
  - Fornisce lo schema che rappresenta il meccanismo di trasporto: http, https, ftp o altro.

/WEB-INF/web.xml (aut. dichiarativa)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
. . .
<web-app>
. . .
    <security-constraint>
    . . .
    </security-constraint>

    <login-config>
          <auth-method> BASIC </auth-method>
          <realm-name>Esempio di autenticazione Basic </realm-name>
    </login-config>
</web-app>
```

Il metodo **`HttpServletRequest.getAuthType()`** fornisce il metodo di autenticazione utilizzato

Autenticazione BASIC

- Quando un client tenta di accedere ad una risorsa protetta, il servlet container invia automaticamente una finestra di richiesta di nome utente e password
- L'invio della finestra di richiesta avviene in modo trasparente alle pagine JSP e alle servlet
- Questo metodo manca completamente di sicurezza. Le password vengono trasmesse con la codifica base64

Codifica base64

- La password viene trasformata in una sequenza di caratteri appartenenti ad un sottogruppo del set di caratteri ASCII. In questo modo, ogni carattere codificato può essere rappresentato con sei bit.
- Ogni gruppo di 24 bit viene diviso in quattro gruppi di sei bit, ad ognuno dei quali si associa il corrispondente carattere ASCII appartenente al sottogruppo specificato.
- Le password sono estremamente vulnerabili

Esempio sull’autenticazione BASIC

Web Application di test (aut. BASIC)
/JSPS/pagina-protetta.jsp

```
<html><head><title>Una pagina protetta </title></head>
<body>
<%@ include file="show-security.jsp" %>

<%  if (request.isUserInRole("tomcat")){ %>
        Appartieni al ruolo <i>tomcat</i> <br>
<%      } else { %>
        Non appartieni al ruolo <i>tomcat</i><br>
<%      }  %>

<% if (request.isUserInRole("role1")) { %>
       Appartieni al ruolo <i>role1</i> <br>
<%     } else { %>
       Non appartieni al ruolo <i>role1</i><br>
<%     }  %>

</body></html>
```

Web Application di test (aut. BASIC)
/JSPS/show-security.jsp

```
<font size="4" color="blue">
   Informazioni sulla sicurezza
</font> <br>
User principal: <%= request.getUserPrincipal().getName() %>. <br>
Request authenticated with: <%= request.getAuthType() %>. <br>

<% if (request.isSecure()) { %>
      This connection is secure. <br>
<% } else { %>
      This connection is NOT secure. <br>
<% } %>

Server Address: <%= request.getServerName() %> <br>
Remote Host: <%= request.getRemoteHost() %> <br>
Remote Addr; <%= request.getRemoteAddr() %>
```

Web Application di test (aut. BASIC)
/WEB-INF/web.xml

```
<web-app>
    <display-name>Web App x test meccanismi di sicurezza </display-name>
    <description>
    Test dei meccanismi di sicurezza
    </description>

    <security-constraint>
    <!-- risorse protette -->
    <web-resource-collection>
        <web-resource-name>Una pagina protetta </web-resource-name>
        <url-pattern>/jsps/pagina-protetta.jsp</url-pattern>
    </web-resource-collection>
. . .
```

Web Application di test (aut. BASIC) /WEB-INF/web.xml (cont.)

```
. . .
    <auth-constraint>
        <!-- ruoli associati alle risorse indicate sopra -->
        <role-name>tomcat</role-name>
        <role-name>role1</role-name>
    </auth-constraint>
    </security-constraint>

    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>Basic Authentication Example</realm-name>
    </login-config>
</web-app>
```

<CATALINA-HOME>/conf/tomcat-users.xml

```
<tomcat-users>
    <user name="tomcat" password="tomcat" roles="tomcat" />
    <user name="role1" password="tomcat" roles="role1" />
    <user name="both" password="tomcat" roles="tomcat","role1" />
</tomcat-users>
```

Il file tomcat-users.xml appartiene al servlet container e realizza l'associazione tra il principal e il ruolo da lui ricoperto

Autenticazione basata su form

- Permette di controllare l'aspetto e il comportamento della pagina di login
- Il metodo basato su form funziona come il metodo basic ma viene visualizzata la pagina di login anzichè una finestra di dialogo
- Viene inoltre specificata una pagina di errore per il caso di mancata autenticazione
- La password viene trasmessa con la codifica base64

Autenticazione basata su form (cont.)

- Implementare una pagina di login
- Implementare una pagina di errore, che verrà visualizzata in caso di mancata autenticazione
- Nel descrittore della web application specificare che si adotta l'autenticazione basata su form e il nome delle pagine di login e di errore

Web Application di test (aut. FORM)
/JSPS/login.jsp

```
<html><head><title>Pagina di login</title></head>
<body>
<font size="4" color="blue">
    Per favore inserisci i tuoi dati:<br></font>

<form action="j_security_check" method="POST">
<table>
<tr><td>Name:</td>
    <td><input type="text" name="j_username"></td></tr>
<tr><td>Password:</td>
    <td><input type="password" name="j_password" size="8"></td>
</tr>
</table>
<br>
  <input type="submit" value="login">
</form>
</body>
</html>
```

Web Application di test (aut. FORM)
/JSPS/login.jsp

- **`j_username`**, **`j_password`**, **`j_security_check`** sono nomi dettati dalla servlet specification
- **`j_username`**: il nome del campo username
- **`j_password`**: il nome del campo password
- **`j_security_check`**: l'azione del modulo di login

- La pagina di login così definita viene automaticamente richiamata dal servlet container quando si tenta di accedere alla risorsa protetta specificata nel descrittore.

Web Application di test (aut. FORM)
/JSPS/error.jsp

```
<html><head><title>ERRORE !!!</title></head>

<body>

<font size="6" color="red">
    I dati inseriti non sono validi!<br>
</font>

Clicca <a href=' <%=response.encodeURL("login.jsp") %>'>qui</a> per
riprovare!

</body>
</html>
```

Web Application di test (aut. FORM)
/WEB-INF/web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>test di autenticazione mediante form</display-name>
    <description>Test </description>
    <security-constraint>
    <!-- risorse protette -->
    <web-resource-collection>
        <web-resource-name>Una pagina protetta </web-resource-name>
        <url-pattern>/jsps/pagina-protetta.jsp</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <!-- ruoli associati alle risorse indicate sopra -->
        <role-name>tomcat</role-name>
        <role-name>role1</role-name>
    </auth-constraint>
    </security-constraint> . . .
```

Web Application di test (aut. FORM)
/WEB-INF/web.xml (cont.)

```
. . .
<login-config>
        <auth-method>FORM</auth-method>
        <form-login-config>
            <form-login-page>/jsps/login.jsp</form-login-page>
            <form-error-page>/jsps/error.jsp</form-error-page>
        </form-login-config>
    </login-config>
</web-app>
```

/WEB-INF/web.xml

```
. . .
<web-app>
. . .
    <security-constraint>
    <web-resource-collection> . . .
        <url-pattern>/jsps/risorsaA.jsp</url-pattern>
    </web-resource-collection>
    <auth-constraint> . . .
        <role-name>ruolo A</role-name>
    </auth-constraint>
    </security-constraint>
    <security-constraint>
    <web-resource-collection> . . .
        <url-pattern>/jsps/risorsaB.jsp</url-pattern>
    </web-resource-collection>
    <auth-constraint> . . .
        <role-name>ruolo B</role-name>
    </auth-constraint>
 /security-constraint> <
</web-app>
```

Meccanismi di autenticazione dichiarativa

- BASIC (equiv. FORM) e DIGEST
- Sono definiti nel documento RFC 2617, http://ftp.isi.edu/in-notes/rfc2617.txt
- BASIC e FORM: la password viene trasmessa in chiaro
- DIGEST: la password non viene trasmessa, ma viene trasmesso un valore hash.

Autenticazione DIGEST

- Quando un client tenta di accedere ad una risorsa protetta, il servlet container invia automaticamente una finestra di richiesta di nome utente e password
- L'invio della finestra di richiesta avviene in modo trasparente alle pagine JSP e alle servlet
- La password viene codificata con metodi di hashing

Codifica con il metodo DIGEST

- Lo schema Digest è basato su un meccanismo di domanda-risposta.
- L'entità responsabile dell'autenticazione (il server) propone all'utente un valore *nonce* (ogni volta diverso e valido solo per la richiesta corrente)
- Affinchè l'utente possa essere autenticato, deve rispondere con un valore di checksum calcolato in base a
  - Username e Password
  - Il valore nonce proposto
  - Il metodo HTTP
  - URI

/WEB-INF/web.xml (aut. dichiarativa)

Autenticazione dichiarativa vs. programmata

- L'approccio dichiarativo all'autenticazione è basato interamente sul servlet container che gestisce i nomi degli utente, le password e i ruoli
- L'approccio programmato implica invece la gestione diretta della sicurezza da parte di servlet e pagine jsp

Problemi con meccanismi dichiarativi

- L'accesso è tutto-o-niente
  - Gli utenti possono accedere alle risorse protette oppure essere bloccati, senza alternative ulteriori
  - Non è possibile proporre agli utenti contenuti diversi a seconda dei ruoli ricoperti.
- L'accesso, basato esclusivamente su password è completamente controllato dal server.

Problemi con meccanismi dichiarativi

- La soluzione non è sempre portabile
- I server devono supportare *qualche* metodo per la definizione di utenti, password e ruoli, ma server diversi lo fanno in modo diverso.
- E' necessario modificare il file web.xml

Autenticazione programmata

- Le risorse protette (servlets e pagine JSP) sono responsabili della gestione della propria sicurezza.
  - Protabilità del codice. Non ci sono elementi della web application che dipendono dal particolare server utilizzato. Non sono necessarie ulteriori specifiche nel descrittore.
- Per prevenire accessi non autorizzati
  - Ciascuna servlet o pagina JSP deve autenticare l'utente o verificare che sia già stato autenticato.
- Per preservare la sicurezza dei dati sulla rete
  - Ogni servlet o pagina JSP deve controllare il protocollo usato
  - Se gli utenti provano ad usare una connessione HTTP la servlet o la pagina JSP devono redirigere le richieste sul protocollo HTTPS.

Approccio combinato: dichiarativo + programmatico

- Si fa affidamento sul server per la gestione di nomi, password e ruoli attraverso metodi dichiarativi
  - Autenticazione BASIC, basata su form oppure DIGEST
- Si gestisce l'accesso alle risorse in modo esplicito dalle servlet o dalle pagine JSP
  - Esempio: si può cambiare il contenuto della pagina secondo l'identità del richiedente (impossibile con metodi puramente dichiarativi).

Approccio combinato: dichiarativo + programmatico (cont.)

- Si utilizzano i seguenti metodi

  `HttpServletRequest`

  - **`boolean isUserInRole(String)`**
  - **`String getRemoteUser()`**
  - **`Principal getUserPrincipal()`**

  per definire il contenuto della risorsa protetta sulla base dell'identità del richiedente.

