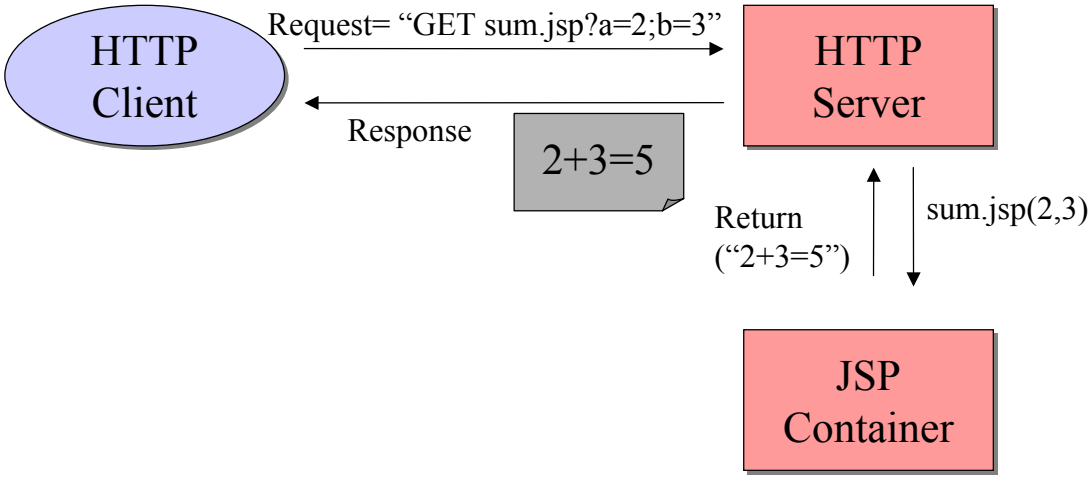
HTML, JavaScript e JSP

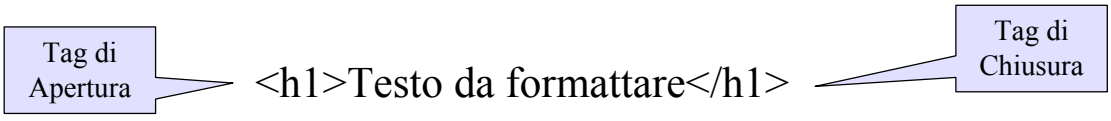


Il Linguaggio HTML

- HTML
  - HyperText Markup Language
- Sviluppato al CERN dal 1989
- Pubblicato nel 1991
  - Browser: Mosaic
- Web rende i servizi esistenti obsoleti
  - WAIS, Hytelnet, Gopher

Il Linguaggio HTML: Tag

- Linguaggio basato su marcatori che definiscono diverse proprietà del testo.



- Marcatori di formattazione senza testo.

```
<br/>
```

- Un tag può avere degli attributi

```
<meta name ="Author" content =“Pinco Pallino">
```

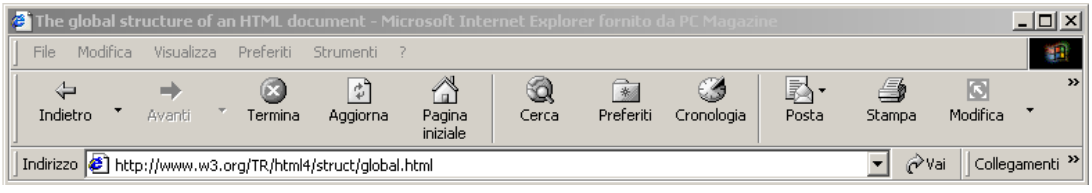
Il Linguaggio HTML: Struttura del Documento

```
<html>
    <head>
        .....
    </head>
    <body>
        .....
    </body>
</html>
```

Il Linguaggio HTML: Head

- Il tag <head> è opzionale
- Il tag <head> può contenere:
  - Il Tag <title>

    <title>The global structure of an HTML Document</title>



  - Più Meta Tags <meta>

    <meta name="Author" content="‘Pinco Pallino">

Il Linguaggio HTML: Meta Tags

- Definizione di coppie proprietà-valore:

  <meta name ="Author" content ="‘Pinco Pallino">
- Due tipi di utilizzo:
  - Definizione di una coppia nome valore generica

    <meta name="‘PropertyName" content="‘PropertyValue">
  - Definizione di una coppia nome valore per creare un header nell’HTTP Response

    <meta http-equiv="Expires" content ="Tue, 20 Aug 1996 14:25:27 GMT">

    Header Risultante → Expires: Tue, 20 Aug 1996 14:25:27 GMT

    (RFC2616)

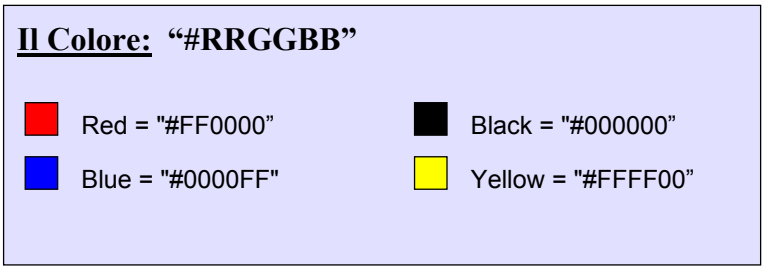
Il Linguaggio HTML: Body

- Il tag <body> delimita il corpo del documento.
- Alcuni Attributi:

background = *uri*
Definisce l'URI di una immagine da usare come sfondo per la pagina.

text = *color*
Definisce il colore del testo.

bgcolor= *color*
In alternativa a background definisce il colore di sfondo della pagina



Il Linguaggio HTML: Esempio

```
<html>
  <head>
    <title>A study of population dynamics</title>
  </head>
  <body bgcolor="white" text="black" link="red"
        alink="fuchsia" vlink="maroon">
    ... document body...
  </body>
</html>
```

Il Linguaggio HTML: Heading

- Creazione di un testo strutturato su più livelli:

<h1>Titolo più significativo</h1>

<h2>Un po' meno significativo </h2>

<h3> Un po' meno significativo </h3>

<h4> Un po' meno significativo </h4>

<h5> Un po' meno significativo </h5>

<h6> Titolo meno significativo </h6>

- Attributi:

align = left|center|right|justify

Il Linguaggio HTML: Testo Strutturato

- Elementi della frase:

Testo da enfatizzare

Enfasi ancor più forte

<cite>Frase o citazione</cite>

<code>static void main(String[] args) { }</code>

a²+b₀ → a^2+b_0

Il Linguaggio HTML: Font Style

- Formattazione del testo:

<tt>monospaced text</tt>	→	`monospaced text`
italic text	→	*italic text*
bold text	→	**bold text**
<big>big text</big>	→	big text
small text	→	small text
<u>underlined text</u>	→	underlined text

Il Linguaggio HTML: Tag Font

- Il tag <font> permette di formattare il testo.
- Attributi:
  - size = [+|-]*n*
    definisce le dimensioni del testo (1-7 o relative)
  - color = *color*
    definisce il colore del testo
  - face = *text*
    definisce il font del testo
- Il tag <basefont> definisce le impostazioni di default di un documento.

Il Linguaggio HTML: Horizontal Rule

- Il tag <hr> serve ad inserire una riga di separazione.
- Attributi:
  - align = *left|center|right*
    definisce l'allineamento della riga rispetto a ciò che la circonda
  - size = *pixels*
    definisce l'altezza della riga
  - width = *length*
    definisce la larghezza della riga in modo assoluto o in percentuale delle dimensioni di ciò che la contiene
  - noshade
    definisce se la riga deve essere "solida" o con un effetto di ombreggiatura.

```
<hr width="50%" align="center">
<hr size="5" width="50%" align="center">
<hr noshade size="5" width="50%" align="center">
```

Il Linguaggio HTML: Paragrafi

- Il Tag <p> delimita un paragrafo:

```
<p>Quel ramo del lago di Como, che volge a
mezzogiorno, tra due catene non interrotte di monti,
tutto a seni e a golfi, a seconda dello sporgere e del
rientrare di quelli, vien, quasi a un tratto, a ristringersi,
e a prender corso e figura di fiume, tra un promontorio
a destra, e un'ampia costiera dall'altra parte;<br/> e il
ponte, che ivi congiunge le due rive, par che renda
ancor piú sensibile all'occhio questa trasformazione, e
segni il punto in cui il lago cessa, e l'Adda
rincomincia, per ripigliar poi nome di lago dove le
rive, allontanandosi di nuovo, lascian l'acqua
distendersi e rallentarsi in nuovi golfi e in nuovi
seni.</p>
```

- Attributi del Tag <p>:

  align = left|center|right|justify

Il Linguaggio HTML: Liste

- Liste non ordinate: Type: disc,circle,square

```
<ul type="disc">
  <li>Unordered information.</li>
  <li>Ordered information.</li>
  <li>Definitions.</li>
</ul>
```

- Unordered information.
- Ordered information.
- Definitions.

- Liste ordinate: Type: 1 (1,2,..), a (a,b,..), A (A,B,..), i (i,ii,..), I (I,II,..)

```
<ol type="I">
  <li>Unordered information.</li>
  <li VALUE="4">Ordered information.</li>
  <li>Definitions. </li>
</ol>
```

I. Unordered information.
IV. Ordered information.
V. Definitions.

- Liste di definizione:

```
<dl>
  <dt><strong>UL</strong></dt>
  <dd>Unordered List.</dd>
  <dt><strong>OL</strong></dt>
  <dd>Ordered List.</dd>
</dl>
```

UL
Unordered List.
OL
Ordered List.

Esercitazioni: Progetto Rubrica

- Realizzazione di una applicazione web: Rubrica
- Rubrica Multiutente
  - Rubrica in grado di gestire contatti per più utenti
  - Riconoscimento utente mediante logon con username e password
- Realizzazione principali funzionalità
  - Visualizzazione contatti
  - Visualizzazione contatti per iniziale
  - Inserimento nuovo contatto
  - Modifica dati contatto
- Rubrica Multilingue
  - Riconoscimento lingua in base all'utente

Progetto Rubrica: 01HTML

- Realizzazione di una applicazione web: Rubrica

- 01HTML

  Visualizzazione di un elenco di contatti senza alcuna funzionalità dinamica realizzato mediante l'uso di liste a degli altri tag di formattazione visti finora.

Il Linguaggio HTML: Tabelle

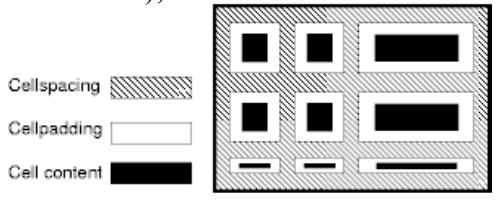
```
<table border="1" >
  <caption align="top"><em>A test table with merged cells</em></caption>
  <tr>
    <th rowspan="2"></th>
    <th colspan="2">Average</th>
    <th rowspan="2">Red<br/>eyes</th>
  </tr>
  <tr><th>height</th><th>weight</th></tr>
  <tr><th>Males</th><td>1.9</td><td>0.003</td><td>40%</td></tr>
  <tr><th>Females</th><td>1.7</td><td>0.002</td><td>43%</td></tr>
</table>
```

A test table with merged cells

	Average		Red eyes
	height	weight	
Males	1.9	0.003	40%
Females	1.7	0.002	43%

Il Linguaggio HTML: <table>

- Tag che racchiude la tabella.
- Attributi:
  - align = left|center|right
    allineamento della tabella rispetto alla pagina;
  - width="n|n%"
    larghezza della tabella (anche in percentuale rispetto alla pagina);
  - bgcolor="#xxxxxx"
    colore di sfondo della tabella;
  - border="n"
    spessore dei bordi della tabella (0 = tabella senza bordi);
  - cellspacing, cellpadding



Il Linguaggio HTML: <tr> <th> <td>

- <tr> è il tag che racchiude ciascuna riga della tabella.
- Attributi:
  - align = left|center|right|justify
    allineamento del contenuto delle cella della riga;
  - valign = top|middel|bottom|baseline
    allineamento verticale del contenuto delle cella della riga;
  - bgcolor="#xxxxxx"
    colore di sfondo della riga.
- <th> e <td> sono i tag che racchiudono la cella.
- Attributi:
  - Gli stessi di <tr>;
  - width,height = *length*
    specifica le dimensioni (larghezza e altezza) della cella, dimensione assoluta (pixels) o valore percentuale;
  - rowspan,colspan = *n*
    indica su quante righe,colonne della tabella si estende la cella.