Approccio combinato: /WEB-INF/web.xml (cont.)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
. . .
<web-app>
    <display-name>Web App x test meccanismi di sicurezza </display-name>
    <description> Test dei meccanismi di sicurezza </description>

    <security-constraint>
    <web-resource-collection> <!-- risorse protette -->
        <web-resource-name>Una pagina protetta </web-resource-name>
        <url-pattern>/jsps/pagina-protetta.jsp</url-pattern>
    </web-resource-collection>
    <auth-constraint><!-- ruoli associati alle risorse -->
        <role-name>pagante</role-name>
        <role-name>nonpagante</role-name>
    </auth-constraint>
    </security-constraint>
</web-app>
```

Approccio combinato: /WEB-INF/pagina-protetta.jsp (cont.)

- Notare che il file web.xml consente l'accesso alla risorsa protetta sia agli utenti paganti che a quelli non paganti

```
<html><head><title>Una pagina protetta </title></head>
<body>
<%  if (request.isUserInRole("pagante")){ %>
        Informazioni riservate agli utenti abbonati
<%      } else if (request.isUserInRole("nonpagante")) {  %>
        Informazioni per chi non si è ancora abbonato
<%      }  %>
</body></html>
```

Approccio combinato: /WEB-INF/pagina-protetta.jsp (cont.)

- Oppure se gli utenti non paganti cercano di accedere ad una risorsa riservata agli abbonati si può decidere di deviare la richiesta verso una pagina di registrazione (o di login):

```
<html><head><title>Una pagina protetta </title></head>
<body>
<%  if (request.isUserInRole("pagante")){ %>
        Informazioni riservate agli utenti abbonati
<%      } else if (request.isUserInRole("nonpagante")) {
        response.sendRedirect("PaginaDiLogin.jsp");
        }  %>
</body></html>
```

Utilizzo di basi di dati da servlet/JSP

Installazione di MySQL

- Installazione di MySQL
  - scaricare il file **`mysql-4.0.18-win.zip`** dalla pagina del corso
  - eseguire setup
  - configurare le variabili di ambiente aggiungendo il percorso **`...\mysql\bin`** al **`PATH`**
- Avviare il server
  - eseguire **`...\mysql\bin\mysqld`**

Il server MySQL (cont.)

- Per installare il server come servizio di Windows NT/XP, eseguire

  `...\mysql\bin\mysql-nt -install`

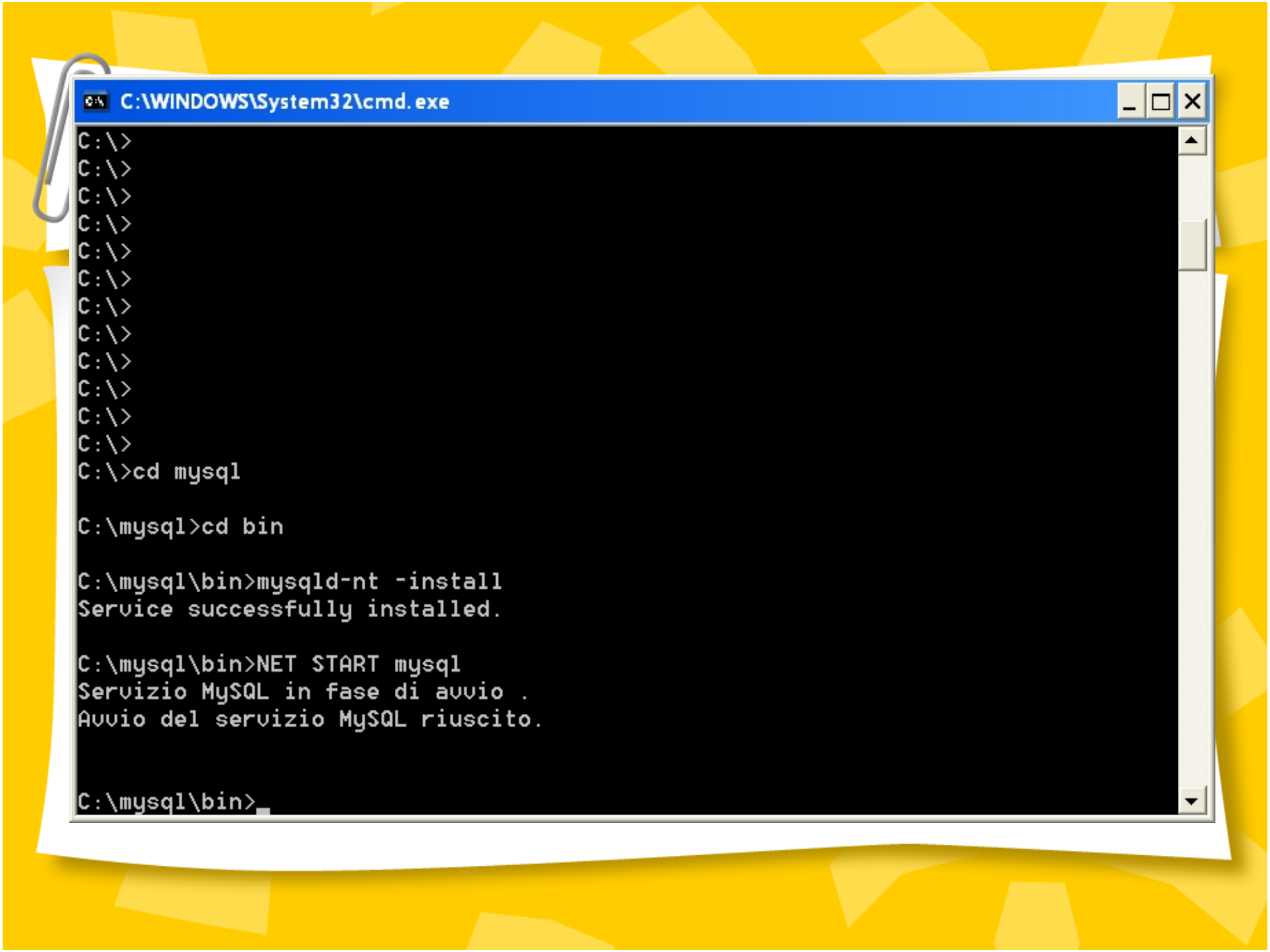
- Per avviare il servizio, eseguire il comando **`NET START mysql`**

- Per arrestare il servizio, eseguire il comando

  `NET STOP mysql`

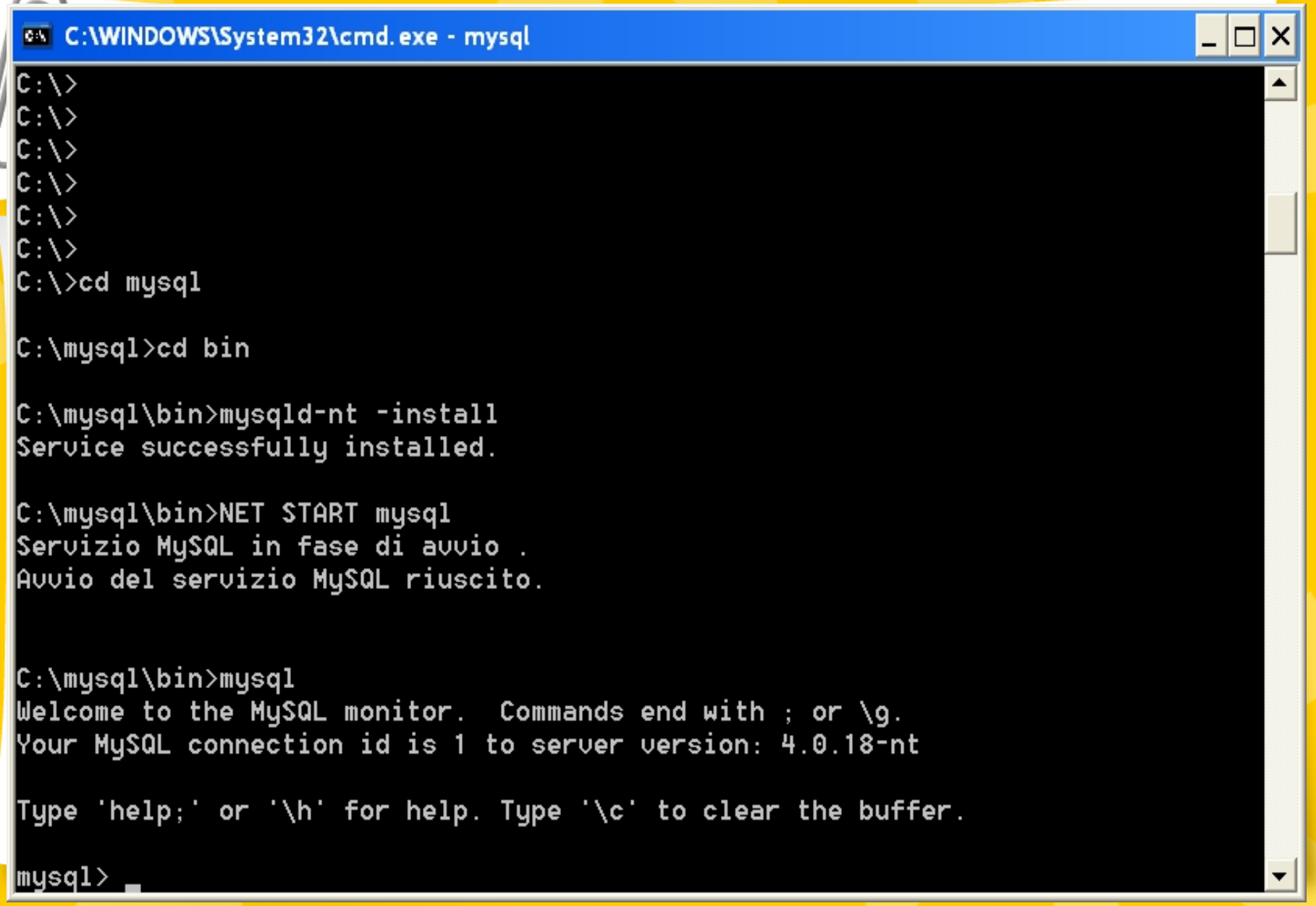
- Per rimuovere il server eseguire

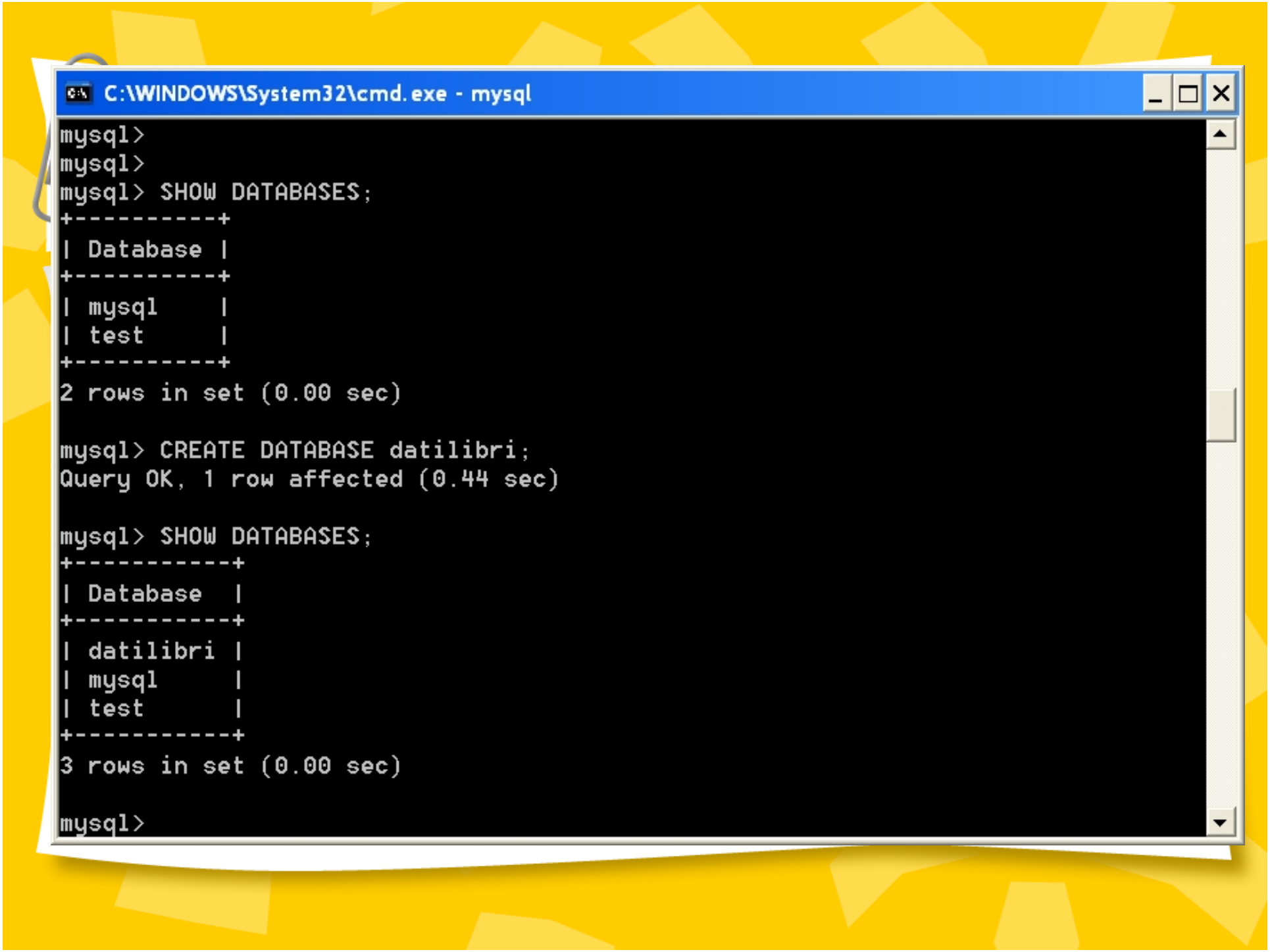
  `...\mysql\bin\mysql-nt -remove`


C:\WINDOWS\System32\cmd.exe
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>cd mysql
C:\mysql>cd bin
C:\mysql\bin>mysqld-nt -install
Service successfully installed.
C:\mysql\bin>NET START mysql
Servizio MySQL in fase di avvio .
Avvio del servizio MySQL riuscito.
C:\mysql\bin>

Il client MySQL

- Per eseguire il client MySQL digitare
  - `mysql`
- Per avere l'elenco dei database ai quali ha accesso il server MySQL, eseguire
  - `SHOW DATABASES;`
- Per creare un nuovo database, eseguire
  - `CREATE DATABASE nome_nuovo_database;`


C:\WINDOWS\System32\cmd.exe - mysql
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>cd mysql
C:\mysql>cd bin
C:\mysql\bin>mysqld-nt -install
Service successfully installed.
C:\mysql\bin>NET START mysql
Servizio MySQL in fase di avvio .
Avvio del servizio MySQL riuscito.
C:\mysql\bin>mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.0.18-nt
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>


C:\WINDOWS\System32\cmd.exe - mysql
mysql>
mysql>
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql |
| test |
+----------+
2 rows in set (0.00 sec)
mysql> CREATE DATABASE datilibri;
Query OK, 1 row affected (0.44 sec)
mysql> SHOW DATABASES;
+-----------+
| Database |
+-----------+
| datilibri |
| mysql |
| test |
+-----------+
3 rows in set (0.00 sec)
mysql>

Utilizzare una delle basi di dati disponibili

- Appena si avvia il server non è possibile modificare alcun database perchè il programma non sa con quale database deve lavorare
  - Istruzione **USE**

    ```
    USE nome_nuovo_database;
    ```
- Una volta selezionato il database da utilizzare è possibile eseguire tutte le operazioni di accesso e modifica ai dati
  - L'istruzione **SHOW TABLES** mostra le tabelle della base di dati

Definizione di tabelle

- Si può aggiungere una tabella ad una base di dati con l'istruzione

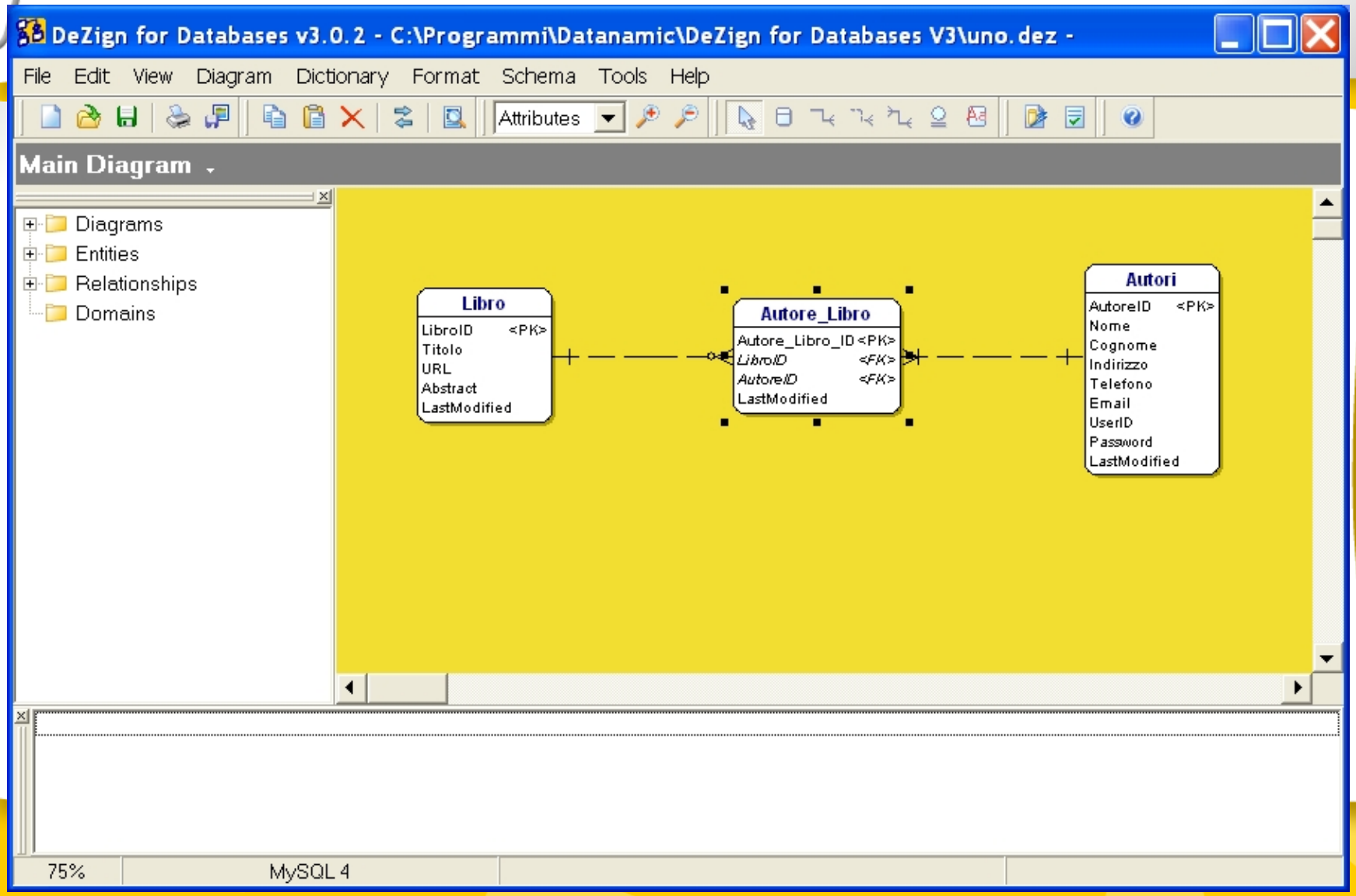
```
CREATE TABLE nome_tabella (
campo1 tipo1,
campo2 tipo2,
..., ...
);
```

Informazioni sulle tabelle

- Per ottenere una descrizione dettagliata dei campi previsti in una tabella utilizzare l'istruzione