Il Linguaggio HTML: Tabelle - Esempio

Il Linguaggio HTML: Tabelle - Esempio

Il Linguaggio HTML: Tabelle - Esempio

Il Linguaggio HTML: Links

- Il Link è il costrutto di base dell'ipertesto.
- Il Link è una connessione fra una risorsa Web ed un'altra.
- Concetto semplice, ma chiave per l'affermazione dell'HTML
- Link → due estremi detti *anchors* e una direzione
- source anchor → destination anchors
- Il destination anchors può essere una qualsiasi risorsa web
  (un immagine, un video, un eseguibile, un documento HTML)
- La risorsa di destinazione si ottiene visitando il link

Il Linguaggio HTML: Links - Esempio

```
<body>
   ...some text...
   <p>
      You'll find a lot more in
      <a href="chapter2.html">chapter two</a>.
      See also this
      <a href ="../images/forest.gif">
         map of the enchanted forest.
      </a>
   </p>
</body>
```


Il Linguaggio HTML: Links - Destination Anchors

```
……..
<h1>Table of Contents</h1>
<p>
   <a href="#section1">Introduction</a><br/>
   <a href ="#section2">Some background</a><br/>
   <a href ="#section2.1">On a more personal note</a><br/>
     ...the rest of the table of contents... ...the document body...
</p>
<h2><a name="section1">Introduction</a></h2>
    ...section 1...
<h2><a name ="section2">Some background</a></h2>
    ...section 2...
<h3><a name ="section2.1">On a more personal note</a></h3>
    ...section 2.1...
……..
```

Il Linguaggio HTML: Links - Destination Anchors

```
……..
<h1>Table of Contents</h1>
<p>
   <a href="#section1">Introduction</a><br/>
   <a href ="#section2">Some background</a><br/>
   <a href ="#section2.1">On a more personal note</a><br/>
      ...the rest of the table of contents... ...the document body...
</p>
<h2 id="section1">Introduction</h2>
   ...section 1...
<h2 id ="section2">Some background</h2>
   ...section 2...
<h3 id ="section2.1">On a more personal note</h3>
   ...section 2.1...
……..
```


Il Linguaggio HTML: Tag <img>

- Consente di inserire immagini in un documento HTML

- Attributi:
  - src = *uri*
    specifica l'indirizzo dell'immagine (required)
  - alt = *text*
    specifica un testo alternativo nel caso fosse impossibile visualizzare l'immagine
  - align = bottom|middle|top|left|right
    definisce la posizione dell'immagine rispetto al testo che la circonda
  - width,height = *length* (pixels)
    definisce larghezza e altezza dell'immagine
  - border = *pixels*
    definisce lo spessore del bordo dell'immagine (0 = nessun bordo)
  - hspace,vspace = *pixels*
    definisce lo spazio in orizzontale e in verticale fra l'immagine e ciò che la circonda.

Il Linguaggio HTML: Client Side Maps

- Le Mappe Client Side consentono di assegnare funzioni diverse a diverse regioni di una immagine
- Esempio:

```
<p><img src="navbar1.gif" usemap="#map1"></p>

        ...the rest of the page here...

<map name="map1">
   <area href="guide.html" alt="Access Guide"
         shape="rect" coords="0,0,118,28"/>
   <area href ="shortcut.html" alt ="Go"
         shape ="circle" coords ="184,200,60"/>
   <area href ="top10.html" alt ="Top Ten"
         shape ="poly" coords ="276,0,276,28,100,200,50,50,276,0"/>
   <area nohref shape ="default">
</map>
```


Il Linguaggio HTML: tag area

- Il tag area associa un'area dell'immagine ad un link
- Attributi:
  - shape = default|rect|circle|poly
    specifica la forma dell'area da definire
  - coords = *coordinates*
    - rect: left-x, top-y, right-x, bottom-y
    - circle: center-x, center-y, radius
    - poly: x1, y1, x2, y2, ..., xN, yN
  - href = *uri*
    destination anchor per l'area definita
  - nohref
    se specificato indica che nessun link è associato all'area definita
  - alt = *text*
    specifica un testo alternativo per l'area definita

Il Linguaggio HTML: Form

- Sezione del documento HTML che contiene elementi di controllo con cui l'utente può inserire dati o in generale interagire.
- I dati inseriti possono essere poi inoltrati ad un agente che può processarli.
- Gli elementi di controllo sono accessibili da parte di script client-side.
- Gli elementi di controllo sono caratterizzati da un valore iniziale e da un valore corrente.
- Gli elementi di controllo possone essere:
  - Bottoni
  - CheckBox (Switch on/off)
  - Radio Buttons (Switch mutuamente esclusi)
  - Menu di selezione (Lista di opzioni)
  - Inserimento di testo
  - Oggetti nascosti (elementi di controllo valorizzati ma invisibili)

Il Linguaggio HTML: Tag Form

- Racchiude tutti gli elementi della form
- Attributi:
  - action = *uri*
    specifica l'URI dell'agente che riceverà i dati della form
  - name = *text*
    specifica il nome della form
  - method = get|post
    specifica il modo in cui i dati vengono inviati

```
<form action="http://somesite.com/prog/adduser" method="post">
 ...form contents...
</form>
```

Il Linguaggio HTML: Text Input

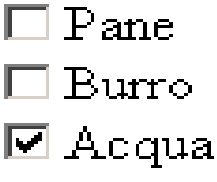
```
<form action="http://somesite.com/prog/adduser" method="post">
   <p>
      First name: <input type="text" name="firstname“/><br/>
   </p>
</form>
```

Attributi:

- name = *text*
  specifica il nome del controllo nella coppia nome/valore
- value = *text*
  specifica un eventuale valore iniziale
- size = *n*
  lunghezza del campo specificata in numero di caratteri
- maxlenght = *n*
  massima lunghezza del testo specificata in numero di caratteri

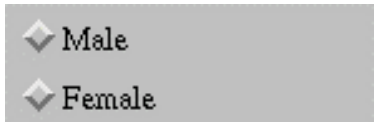
Il Linguaggio HTML: Checkbox

```
<form action="http://somesite.com/prog/adduser" method="post">
   <p>
      <input type=“checkbox" name=“food“ value=“pane”/>Pane<br/>
      <input type=“checkbox" name=“food“ value=“burro”/>Burro<br/>
      <input type=“checkbox" name=“drink“
            value=“acqua” checked=“checked”/>Acqua<br/>
   </p>
</form>
```