```
DESCRIBE nome_tabella;
```

Progetto di una piccola base di dati



Creazione della base di dati da script

- Scrivere tutti i comandi SQL per la creazione delle tabelle e per l'inserimento dei dati dal prompt del client è un procedimento lungo e soggetto a numerosi errori -> si ricorre spesso a script.

- Per poter utilizzare un file di script file_script.sql, si esegue il comando

```
source file_script.sql
```

File di script creaDB.sql

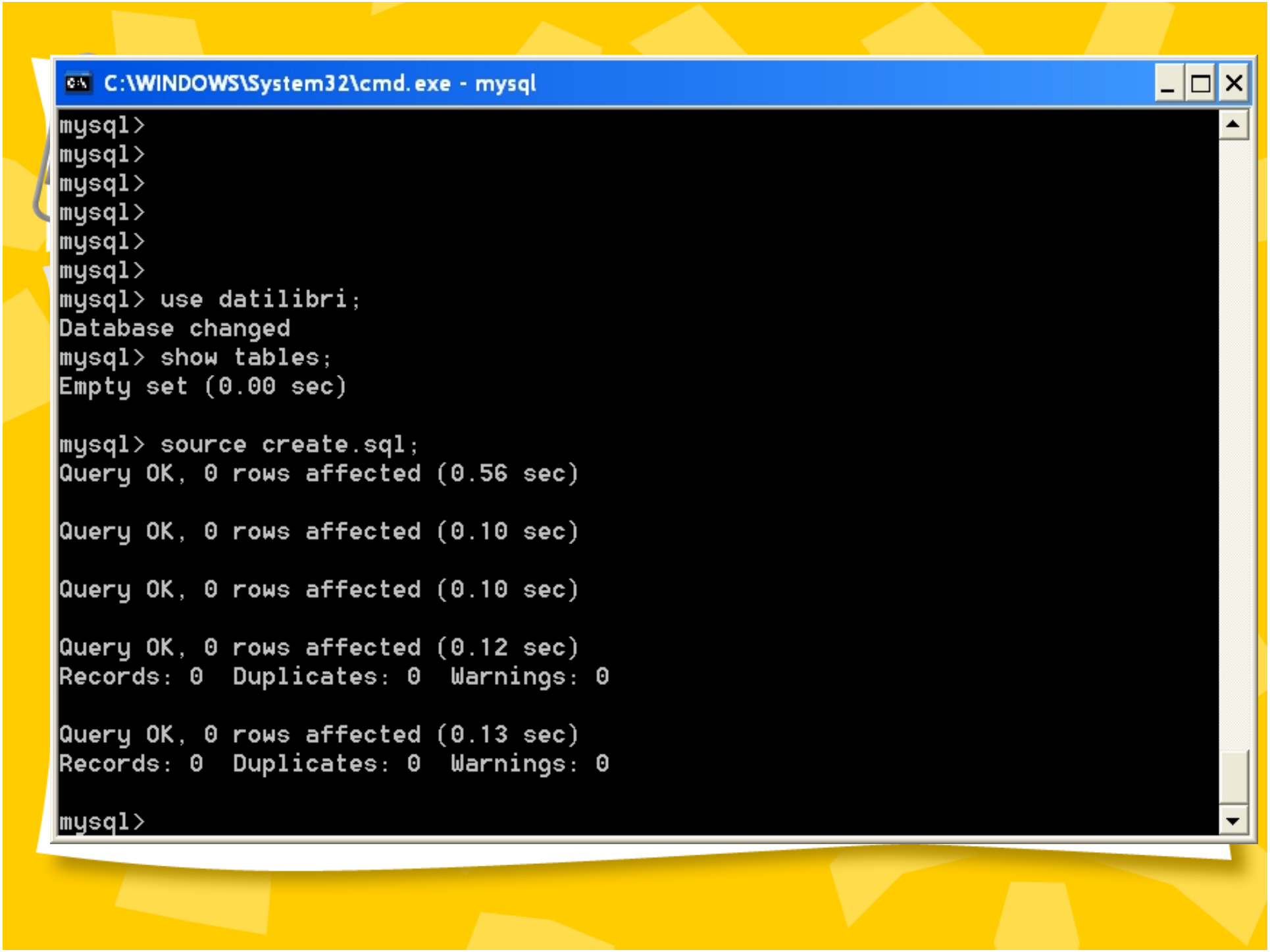
```
CREATE DATABASE esempio;
USE esempio;

CREATE TABLE Libro (
    LibroID INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    Titolo VARCHAR(40) NOT NULL,
    URL VARCHAR(40),
    Abstract TEXT,
    LastModified TIMESTAMP,
    CONSTRAINT LibroID PRIMARY KEY (LibroID),
    UNIQUE KEY IDX_Libro_1(LibroID)
);

. . .
```

File di script creaDB.sql (cont.)

```
. . .
CREATE TABLE Autori (
    AutoreID INTEGER NOT NULL AUTO_INCREMENT,
    Nome VARCHAR(40) NOT NULL,
    Cognome VARCHAR(40) NOT NULL,
    Indirizzo VARCHAR(40),
    Telefono VARCHAR(40),
    Email VARCHAR(40),
    UserID VARCHAR(40),
    Password VARCHAR(40),
    LastModified TIMESTAMP,
    PRIMARY KEY (AutoreID),
    UNIQUE KEY IDX_Autori_1(AutoreID)
);

. . .
```

File di script creaDB.sql (cont.)

```
. . .
CREATE TABLE Autore_Libro (
    Autore_Libro_ID INTEGER NOT NULL AUTO_INCREMENT,
    LibroID INTEGER NOT NULL,
    AutoreID INTEGER NOT NULL,
    LastModified TIMESTAMP,
    PRIMARY KEY (Autore_Libro_ID),
    KEY IDX_Autore_Libro_1(LibroID),
    KEY IDX_Autore_Libro_2(AutoreID)
);

ALTER TABLE Autore_Libro
    ADD FOREIGN KEY (LibroID) REFERENCES Libro (LibroID);

ALTER TABLE Autore_Libro
    ADD FOREIGN KEY (AutoreID) REFERENCES Autori (AutoreID);
```


C:\WINDOWS\System32\cmd.exe - mysql
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql> use datilibri;
Database changed
mysql> show tables;
Empty set (0.00 sec)
mysql> source create.sql;
Query OK, 0 rows affected (0.56 sec)
Query OK, 0 rows affected (0.10 sec)
Query OK, 0 rows affected (0.10 sec)
Query OK, 0 rows affected (0.12 sec)
Records: 0 Duplicates: 0 Warnings: 0
Query OK, 0 rows affected (0.13 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql>

Precisazione sui privilegi di accesso

CREATE DATABASE esempio;

grant all privileges on esempio.* to 'admin'@'localhost' identified by 'admin';
flush privileges;

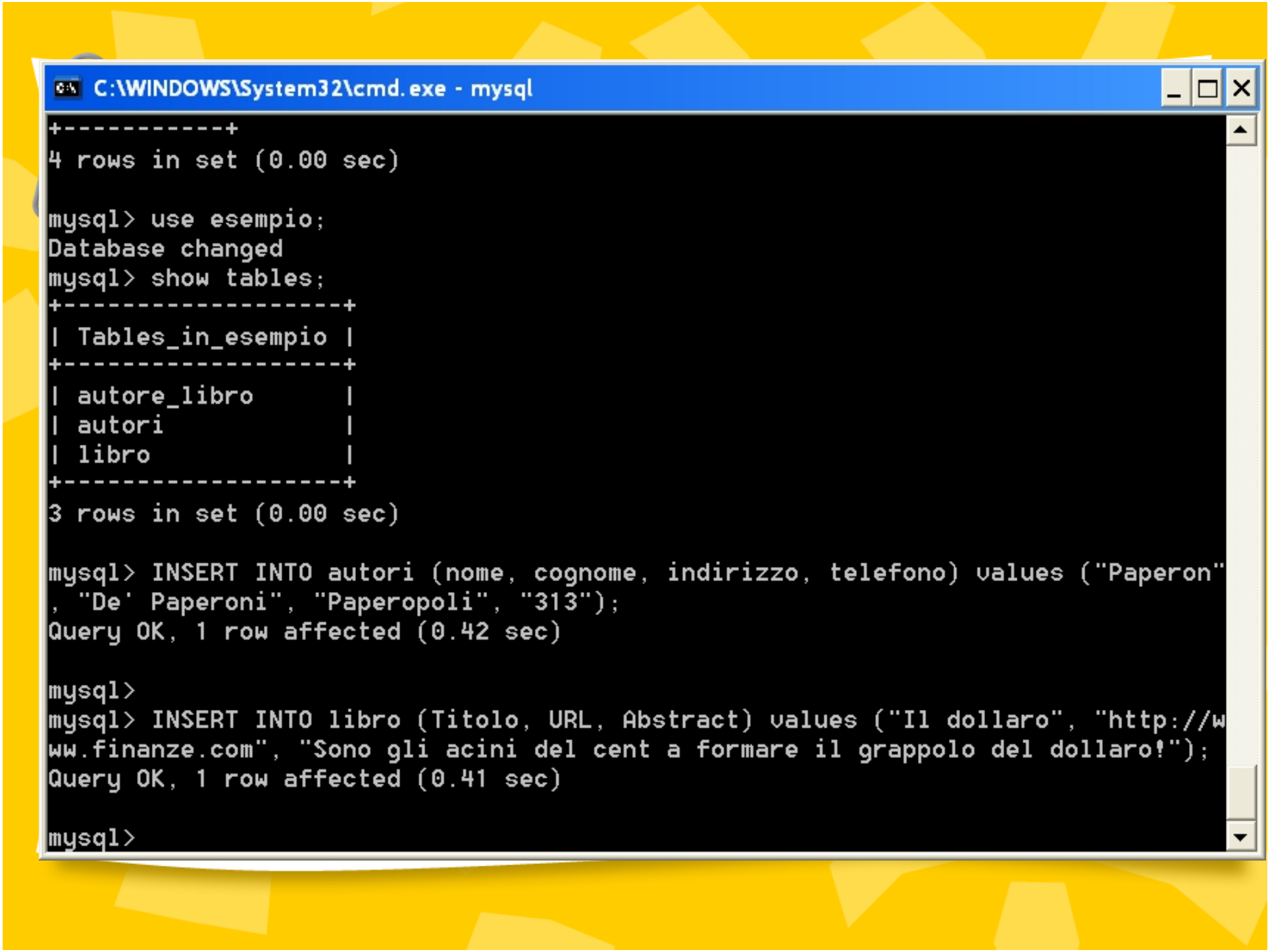
USE esempio;

Per accedere al db si usano user=‘admin’ e password=‘admin’

Inserimento di un record in una tabella

- Per inserire un nuovo record in una tabella si usa il comando insert

```
INSERT INTO nome_tabella
(campo_1, campo_2, ..., campo_n)
VALUES (“valore1”, “valore2”, ... “valoren”);
```


C:\WINDOWS\System32\cmd.exe - mysql
+-----------+
4 rows in set (0.00 sec)
mysql> use esempio;
Database changed
mysql> show tables;
+-------------------+
| Tables_in_esempio |
+-------------------+
| autore_libro |
| autori |
| libro |
+-------------------+
3 rows in set (0.00 sec)
mysql> INSERT INTO autori (nome, cognome, indirizzo, telefono) values ("Paperon"
, "De' Paperoni", "Paperopoli", "313");
Query OK, 1 row affected (0.42 sec)
mysql>
mysql> INSERT INTO libro (Titolo, URL, Abstract) values ("Il dollaro", "http://w
ww.finanze.com", "Sono gli acini del cent a formare il grappolo del dollaro!");
Query OK, 1 row affected (0.41 sec)
mysql>

Modifica di record di una tabella

- Modificare dei dati in una tabella
  - **UPDATE** tableName
    SET fieldName1 = value1, … , fieldNameN = valueN
    WHERE criteria
    - **Es: UPDATE** autori
      SET cognome = 'Jones'
      WHERE cognome = 'Smith' **AND** nome = 'Sue'

Rimozione di record da una tabella

- Per rimuovere dei dati da una tabella si usa l'istruzione **DELETE**
- `DELETE FROM tableName WHERE criteria`
- `DELETE FROM autori WHERE cognome = 'Jones' AND nome = 'Sue'`

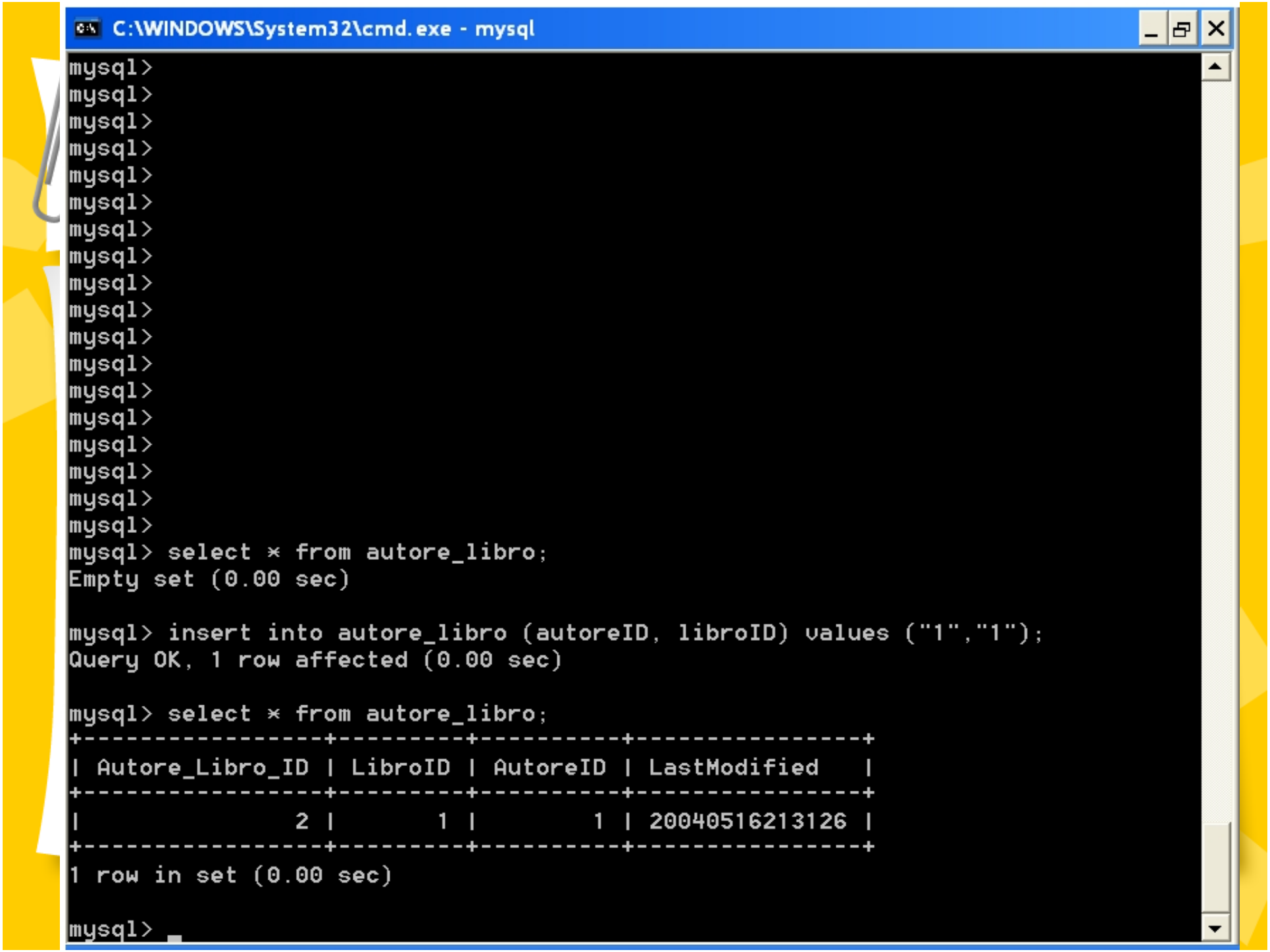
Consultazione di una tabella

- Istruzione