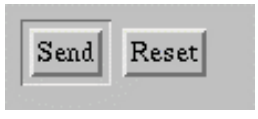
Il Linguaggio HTML: Radio

```
<form action="http://somesite.com/prog/adduser" method="post">
   <p>
      <input type="radio" name="sex" value="Male">Male<br/>
      <input type="radio" name="sex" value="Female">Female<br/>
   </p>
</form>
```



Il Linguaggio HTML: Buttons and Hidden

```
<form action="http://somesite.com/prog/adduser" method="post">
   <p>
      <input type="submit" value="Send"><br/>
      <input type="reset"><br/>
      <input type="image" src="invia.gif alt="Invia"><br/>
      <input type="button" name="check" value="Check"><br/>
      <input type="hidden" name="status" value="view"><br/>
   </p>
</form>
```



Il Linguaggio HTML: Menù di selezione

```
<form action="http://somesite.com/prog/component-select"
        method="post">
 <p>
  <select multiple size="3" name="component-select">
   <option selected value="Component_1_a">Component_1</option>
   <option selected value="Component_1_b">Component_2</option>
   <option>Component_3</option>
   <option>Component_4</option>
  </select>
  <input type="submit" value="Send"><< input type="reset">
 </p>
</form>
```

Progetto Rubrica: 02HTML

- Realizzazione di una applicazione web: Rubrica

- 02HTML

  Visualizzazione di un elenco di contatti con possibilità di spostarsi su l'unica pagina presente attraverso l'uso di anchor. Vengono inoltre utilizzate tabelle e immagini e viene impostata una pagina statica di form di inserimento per i contatti.

- 03HTML

  Variante realizzata su più pagine utilizzando link a pagine diverse.

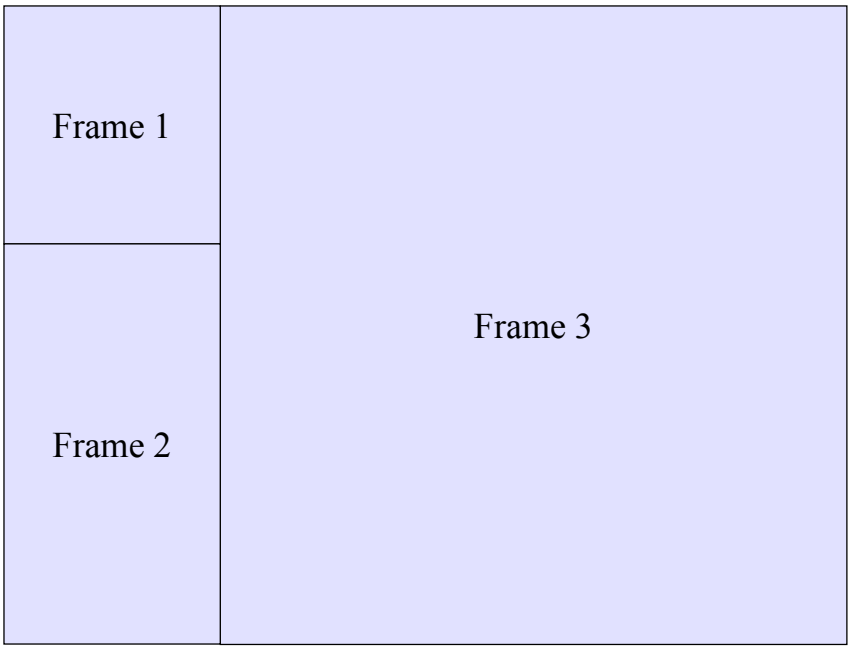
Il Linguaggio HTML: Frames

- Strumento che consente di presentare i documenti mediante viste multiple.
- E' possibile suddividere il documento in finestre o porzioni di finestra indipendenti.
- Offre la possibilità di mantenere sezioni del documento sempre visibili.
- Le porzioni del documento possono interagire e scambiare informazioni.

Il Linguaggio HTML: Frames

```
<html>
  <head>
    <title>A simple frameset document</title>
  </head>
  <frameset cols="20%, 80%">
    <frameset rows="100, 200">
      <frame src="contents_of_frame1.html">
      <frame src="contents_of_frame2.gif">
    </frameset>
    <frame src="contents_of_frame3.html">
  </frameset>
  <noframes>
    ……..
  </noframes>
</html>
```

Il Linguaggio HTML: Frames



Il Linguaggio HTML: Noframes

```
…….
<noframes>
  <p>
    This frameset document contains:
    <ul>
      <li><a href="contents_of_frame1.html">Some contents</a></li>
      <li><img src="contents_of_frame2.gif" alt="A neat image"></a></li>
      <li><a href="contents_of_frame3.html">Some other contents</a></li>
    </ul>
  </p>
</noframes>
…….
```

Il Linguaggio HTML: Tag Frameset

- Specifica il layout della struttura a frame della finestra.
- Attributi:
  - rows = *length,length,...*
    layout dei frame in senso orizzontale
  - cols = *length,length,...*
    layout dei frame in senso verticale

```
<FRAMESET rows="50%, 50%">
...the rest of the definition...
</FRAMESET>
```

```
<FRAMESET cols="1*,250,3*">
...the rest of the definition...
</FRAMESET>
```

```
<FRAMESET rows="30%,70%"
          cols="33%,34%,33%">
 ...the rest of the definition...
 </FRAMESET>
```

```
<FRAMESET rows="30%,400,*,2*">
 ...the rest of the definition...
</FRAMESET>
```


Il Linguaggio HTML: Nested Frameset

```
<html>
  <head>
    <title>A nested frameset document</title>
  </head>
  <frameset cols="33%, 33%, 34%">
    ...contents of first frame...
    <frameset rows="40%, 50%">
      ...contents of second frame, first row...
      ...contents of second frame, second row...
    </frameset>
    ...contents of third frame...
  </frameset>
</html>
```

Il Linguaggio HTML: Tag Frame

- Definisce le caratteristiche di ciascun frame.
- Attributi:

• name = *text*	specifica il nome del frame
• src = *uri*	specifica il contenuto iniziale del frame
• noresize	*se* specificato indica che il frame non può essere ridimensionato
• scrolling = *auto|yes|no*	specifica se il frame debba essere dotato di scrollbar
• frameborder = *1|0*	specifica se il frame debba avere dei bordi di separazione dei frame adiacenti
• marginwidth = *pixels*	quantità di spazio vuoto fra i bordi verticali del frame e il suo contenuto
• marginheight = *pixels*	quantità di spazio vuoto fra i bordi orizzontali del frame e il suo contenuto