```
SELECT ... FROM ... WHERE ...
```

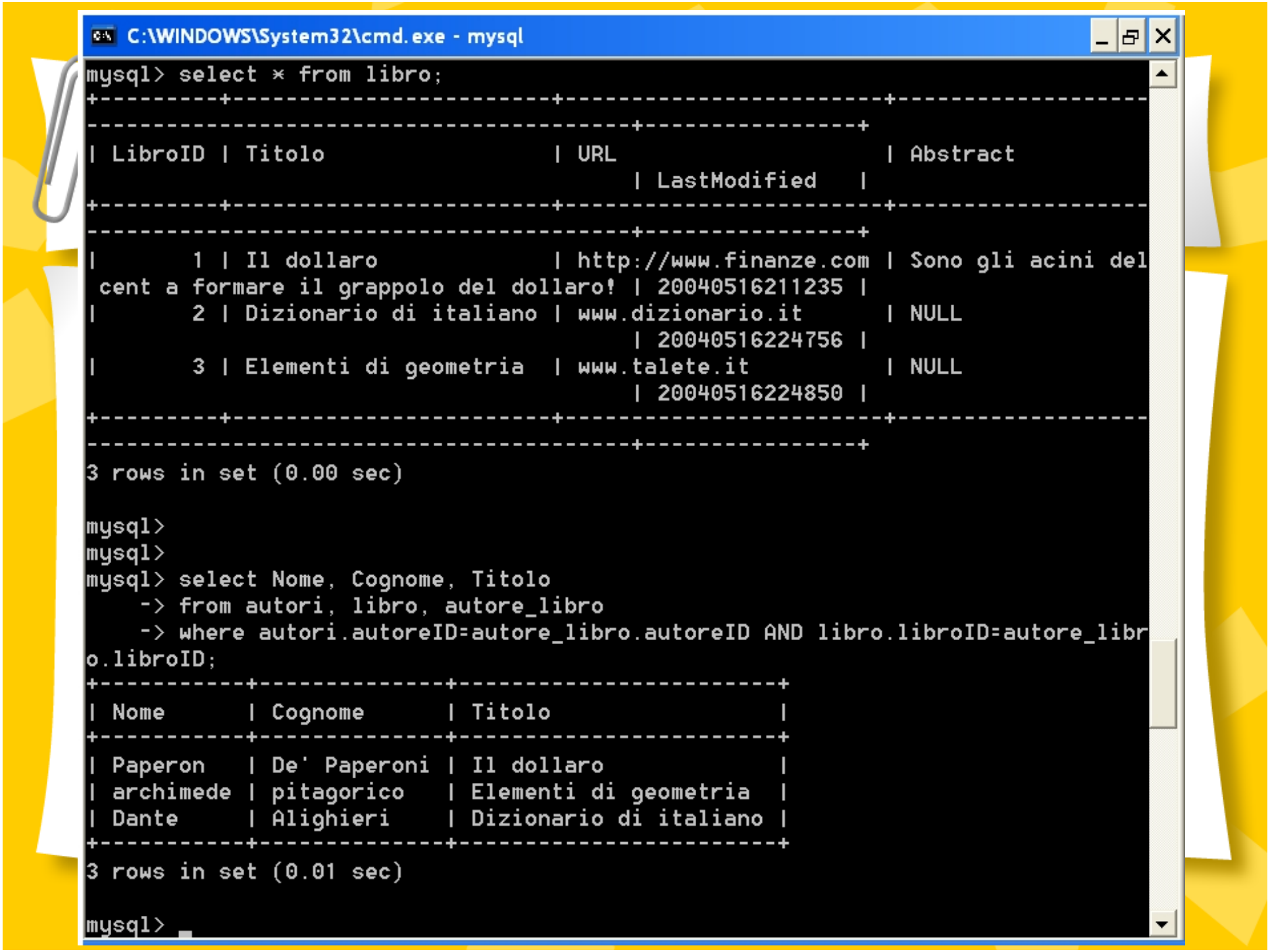
Es:

```
SELECT campo1, campo2
 FROM nome_tabella
 WHERE campo3=valore_x;
```


C:\WINDOWS\System32\cmd.exe - mysql
mysql> select * from autore_libro;
Empty set (0.00 sec)
mysql> insert into autore_libro (autoreID, libroID) values ("1","1");
Query OK, 1 row affected (0.00 sec)
mysql> select * from autore_libro;
| Autore_Libro_ID | LibroID | AutoreID | LastModified |
| 2 | 1 | 1 | 20040516213126 |
1 row in set (0.00 sec)
mysql>

Fusione dei dati di più tabelle (join)

- Per combinare i dati di più tabelle
  -
```
SELECT fieldName1, fieldName2, …
    FROM table1, table2
    WHERE table1.fieldName = table2.fieldName
```
  -
```
SELECT Nome, Cognome, LibroID
    FROM autori, autore_libro
    WHERE autori.autoreID =
          autore_libro.autoreID
    ORDER BY Cognome, Nome
```


C:\WINDOWS\System32\cmd.exe - mysql
mysql> select * from libro;
+---------+-------------------------+-------------------------+------------------
--+----------------+
| LibroID | Titolo | URL | Abstract
                                       | LastModified |
+---------+-------------------------+-------------------------+------------------
--+----------------+
| 1 | Il dollaro | http://www.finanze.com | Sono gli acini del
 cent a formare il grappolo del dollaro! | 20040516211235 |
| 2 | Dizionario di italiano | www.dizionario.it | NULL
                                       | 20040516224756 |
| 3 | Elementi di geometria | www.talete.it | NULL
                                       | 20040516224850 |
+---------+-------------------------+-------------------------+------------------
--+----------------+
3 rows in set (0.00 sec)
mysql>
mysql>
mysql> select Nome, Cognome, Titolo
    -> from autori, libro, autore_libro
    -> where autori.autoreID=autore_libro.autoreID AND libro.libroID=autore_libr
o.libroID;
+-----------+-------------+-------------------------+
| Nome | Cognome | Titolo |
+-----------+-------------+-------------------------+
Paperon	De' Paperoni	Il dollaro
archimede	pitagorico	Elementi di geometria
Dante	Alighieri	Dizionario di italiano
+-----------+-------------+-------------------------+
3 rows in set (0.01 sec)
mysql> _

Ordinamento dei risultati di una query

- Clausola **ORDER BY** opzionale
  - `SELECT fieldName1, fieldName2, … FROM tableName ORDER BY field ASC`
  - `SELECT fieldName1, fieldName2, … FROM tableName ORDER BY field DESC`
- L'ordinamento può anche essere richiesto su più campi
  - `ORDER BY field1 sortingOrder, field2 sortingOrder, …`

Utilizzare MySQL da servlet/JSP

- Creare una connessione ad un database
- Interrogare/modificare il database
- Mostrare il risultato di un'interrogazione

…. iniziamo con degli esempi pratici che utilizzeremo per comprendere i dettagli teorici…

Utilizzare MySQL da servlet/JSP

- Si utilizzano delle API (driver) che forniscono i metodi per l'apertura di una connessione con il database, il recupero e l'aggiornamento dei dati
- Uno dei driver maggiormente consigliati per prestazioni e affidabilità è il

  `MySQL Connector/J` che trovate alla pagina del corso
  - Il file **`mysql-connector-java-3.0.11-stable-bin.jar`** va incluso in una delle directory del **`CLASSPATH`** (esempio **`/context-root/WEB-INF/lib`**, oppure in **`<CATALINA_HOME>/common/lib`**)

Esercizio: servlet che visualizza i dati del database "esempio" (1/4)

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
    public class MySQLServlet extends javax.servlet.http.HttpServlet {
        public void doGet(HttpServletRequest request,
                          HttpServletResponse response)
                throws IOException, ServletException {
            response.setContentType("text/html");
            PrintWriter out=response.getWriter();
            out.println("<html><head><title> Test di MySQL
                                        </title></head><body>");
            out.println("<H1> Questa pagina è prodotta da <br>"+
                        "una servlet che esegue una query al DB "</H1><br>");
            out.println("<table border=\"1\" cellpadding=\"5\""
                + " cellspacing=\"0\" width=\"400\">");

. . .
```

Esercizio: servlet che visualizza i dati del database "esempio" (2/4)

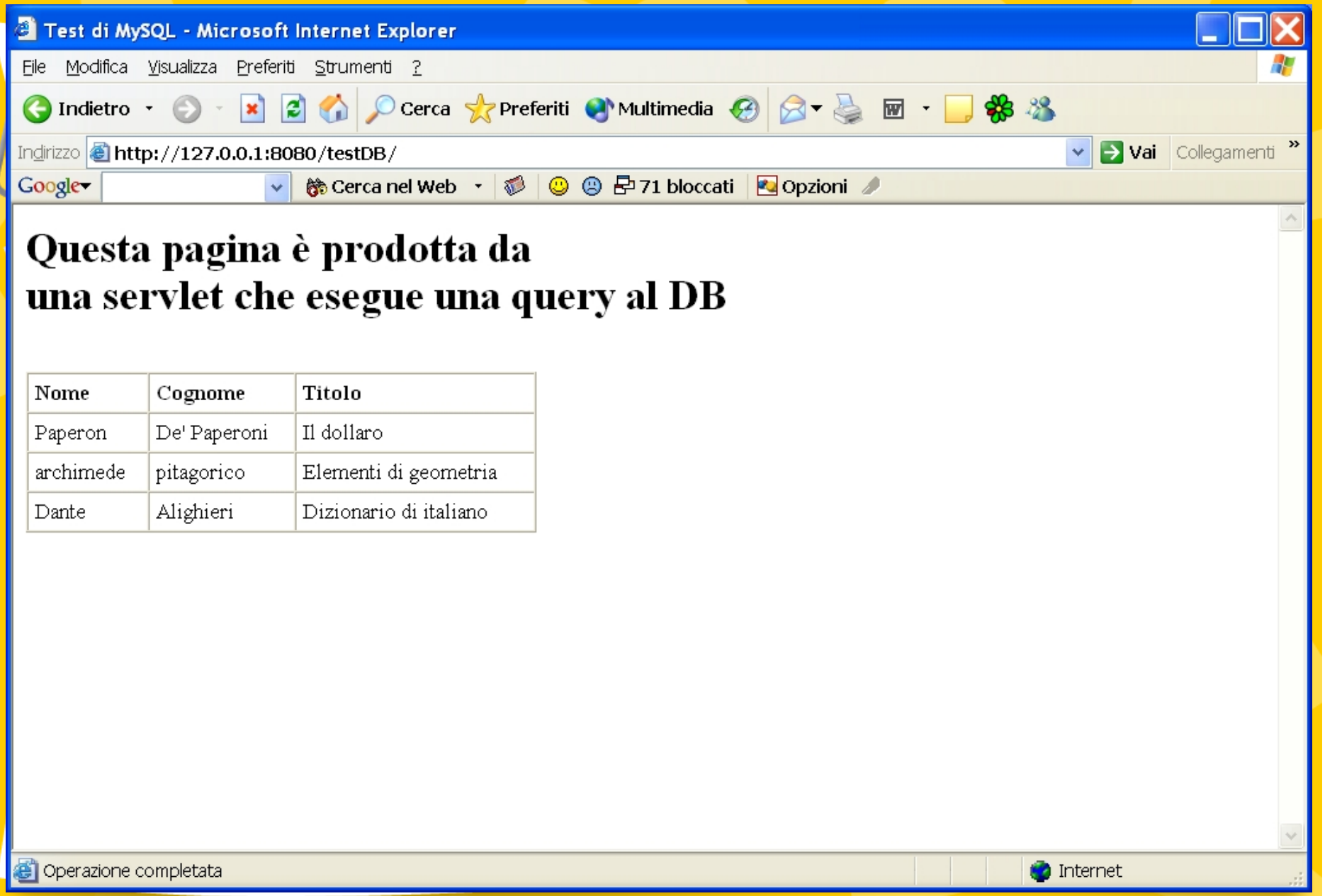
```
String connectionURL="jdbc:mysql://localhost:3306/esempio";
Connection connection = null;
Statement statement=null;
ResultSet resultSet=null;
try {
        //caricamento dinamico della classe e
        //registrazione del driver presso il driver manager
    Class.forName( "com.mysql.jdbc.Driver");
        // connect to database
    connection = DriverManager.getConnection(connectionURL);
        // create Statement to query database
    statement = connection.createStatement();

        // query database
    String query_autori_libri;
    query_autori_libri="SELECT Nome, Cognome, Titolo ";
    query_autori_libri+="from autori, libro, autore_libro ";
    query_autori_libri+="where autori.autoreID=autore_libro.autoreID ";
    query_autori_libri+="AND libro.libroID=autore_libro.libroID;";
    resultSet =
       statement.executeQuery(query_autori_libri);
```

Esercizio: servlet che visualizza i dati del database "esempio" (3/4)

```
// process query results
        StringBuffer results = new StringBuffer();
        ResultSetMetaData metaData = resultSet.getMetaData();
        results.append("<tr>");
        for ( int i = 1; i <= 3; i++ ) {
           results.append( "<td><b>" + metaData.getColumnName( i )
           + "</b></td>" );
        }
        results.append( "</tr>" );

        while ( resultSet.next() ) {
            results.append("<tr>");
            for ( int i = 1; i <= 3; i++ ) {
                results.append("<td>"+ resultSet.getObject( i )
                + "</td>" );
               }
              results.append( "</tr>" );
            }
       out.println(results.toString());
       } //fine del try
```