Il Linguaggio HTML: Tag Frame

```
<html>
  <head>
    <title>A frameset document</title>
  </head>
  <frameset cols="33%,33%,33%">
    <frameset rows="*,200">
      <frame src="contents_of_frame1.html" scrolling="no">
      <frame src="contents_of_frame2.gif"
                marginwidth="10" marginheight="15" noresize>
    </frameset>
      <frame src="contents_of_frame3.html" frameborder="0">
      <frame src="contents_of_frame4.html" frameborder="0">
  </frameset>
</html>
```

Il Linguaggio HTML: Attributo Target

- Specifica il frame dove un nuovo document verrà aperto

  target = *frame-target*

- Tag che supportano l'attributo target:
  - A

    indica in quale frame deve essere aperto il destination anchor
  - AREA

    indica in quale frame deve essere aperto il destination anchor
  - FORM

    indica in quale frame deve essere aperto il documento di risposta al submit

Il Linguaggio HTML: frame-target

- Nome del frame:
  - Stringa alfanumerica che inzia con a-zA-Z
- Frame speciali:
  - _blank

    una nuova finestra
  - _self

    il frame stesso
  - _parent

    frame di gerarchia immediatamente superiore
  - _top

    la finestra originale che contiene la gerarchia di frame

Il Linguaggio HTML: Target - Esempio

```
<html>
  <head>
    <title>A frameset document</title>
  </head>
  <frameset rows="50%,50%">
    <frame name="fixed" src="init_fixed.html">
    <frame name="dynamic" src="init_dynamic.html">
  </frameset>
</html>
```

```
<html>
  <head>
    <title>A document with anchors with specific targets</title>
  </head>
  <body>
    <p>Now you may advance to
      <a href="slide2.html" target="dynamic">slide 2.</a></br>
      You're doing great. Now on to
      <a href="slide3.html" target="dynamic">slide 3.</a>
    </p>
  </body>
</html>
```


Progetto Rubrica: 04HTML

- Realizzazione di una applicazione web: Rubrica
- 04HTML

  Visualizzazione dell'elenco contatti mediante una struttura a frame

Il Linguaggio HTML: Style sheets

- L'HTML è nato in ambienti scientifici
- Linguaggio orientato più ai contenuti che alla presentazione
- Sono state fatte forzature per aggirare le limitazioni
  - Estensioni proprietarie
  - Conversione del testo in immagini
  - Uso di immagini per il controllo degli spazi bianchi
  - Uso delle tabelle per la formattazione
- Le forzature presentano spesso effetti collaterali
- Scopo degli Style sheets è risolvere parte di tali limitazioni

Fogli di stile (CSS)

•Standardizzazione curata da W3C (http://www.w3c.org)

•1996, CSS1: servono per modificare l'aspetto degli elementi nelle pagine HTML (colore, dimensione, …)

•1998, CSS2: permettono funzioni più sofisticate e

•introducono la possibilità di posizionamento assoluto per gli elementi nelle pagine HTML

Fogli di stile (CSS)

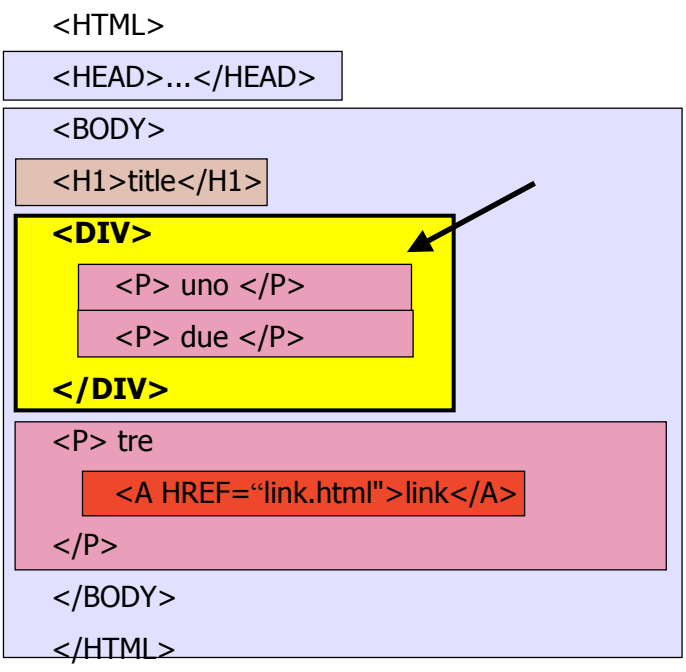
- CSS: **Cascading** Style Sheets
- Separazione del contenuto del documento dalla sua rappresentazione
- Gestione uniforme dell'aspetto di un insieme di pagine html

```
Esempio:
<FONT FACE="Arial"><I>Testo della pagina</I></FONT>
applicato a tutto un sito web!
```


CSS: vantaggi

- permettono di modificare il look & feel di un documento in modo efficiente
- lo stesso stile può essere applicato a più documenti
- il sorgente HTML è più pulito
- si possono progettare le pagine HTML per più browser

CSS: struttura della pagina



Un documento HTML può essere visto come un insieme di blocchi (contenitori) sui quali si può agire con stili diversi. Ogni tag HTML definisce un blocco.

CSS: rules

Foglio di stile

collezione di regole stilistiche che definiscono l'aspetto degli elementi

Regola

Elenco di proprietà. Sono coppie CHIAVE:VALORE

Le regole vengono associate ai tag

tag { proprietà1:valore1; proprietà2:valore2; ... }

CSS: proprietà

per lo sfondo

background-color

background-image

per i margini

margin-left

margin-right

margin-top

margin-bottom

per il testo

font-style

font-weight

font-size

font-family

text-align

text-transform

text-color

text-decoration

CSS: esempio

```
body { color:black; background:yellow; }
p { font-size:120%; font-style:italic; color:green; }
h1 { margin-left:10%; margin-right:10%; }
h2 { font-family: "Times New Roman", Arial; }
A:link { color:red; text-decoration:none; }
A:visited { color:blue; }
```

CSS: dove si definiscono

Embedded

```
<HEAD>
     <STYLE TYPE="text/css">
          H1 {color:blue; font-style:italic;}
          H2 {color:red; font-style:italic;}
          H3 {color:yellow; font-style:italic;}
          B {color:green; font-style:italic;}
     </STYLE>
</HEAD>
<BODY>...</BODY>
```

External

```
<HTML>
<HEAD>
   <LINK REL="stylesheet"
TYPE="text/css"
HREF="stile.css">
</HEAD>
```