Esercizio: servlet che visualizza i dati del database "esempio" (4/4)

```
        // detect problems interacting with the database
             catch ( SQLException e ) {
                 System.err.println("SQL Problem: "+e.getMessage());
                 System.err.println("SQL State: "+e.getSQLState());
                 System.err.println("Error: "+e.getErrorCode());
                 System.exit( 1 );
             }

             // detect problems loading database driver
             catch ( ClassNotFoundException e ) {
                System.err.println("Non trovo il driver"+ e.getMessage());
             }

             finally {  //eseguita sempre a meno che non venga chiamato exit()
                   try  {
                        if (connection!=null) connection.close();
                   }
                   catch (SQLException e) {
                        System.err.println(e.getMessage());
                   }
             }
             out.println("</table></body></html>");
        }
}
```

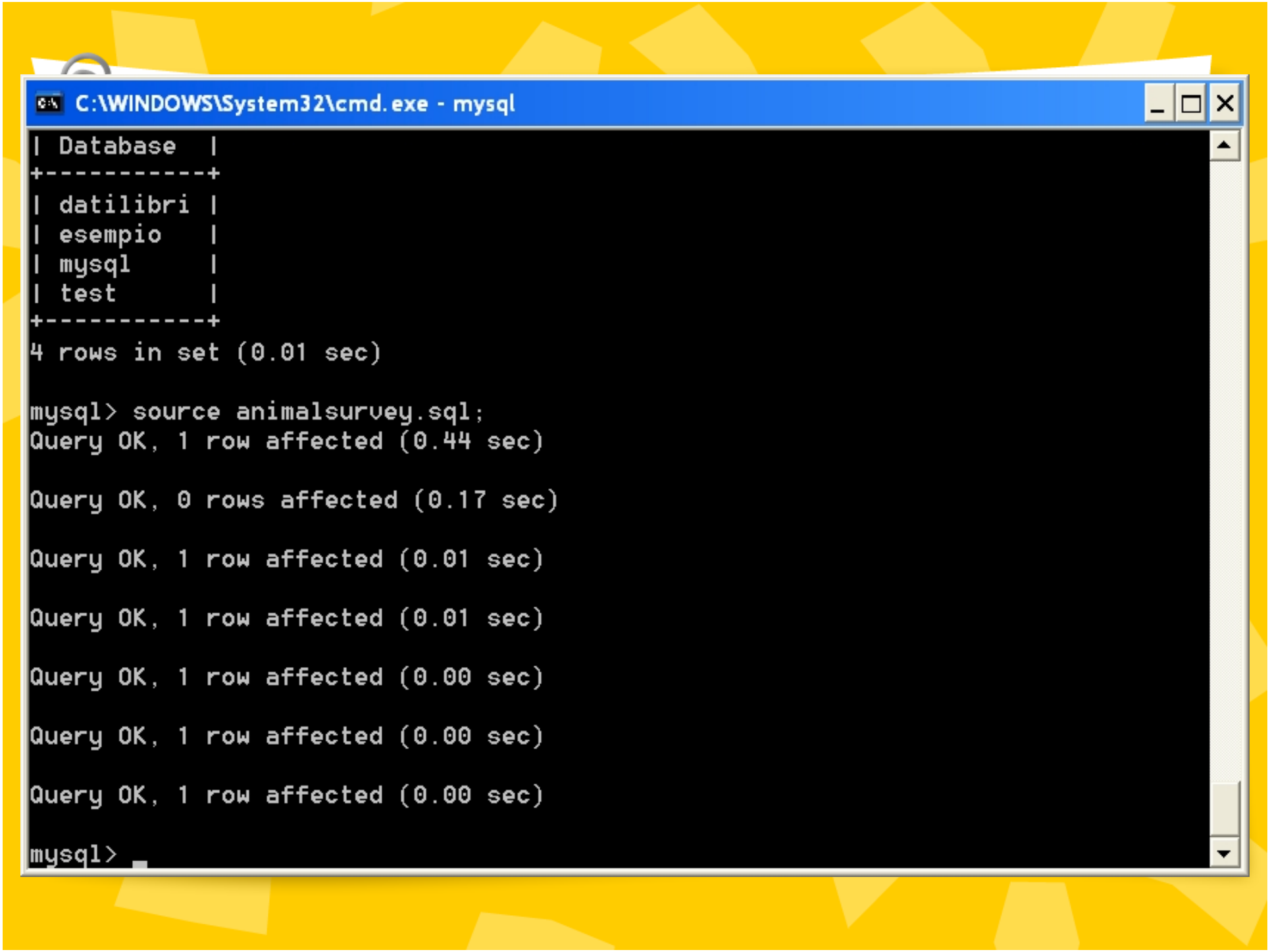

Test di MySQL - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
Indietro
Cerca
Preferiti
Multimedia
Indirizzo
http://127.0.0.1:8080/testDB/
Vai
Collegamenti
Google
Cerca nel Web
71 bloccati
Opzioni
Questa pagina è prodotta da
una servlet che esegue una query al DB
Nome
Cognome
Titolo
Paperon
De' Paperoni
Il dollaro
archimede
pitagorico
Elementi di geometria
Dante
Alighieri
Dizionario di italiano
Operazione completata
Internet

Esercizio: sondaggio online

- In una base di dati si vogliono rappresentare 4 categorie di animali. All'utente dell'applicazione viene sottoposto un form in cui selezionare il proprio animale preferito. Al termine della votazione viene visualizzata una pagina con le percentuali di voti riscosse da ciascun animale.

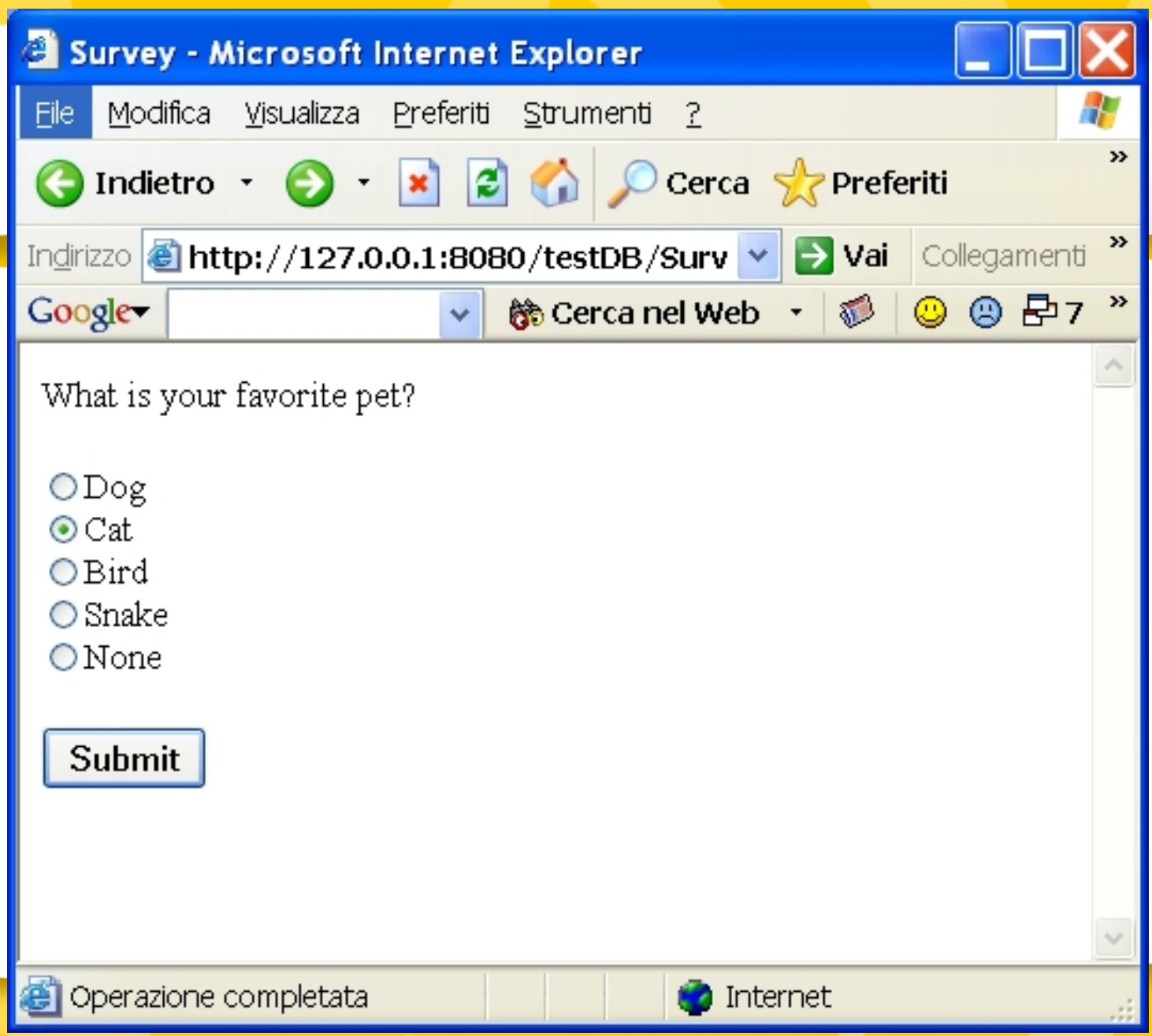
Creazione della base di dati: animalsurvey.sql

```
CREATE DATABASE animalsurvey;
USE animalsurvey;
create table surveyresults (
          id int NOT NULL ,
          surveyoption varchar (20) NOT NULL ,
          votes int NOT NULL ,
          constraint surveyresults_id primary key (id)
);

insert into surveyresults (id,surveyoption,votes) values (1, 'Dog', 0);
insert into surveyresults (id,surveyoption,votes) values (2, 'Cat', 0);
insert into surveyresults (id,surveyoption,votes) values (3, 'Bird', 0);
insert into surveyresults (id,surveyoption,votes) values (4, 'Snake', 0);
insert into surveyresults (id,surveyoption,votes) values (5, 'None', 0);
```


C:\WINDOWS\System32\cmd.exe - mysql
| Database |
+----------+
| datilibri |
| esempio |
| mysql |
| test |
+----------+
4 rows in set (0.01 sec)
mysql> source animalsurvey.sql;
Query OK, 1 row affected (0.44 sec)
Query OK, 0 rows affected (0.17 sec)
Query OK, 1 row affected (0.01 sec)
Query OK, 1 row affected (0.01 sec)
Query OK, 1 row affected (0.00 sec)
Query OK, 1 row affected (0.00 sec)
Query OK, 1 row affected (0.00 sec)
mysql>

Form di realizzazione del sondaggio: survey.html

```
<html><head><title>Survey</title></head>
<body>
<form method = "post" action = "/testDB/Survey">
   <p>What is your favorite pet?</p>
   <p>
      <input type = "radio" name = "animal"
         value = "1" />Dog<br />
      <input type = "radio" name = "animal"
         value = "2" />Cat<br />
      <input type = "radio" name = "animal"
         value = "3" />Bird<br />
      <input type = "radio" name = "animal"
         value = "4" />Snake<br />
      <input type = "radio" name = "animal"
         value = "5" checked = "checked" />None
   </p>
   <p><input type = "submit" value = "Submit" /></p>
</form>
</body> </html>
```


Survey - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
Indietro
Cerca
Preferiti
Indirizzo
http://127.0.0.1:8080/testDB/Surv
Vai
Collegamenti
Google
Cerca nel Web
What is your favorite pet?
Dog
Cat
Bird
Snake
None
Submit
Operazione completata
Internet

Realizzazione di una query attraverso l'interfaccia **`PreparedStatement`**

- Interfaccia **`PreparedStatement`**s
  - Più flessibile
  - Più efficiente
- Definire una query con un'oggetto che implementa l'interfaccia **`PreparedStatement`**
  - `PreparedStatement stringa_da_configurare = connection.prepareStatement(`

```
"SELECT field1, field2, field3" +
"FROM tableA, tableB" +
"WHERE tableA.field_X = tableB.field_Y "  +
"AND fieldJ = ? AND fieldK = ?" );
```

Realizzazione di una query attraverso l'interfaccia `PreparedStatement`(cont.)

- Configurare i parametri in **`PreparedStatement`**
  - `stringa_da_configurare.setString( 1, "Rossi" );`
  - `stringa_da_configurare.setString( 2, "Paolo" );`
- **`PreparedStatement`** con i parametri configurati
  -
```
SELECT field1, field2, field3
  FROM tableA, tableB
  WHERE tableA.field_X = tableB.field_Y AND
        fieldJ = 'Rossi' AND fieldK = 'Paolo'
```

Servlet che gestisce la base di dati del sondaggio: SurveyServlet.java (1/5)

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

public class SurveyServlet extends HttpServlet {
   private Connection connection;
   private PreparedStatement updateVotes, totalVotes, results;

   // set up database connection and prepare SQL statements
   public void init( ServletConfig config )
      throws ServletException
   {
      // attempt database connection and create PreparedStatements
      try {
         Class.forName("com.mysql.jdbc.Driver");
         connection = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/animalsurvey" );
         // PreparedStatement to add one to vote total for a
         // specific animal
         updateVotes =
            connection.prepareStatement(
               "UPDATE surveyresults SET votes = votes + 1 " +
               "WHERE id = ?");
```

Servlet che gestisce la base di dati del sondaggio: SurveyServlet.java (2/5)

```
// PreparedStatement to sum the votes
         totalVotes =
            connection.prepareStatement(
               "SELECT sum( votes ) FROM surveyresults"
            );

         // PreparedStatement to obtain surveyoption table's data
         results =
            connection.prepareStatement(
               "SELECT surveyoption, votes, id " +
               "FROM surveyresults ORDER BY id"
            );
      }

      // for any exception throw an UnavailableException to
      // indicate that the servlet is not currently available
      catch ( Exception exception ) {
         exception.printStackTrace();
         throw new UnavailableException( exception.getMessage() );
      }

   }  // end of init method
```

Servlet che gestisce la base di dati del sondaggio: SurveyServlet.java (3/5)

```
// process survey response
   protected void doPost( HttpServletRequest request,
      HttpServletResponse response )
         throws ServletException, IOException
   {
      // set up response to client
      response.setContentType( "text/html" );
      PrintWriter out = response.getWriter();
      DecimalFormat twoDigits = new DecimalFormat( "0.00" );

      // start XHTML document

      out.println(
         "<html><head>" );

      // read current survey response
      int value =
         Integer.parseInt( request.getParameter( "animal" ) );
```

Servlet che gestisce la base di dati del sondaggio: SurveyServlet.java (4/5)

```
// attempt to process a vote and display current results
      try {
         // update total for current survey response
         updateVotes.setInt( 1, value );
         updateVotes.executeUpdate();
         // get total of all survey responses
         ResultSet totalRS = totalVotes.executeQuery();
         totalRS.next();
         int total = totalRS.getInt( 1 );
         // get results
         ResultSet resultsRS = results.executeQuery();
         out.println( "<title>Thank you!</title></head><body> " );
         out.println( "<p>Thank you for participating." );
         out.println( "<br />Results:</p>" );
         // process results
         int votes;
         while ( resultsRS.next() ) {
            out.print( resultsRS.getString( 1 )+": " );
            votes = resultsRS.getInt( 2 );
            out.print( twoDigits.format(( double ) votes / total * 100 ) );
            out.print( "%  responses: "+ votes );
         }
         resultsRS.close();
```

Configura il primo parametro di **PreparedStatement updateVotes** dove avevamo "…where id=?"

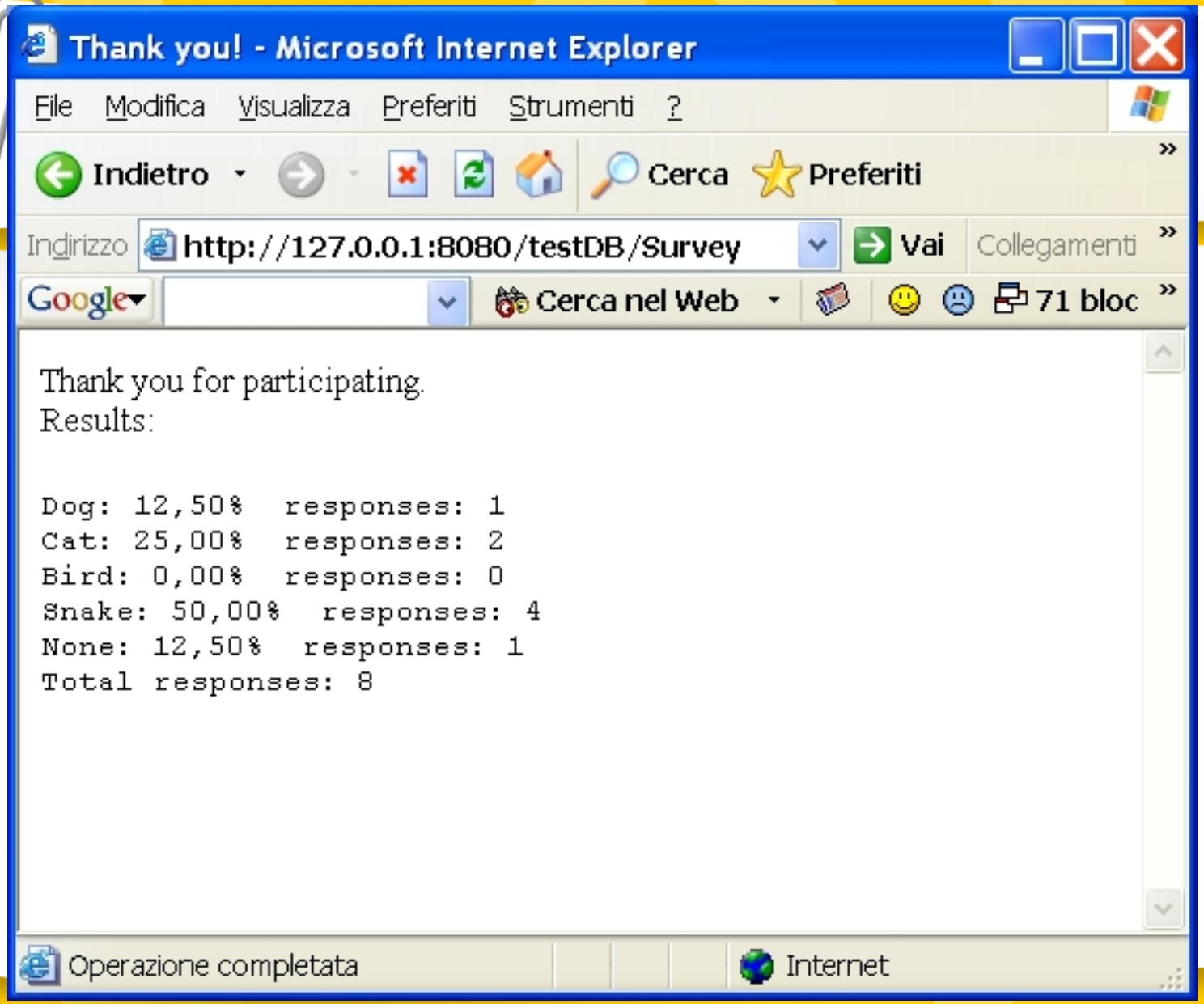
Esegue la query **totalVotes** per calcolare il numero di voti ricevuti.

Esegue la **PreparedStatement results** e elabora **ResultSet** per creare il risultato del sondaggio.

Servlet che gestisce la base di dati del sondaggio: SurveyServlet.java (5/5)

```
   out.print( "Total responses: " + total + "</body></html>");
    out.close();
      }
      // if database exception occurs, return error page
      catch ( SQLException sqlException ) {
         sqlException.printStackTrace();
         out.println( "<title>Error</title></head>"" );
         out.println( "<body><p>Database error occurred. " );
         out.println( "Try again later.</p></body></html>" );
         out.close();
      }
   }  // end of doPost method
   // close SQL statements and database when servlet terminates
   public void destroy()
   {
      // attempt to close statements and database connection
      try {
         updateVotes.close();
         totalVotes.close();
         results.close();
         connection.close();
      }
      // handle database exceptions by returning error to client
      catch( SQLException sqlException ) {sqlException.printStackTrace();}
   }  // end of destroy method
}
```

Il metodo destroy elimina ciascuna **PreparedStatement** e chiude tutte le connessioni.


Thank you! - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
Indietro
Cerca
Preferiti
Indirizzo
http://127.0.0.1:8080/testDB/Survey
Vai
Collegamenti
Google
Cerca nel Web
71 bloc
Thank you for participating.
Results:
Dog: 12,50% responses: 1
Cat: 25,00% responses: 2
Bird: 0,00% responses: 0
Snake: 50,00% responses: 4
None: 12,50% responses: 1
Total responses: 8
Operazione completata
Internet

JDBC (Java Database Connectivity)

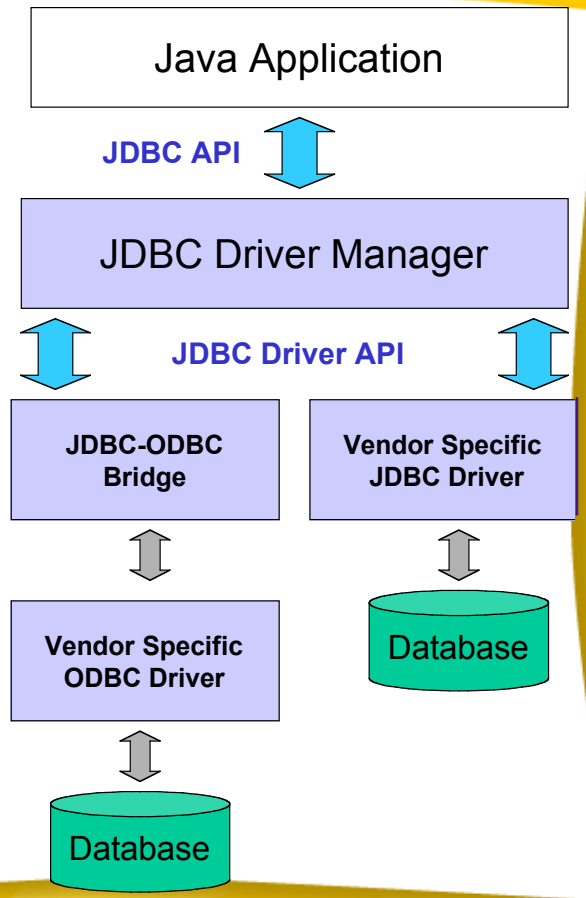
- JDBC è costituito da una libreria per l'accesso a basi di dati di tipo relazionale
  - Le API JDBC standardizzano
    - Le modalità di connessione con una base dati
    - Modalità di interrogazione della base dati
    - Modalità di creazione di query parametrizzate
    - Le strutture dati con cui il risultato di una query può essere trattato
      - Come determinare il numero di colonne di una tabella
      - Utilizzo di metadati, etc.
  - Le API JDBC non standarizzano la sintassi SQL
  - Le classi che costituiscono il supporto JDBC si trovano nel package java.sql

Risorse online

- Sito della Sun su JDBC
  - http://java.sun.com/products/jdbc/
- Tutorial su JDBC
  - http://java.sun.com/docs/books/tutorial/jdbc/
- Elenco dei driver JDBC disponibili
  - http://industry.java.sun.com/products/jdbc/drivers/
- Informazioni sulle API di java.sql
  - http://java.sun.com/j2se/1.4/docs/api/java/sql/package-summary.html

Driver JDBC

- JDBC consiste di:
  - API JDBC API, puramente basate su JAVA
  - Un manager di driver JDBC, che comunica con i driver specifici dell'applicativo in uso per realizzare il DB.



Driver JDBC

- L'applicazione inoltra all'API JDBC le chiamate per l'apertura di una connessione con il database, recupera e aggiorna i dati, esegue i comandi previsti e chiude la connessione.
- I driver del database si connettono ad un database specifico oppure ad un protocollo intermedio (come ODBC o altro middleware)

Driver JDBC

- I database riconoscono il linguaggio SQL in modi diversi.
- I database forniscono protocolli diversi per la connessione al loro motore.
- Compete al driver provvedere a tutti i problemi di conversione tra i comandi JDBC e il motore del database.

n.b.: Le API JDBC e il driver manager fanno parte del JDK, mentre i driver sono reperibili presso il fornitore della base di dati

Sette passi per connettersi ad una base di dati sfruttando il supporto JDBC

1. Caricare il driver
2. Definire l'URL per la connessione con la base dati
3. Instaurare la connessione
4. Creare un oggetto Statement che rappresenta la query da inoltrare
5. Eseguire la query
6. Elaborare il risultato
7. Chiudere la connessione

I sette passi nel dettaglio

1. Caricare il driver