Inline

```
<H1 STYLE="color:red; text-ransform:capitalize;">
Test di stile
</H1>
```


CSS: ereditarietà e cascade

- Vale il concetto di ereditarietà degli stili
- Si possono importare più fogli di stile nello stesso documento
- Conflitti risolti mediante le *cascade rules*
- Viene data priorità alle regole definite nel documento

CSS: classi e identificatori

```
<HEAD>
<STYLE TYPE="text/css">
H1 { font-style:italic; }
.hot { color:red; text-decoration:underline; }
</STYLE>
</HEAD>
<BODY>
<H1> Primo titolo </H1>
<H1 CLASS="hot"> Titolo da evidenziare </H1>
</BODY>
</HTML>
```

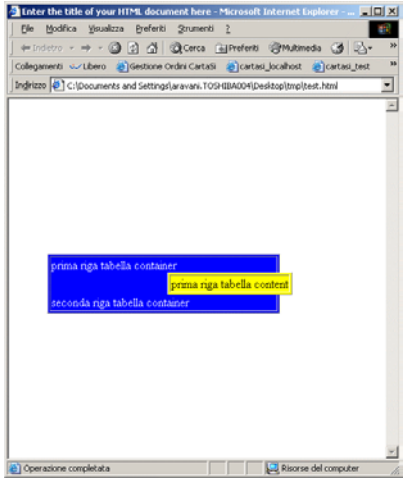
La definizione di classi permette di applicare lo stesso stile a tag differenti.

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
P { color:blue; }
#speciale { color:red; text-transform:uppercase; }
</STYLE>
</HEAD>
<BODY>
<I ID="speciale">test</I>
```

Gli identificatori, permettono di definire delle regole che si applicano ad un solo elemento in un documento

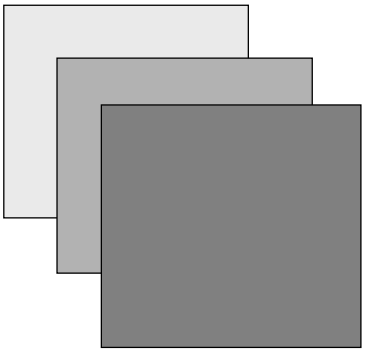
CSS: posizionamento

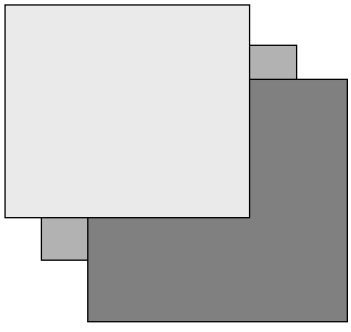
```
<html>
          <head>
<title>Enter the title of your HTML document here</title>
          </head>
          <body>
  <DIV STYLE="position:absolute; left:50; top:200;">
    <TABLE WIDTH="300" BGCOLOR="blue" BORDER="1">
      <TR>
        <TD>
          <FONT COLOR="#ffffff">
          prima riga tabella container
          <DIV STYLE="position:relative; left:150;">
          <TABLE BGCOLOR="yellow" BORDER="1">
             <TR><TD>prima riga tabella content</TD></TR>
          </TABLE>
          </DIV>
          seconda riga tabella container
          </FONT>
        </TD>
      </TR>
    </TABLE>
  </DIV>
</body>
</html>
```



CSS: z-index

In caso di sovrapposizione di elementi, l'attributo z-index, permette di definire la distribuzione degli elementi in *profondità* (z è la terza dimensione)

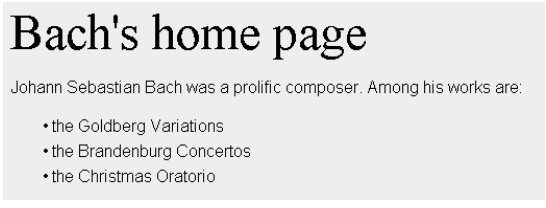




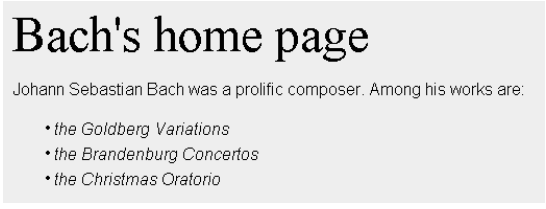
```
<STYLE TYPE="text/css">
#posiz1 { position:absolute; z-index:1; left: 50; top:50; }
#posiz2 { position:absolute; z-index:3; left:100; top:100; }
#posiz3 { position:absolute; z-index:2; left:150; top:150; }
</STYLE>
```


CSS: Esempio

```
H1 { font: 36pt serif }
```

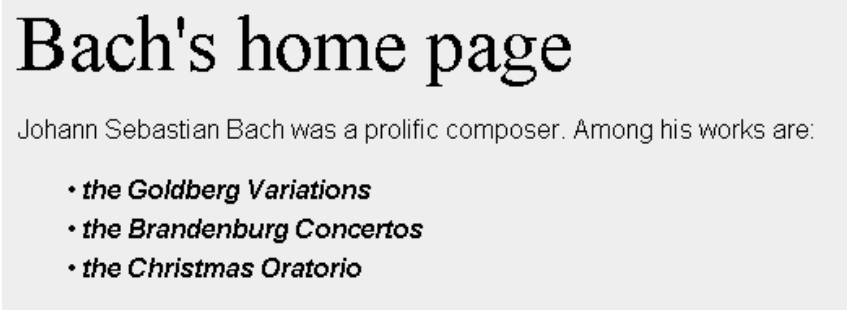



```
H1 {
  font-size: 36pt;
  font-family: serif;
  font-style: normal;
  font-weight: normal;
  font-variant: normal;
  line-height: normal;
}
UL { font-style: italic }
```



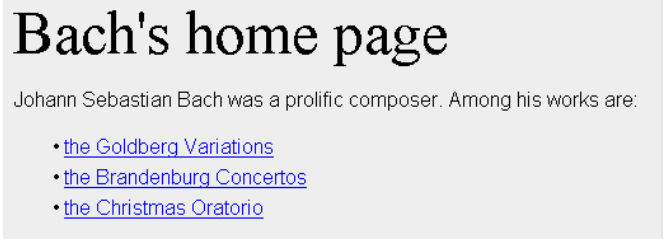
CSS: Esempio

```
UL {
  font-style: italic;
  font-weight: bold;
}
```

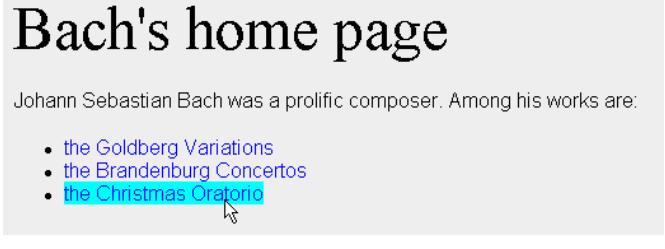


CSS: Esempio

```
A:link { text-decoration: underline }
```




```
A:link, A:visited { text-decoration: none }
A:hover { background: cyan }
```



CSS: Riferimenti

- World wide web Consortium CSS Reference:
  http://www.w3.org/Style/CSS/

- World wide web Consortium CSS2 Specifications:
  http://www.w3.org/TR/REC-CSS2/

Progetto Rubrica: 05CSS

- Realizzazione di una applicazione web: Rubrica

- 05CSS

  Esempio di utilizzo degli style sheets per la formattazione delle pagine di visualizzazione contatti del progetto rubrica

Javascript: Client Side Script

- Uno Script Client Side è un programma contenuto in un documento HTML
- Il supporto del linguaggio HTML è indipendente dal linguaggio di scripting
- Uno Scipt Client Side offre numerose possibilità di estendere il documento:
  - modifica dinamica del documento in fase di caricamento
  - modifica dinamica e elaborazione dei contenuti di una form
  - esecuzione di operazioni al verificarsi di un evento
- In un documento possono essere presenti due tipi di script:
  - script eseguiti una sola volta nella fase di caricamento
  - script eseguiti al verificarsi di un evento

Javascript: HTML – Tag script

- Permette di includere uno script in un documento HTML
- Può essere usato più volte sia nella sezione <head> che nella sezione <body>
- Attributi:
  - src = *uri*
    specifica l'indirizzo di uno script in un file esterno
  - type = *content-type*
    specifica il linguaggio di scripting utilizzato
    (sostituisce language che è deprecato)
- E' possibile specificare un contenuto per browser che non supportano lo scripting con il tag <noscript>

Javascript

- Linguaggio di scripting a oggetti, cross-platform di Netscape
- Possiede un core di oggetti, funzioni, operatori e strutture di controllo
- *Client-side Javascript* estende il core con oggetti per il controllo del browser e del DOM (Document Object Model)
- Anche se si assomigliano Javascript è molto diverso da Java

JavaScript	**Java (Applet)**
Interpreted (not compiled) by client.	Compiled bytecodes downloaded from server, executed on client.
Object-oriented. No distinction between types of objects. Inheritance is through the prototype mechanism, and properties and methods can be added to any object dynamically.	Class-based. Objects are divided into classes and instances with all inheritance through the class hierarchy. Classes and instances cannot have properties or methods added dynamically.
Code integrated with, and embedded in, HTML.	Applets distinct from HTML (accessed from HTML pages).
Variable data types not declared (dynamic typing).	Variable data types must be declared (static typing).
Cannot automatically write to hard disk.	Cannot automatically write to hard disk.

Javascript: Valori e Variabili

- Javascript riconosce i seguenti tipi di valori:
  - Numerici (interi o floating point)
  - Booleani (true or false)
  - Stringhe
  - *null*
  - *undefined*
- Javascript è un linguaggio a tipizzazione dinamica:

```
var answer = 42;
answer = "Thanks for all the fish...“;
```

- Variabili:
  - Il nome deve iniziare con una lettera o con _
  - Javascript è case sensitive
  - Assegnazione: x=42; var x=42;
  - Una variabile non assegnata è *undefined*
  - Scope: *global,local*

Javascript: Operatori di Assegnamento

Shorthand operator	Meaning
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y
x <<= y	x = x << y
x >>= y	x = x >> y
x >>>= y	x = x >>> y
x &= y	x = x & y
x ^= y	x = x ^ y
x |= y	x = x | y

Javascript: Operatori di Confronto

Operator	Description	Examples returning true
Equal (==)	Returns true if the operands are equal. If the two operands are not of the same type, JavaScript attempts to convert the operands to an appropriate type for the comparison.	3 == var1 "3" == var1 3 == '3'
Not equal (!=)	Returns true if the operands are not equal. If the two operands are not of the same type, JavaScript attempts to convert the operands to an appropriate type for the comparison.	var1 != 4 var2 != "3"
Strict equal (===)	Returns true if the operands are equal and of the same type.	3 === var1
Strict not equal (!==)	Returns true if the operands are not equal and/or not of the same type.	var1 !== "3" 3 !== '3'
Greater than (>)	Returns true if the left operand is greater than the right operand.	var2 > var1
Greater than or equal (>=)	Returns true if the left operand is greater than or equal to the right operand.	var2 >= var1 var1 >= 3
Less than (<)	Returns true if the left operand is less than the right operand.	var1 < var2
Less than or equal (<=)	Returns true if the left operand is less than or equal to the right operand.	var1 <= var2 var2 <= 5

Javascript: Operatori Aritmetici

Operator	Description	Example
% (Modulus)	Binary operator. Returns the integer remainder of dividing the two operands.	12 % 5 returns 2.
++ (Increment)	Unary operator. Adds one to its operand. If used as a prefix operator (++x), returns the value of its operand after adding one; if used as a postfix operator (x++), returns the value of its operand before adding one.	If x is 3, then ++x sets x to 4 and returns 4, whereas x++ sets x to 4 and returns 3.
-- (Decrement)	Unary operator. Subtracts one to its operand. The return value is analogous to that for the increment operator.	If x is 3, then --x sets x to 2 and returns 2, whereas x++ sets x to 2 and returns 3.
- (Unary negation)	Unary operator. Returns the negation	

Javascript: Operatori bit a bit

Operator	Usage	Description
Bitwise AND	*a* & *b*	Returns a one in each bit position for which the corresponding bits of both operands are ones.
Bitwise OR	*a* | *b*	Returns a one in each bit position for which the corresponding bits of either or both operands are ones.
Bitwise XOR	*a* ^ *b*	Returns a one in each bit position for which the corresponding bits of either but not both operands are ones.
Bitwise NOT	~ *a*	Inverts the bits of its operand.
Left shift	*a* << *b*	Shifts a in binary representation b bits to left, shifting in zeros from the right.
Sign-propagating right shift	*a* >> *b*	Shifts a in binary representation b bits to right, discarding bits shifted off.
Zero-fill right shift	*a* >>> *b*	Shifts a in binary representation b bits to the right, discarding bits shifted off, and shifting in zeros from the left.

Javascript: Operatori logici

&&	expr1 && expr2	(Logical AND) Returns expr1 if it can be converted to false; otherwise, returns expr2. Thus, when used with Boolean values, && returns true if both operands are true; otherwise, returns false.
||	expr1 || expr2	(Logical OR) Returns expr1 if it can be converted to true; otherwise, returns expr2. Thus, when used with Boolean values, || returns true if either operand is true; if both are false, returns false.
!	!expr	(Logical NOT) Returns false if its single operand can be converted to true; otherwise, returns true.

Javascript: Operatori Speciali

- new

  crea un'istanza di un oggetto

- this

  referenzia l'oggetto corrente

- typeof

  ritorna il tipo di un oggetto

- void

  valuta una espressione senza tornare alcun valore

Javascript: Operatori - Precedenza

Operator type	Individual operators
comma	,
assignment	= += -= *= /= %= <<= >>= >>>= &= ^= |=
conditional	?:
logical-or	||
logical-and	&&
bitwise-or	|
bitwise-xor	^
bitwise-and	&
equality	== !=
relational	< <= > >=
bitwise shift	<< >> >>>
addition/subtraction	+ -
multiply/divide	* / %
negation/increment	! ~ - + ++ -- typeof void delete
call	()
create instance	new
member	. []

Javascript: Istruzioni Condizionali