```
try {
  Class.forName("com.mysql.jdbc.Driver");
//oppure
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
} catch { ClassNotFoundException cnfe) {
  System.out.println("Error loading driver: " cnfe);
}
```

= *caricamento dinamico della classe e registrazione del driver presso il driver manager*

Note sul caricamento del driver

- Il caricamento della classe del driver avviene in modo dinamico, a tempo di esecuzione.
- Il codice che inizializza la classe del driver provvede a registrare il driver presso il driver manager
- Il driver può essere caricato anche nel seguente modo:

```
DriverManager.registerDriver("classe.del.driver");
```

- Cambiando il driver è possibile usare un db completamente diverso senza modificare il codice

I sette passi nel dettaglio

2. Definire il percorso (URL) della connessione

```
String host = "dbhost.yourcompany.com";
String dbName = "someName";
int port = 1234;
String accessURL = "jdbc:mySQL:" + host +
                   ":" + port + ":" + dbName;
```

I sette passi nel dettaglio

3. Instaurare una connessione

```
String username = "Paperone";
String password = "numero1";
Connection connection =
    DriverManager.getConnection(accessURL,
                                username,
                                password);
```

Informazioni sulla base di dati

- Una volta instaurata la connessione è possibile richiedere informazioni generiche sulla base dati:

```
DatabaseMetaData dbMetaData =
                              connection.getMetaData();
 String productName =
              dbMetaData.getDatabaseProductName();
 String productVersion =
              dbMetaData.getDatabaseProductVersion();
System.out.println("Database: " +  productName);
System.out.println("Version: " +  productVersion);
```

I sette passi nel dettaglio

4. Creare un oggetto Statement che rappresenta la query da inoltrare

```
Statement statement =
    connection.createStatement();
```

I sette passi nel dettaglio

5. Eseguire una query

```
Statement statement = connection.createStatement();
String query =
    "SELECT col1, col2, col3 FROM sometable";
ResultSet resultSet =
    statement.executeQuery(query);
```

- Notare che executeQuery() restituisce un ResultSet e non altera la base di dati
- Per modificare la base dati, usare l'istruzione **`executeUpdate`**, fornendo comandi SQL come **`UPDATE`**, **`INSERT`**, o **`DELETE`**. Viene restituito un intero corrispondente al numero di righe modificate
- Usare **`setQueryTimeout`** per specificare il massimo intervallo di tempo che si è disposti ad aspettare prima per ottenere la risposta

I sette passi nel dettaglio

6. Elaborazione della risposta

```
while(resultSet.next()) {
   System.out.println(resultSet.getString(1) +
   " " + resultSet.getString(2) +
   " " + resultSet.getString(3));
}
```

- La prima colonna ha indice 1, non 0
- **`ResultSet`** fornisce numerosi metodi **`getXxx`** con argomento un indice di colonna *o il nome della colonna*

I sette passi nel dettaglio

7. Chiudere la connessione

```
connection.close();
```

   – Dal momento che le operazioni di apertura di una connessione sono molto costose, posporre questa operazione se sono necessarie ulteriori interazioni con la base dati

Utilizzo di MetaData

- Dati che riguardano l'applicazione
  - `connection.getMetaData().getDatabaseProductName()`
  - `connection.getMetaData().getDatabaseProductVersion()`
- Dati che riguardano le tabelle della base dati
  - `resultSet.getMetaData().getColumnCount()`
    - Se usate il risultato per elencare gli elementi di un record, ricordate che l'indice deve partire da 1 e non da 0
  - `resultSet.getMetaData().getColumnName(indice_colonna)`

Utilizzo di `Statement`

- Attraverso l'oggetto **`Statement`**, si possono inviare comandi SQL al database.
- Esistono alcuni tipi di statement:
  - **`Statement`**
    - Per eseguire un comando SQL semplice
  - **`PreparedStatement`**
    - Per eseguire un comando SQL precompilato e parametrizzato

Metodi di `Statement`

- **`executeQuery`**
  - Esegue la query e fornisce il risultato in una tabella (ResultSet)
  - La tabella del risultato può essere vuota ma mai **`null`**

```
ResultSet results =
    statement.executeQuery("SELECT a, b FROM table");
```

- **`executeUpdate`**
  - Usato per eseguire modifiche attraverso i comandi SQL **`INSERT`**, **`UPDATE`**, e **`DELETE`**
  - Il risultato è il numero di righe che sono state modificate
  - Supporta anche comandi Data Definition Language (DDL) come **`CREATE TABLE`**, **`DROP TABLE`** e **`ALTER TABLE`**

```
int rows =
        statement.executeUpdate("DELETE FROM EMPLOYEES" +
                                "WHERE STATUS=0");
```

Metodi di `Statement`

- **`getMaxRows/setMaxRows`**
  - Determina il massimo numero di righe che possono essere contenute in un **`ResultSet`**
  - A meno che non venga esplicitamente dichiarato, il numero di righe della risposta è illimitato (return value: 0)
- **`getQueryTimeout/setQueryTimeout`**
  - Specifica il periodo di tempo che il driver attenderà per ottenere una risposta prima di lanciare una eccezione **`SQLException`**

PreparedStatements

- Idea
  - Se dovete eseguire comandi simili ripetutamente, potete farlo efficientemente usando una statement precompilata e parametrizzata
  - Si crea una statement in una forma standard che viene compilata prima di essere utilizzata
  - Ogni volta che volete utilizzare una statement precompilata, dovete rimpiazzare i parametri marcati con “?” utilizzando i metodi **`setXxx`**

PreparedStatements

- **`PreparedStatement`** estende **`Statement.`**
  I metodi
  - executeQuery()
  - executeUpdate()

  vengono ereditati ma non prevedono nessun parametro.

PreparedStatement, Esempio

```
Connection connection =
  DriverManager.getConnection(url, user, password);
PreparedStatement statement =
  connection.prepareStatement("UPDATE employees "+
                              "SET salary = ? " +
                              "WHERE id = ?");
int[] newSalaries = getSalaries();
int[] employeeIDs = getIDs();
for(int i=0; i<employeeIDs.length; i++) {
  statement.setInt(1, newSalaries[i]);
  statement.setInt(2, employeeIDs[i]);
  statement.executeUpdate();
}
```

Metodi di PreparedStatement

- **`setXxx`**
  - Configura i parametri indicati con (?)
- **`clearParameters`**
  - Annulla tutti i parametri configurati

Realizzazione di una query attraverso l'interfaccia **PreparedStatement**

- Interfaccia **PreparedStatement**s
  - Più flessibile
  - Più efficiente
- Definire una query con un oggetto che implementa l'interfaccia **PreparedStatement**
  -
```
PreparedStatement authorBooks =
connection.prepareStatement(
        "SELECT cognome, nome, titolo " +
        "FROM autori, libro, autore_libro" +
        "WHERE autori.autoreID = autore_libro.autoreID "  +
        "AND libro.libroID = autore_libro.libroID AND " +
        "cognome = ? AND nome = ?" );
```

Realizzazione di una query attraverso l'interfaccia `PreparedStatement`(cont.)

- Configurare i parametri in **`PreparedStatement`**
  - `authorBooks.setString( 1, "Rossi" );`
  - `authorBooks.setString( 2, "Paolo" );`
- **`PreparedStatement`** con i parametri configurati
  -
```
SELECT cognome, nome, titolo
  FROM autori, libro, autore_libro
  WHERE autori.autoreID = autore_libro.autoreID AND
        libro.libroID = autore_libro.libroID AND
        cognome = 'Rossi' AND nome = 'Paolo'
```

Transazioni

- Per impostazione predefinita, dopo l'esecuzione di un comando SQL, il commit dei cambiamenti avviene immediatamente
- Impostando al valore off la configurazione di auto-commit si possono raggruppare due o più statement in una transazione
  - `connection.setAutoCommit(false)`
- La chiamata di `commit` modifica permanentemente i record registrando i cambiamenti conseguenti alla transazione
- La chiamata di `rollback` serve per il ripristino in caso di errori

Transazioni: esempio

```
Connection connection =
  DriverManager.getConnection(url, username, passwd);
connection.setAutoCommit(false);
try {
  statement.executeUpdate(...);
  statement.executeUpdate(...);

  connection.commit();
} catch (Exception e) {
  try {
    connection.rollback();
  } catch (SQLException sqle) {
     // report problem
  }
} finally {
  try {
   connection.close();
  } catch (SQLException sqle) { }
}
```

Metodi di Connection per la gestione delle transazioni

- `getAutoCommit/setAutoCommit`
  - Imposta o legge il valore della modalità auto-commit
- `commit`
  - Forza tutti i cambiamenti richiesti dall'ultima chiamata di commit
- `rollback`
  - Elimina i cambiamenti richiesti a partire dalla precedente chiamata di commit

Domande utili per verificare la propria preparazione per l'orale

Domande di verifica (1)

- Elencare tutte le tecniche note per il mantenimento di informazioni relative alla sessione di navigazione di un utente
- Descrivere l'utilizzo di campi input di tipo hidden nei form HTML per il mantenimento di informazioni sulla sessione di navigazione di un utente.
- Descrivere l'utilizzo di cookie per il mantenimento di informazioni sulla sessione di navigazione di un utente.

Domande di verifica (2)

- Descrivere tecniche di programmazione lato server basate sull'uso di oggetti persistenti per il mantenimento di informazioni relative alla sessione di navigazione di un utente
- Descrivere il ciclo di vita di una Servlet e di una HttpServlet
- Descrivere come l'interazione client-server possa scatenare l'esecuzione di programmi java da parte del server (protocolli e oggetti coinvolti)

Domande di verifica (3)

- Descrivere lo scopo del deployment descriptor indicandone i campi che si ritiene abbiano valore fondamentale nel funzionamento di una web application
- Descrivere e motivare la struttura delle cartelle di una web application
- Illustrare i metodi noti per la redirezione di richieste da una servlet ad altre risorse (servlet o pagine)

Domande di verifica (4)

- *Ciclo di vita di una pagina JSP*
- *Spiegare come avviene l'esecuzione e l'interpretazione del codice misto presente in una pagina JSP, in particolare descrivere l'elaborazione dei seguenti elementi:*
  - *direttive,*
  - *scriptlet,*
  - *espressioni,*
  - *dichiarazioni,*
  - *commenti html,*
  - *commenti jsp,*
  - *commenti java,*
  - *Azioni standard JSP: include,*
  - *Azioni standard JSP: forward,*
  - *Azioni standard JSP: useBean*
    - *Azioni standard JSP: setProperty,*
    - *Azioni standard JSP: getProperty*
  - *Tag personalizzati*

Domande di verifica (5)

- *Differenza tra direttiva e azione standard di inclusione*
- *Descrivere prima teoricamente e poi con un esempio l'utilità dei bean all'interno di pagine JSP*
- *Ciclo di vita di un tag che implementa l'interfaccia Tag*
- *Descrivere i metodi di autenticazione dichiarativa supportati dal container TOMCAT*
- *Descrivere un possibile approccio completamente programmato all'autenticazione*
- *Come si può usare il supporto JDBC da una servlet o da una pagina JSP?*