```
if (condition) {
  statements1
} [else {
  statements2
} ]
```

```
switch (expression)
{
  case label :
    statement;
    break;
  case label :
    statement;
    break;
  ...
  default :
statement;
}
```

Javascript: Loop

```
for ([initialExpression]; [condition]; [incrementExpression]) {
  statements
}
```

```
do {
  statement
} while (condition)
```

```
while (condition) {
  statements
}
```


Javascript: Functions

- Definizione di Funzione
  (Parametri passati sempre per valore)

```
function square(number) {
  return number * number;
}
```

- Chiamata a Funzione

```
square(5);
```

- E' possibile realizzare funzioni con numero variabile di parametri

Javascript: Funzioni predefinite

- eval(*expr*)
  valuta la stringa *expr* (espressione,istruzioni)
- isFinite(*number*)
  number=NaN,+inf,-inf → false
- isNaN(*testValue*)
  testValue = NaN → true
- parseInt(*str, [radix]*)
  converte la string *str* in un intero in base *radix*
- parseFloat(*str*)
  converte la string *str* in un floating point

Javascript: Oggetti predefiniti (Core)

- Array
- Boolean
- Date
- Math
- Number
- String
- ……..

Javascript: Oggetto Array

- Creazione di un Array

```
arrayObjectName = new Array(element0, element1, ..., elementN)
arrayObjectName = new Array(arrayLength)
```

- Popolamento di un Array

```
emp[1] = "Casey Jones"
emp[2] = "Phil Lesh"
emp[3] = "August West“
myArray = new Array("Hello", myVar, 3.14159)
```

- Riferimento ad un elemento

```
myArray[0]
```

- E’ possibile definire Array multidimensionali

Javascript: Gestione degli Eventi

- Le applicazioni Javascript sono principalmente basate sugli *eventi*
- Gli *eventi* si verificano come risultato di una azione dell’utente
- Per attivare uno script allo scatenarsi di un evento è necessario definire un gestore dell’evento (*Event Handler*)

Javascript: Eventi

Event	Applies to	Occurs when	Event handler
Abort	images	User aborts the loading of an image (for example by clicking a link or clicking the Stop button)	onAbort
Blur	windows and all form elements	User removes input focus from window or form element	onBlur
Change	text fields, textareas, select lists	User changes value of element	onChange
Click	buttons, radio buttons, checkboxes, submit buttons, reset buttons, links	User clicks form element or link	onClick
DragDrop	windows	User drops an object onto the browser window, such as dropping a file on the browser window	onDragDrop
Error	images, windows	The loading of a document or image causes an error	onError
Focus	windows and all form elements	User gives input focus to window or form element	onFocus
KeyDown	documents, images, links, text areas	User depresses a key	onKeyDown
KeyPress	documents, images, links, text areas	User presses or holds down a key	onKeyPress
KeyUp	documents, images, links, text areas	User releases a key	onKeyUp

Javascript: Eventi

Event	Applies to	Occurs when	Event handler
Load	document body	User loads the page in the Navigator	onLoad
MouseDown	documents, buttons, links	User depresses a mouse button	onMouseDown
MouseMove	nothing by default	User moves the cursor	onMouseMove
MouseOut	areas, links	User moves cursor out of a client-side image map or link	onMouseOut
MouseOver	links	User moves cursor over a link	onMouseOver
MouseUp	documents, buttons, links	User releases a mouse button	onMouseUp
Move	windows	User or script moves a window	onMove
Reset	forms	User resets a form (clicks a Reset button)	onReset
Resize	windows	User or script resizes a window	onResize
Select	text fields, textareas	User selects form element's input field	onSelect
Submit	forms	User submits a form	onSubmit
Unload	document body	User exits the page	onUnload

Javascript: Event Handler

- E' possibile definire un event handler per un tag che supporta il relativo evento.

  <tag eventHandler="JavaScript Code">

- Esempio

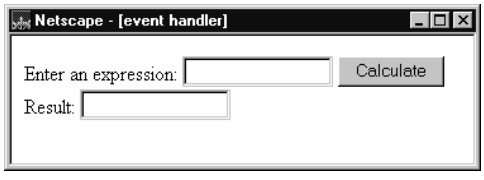
  <input type="button" value="Calculate" onClick="compute(this.form)"/>

- E' possibile inserire più istruzioni in sequenza, ma è meglio definire delle funzioni.
- E' necessario alternare i doppi apici con l'apice singolo

  <input type="button" name="Button1" value="Open Sesame!"
  onClick="window.open('mydoc.html', 'newWin')">

Javascript: Event Handler

```
<head>
 <script>
  <!--- Hide script from old browsers
   function compute(f) {
    if (confirm("Are you sure?"))
     f.result.value = eval(f.expr.value)
    else
     alert("Please come back again.")
   }
  // end hiding from old browsers -->
 </script>
</head>
<body>
 <form>
  Enter an expression:
  <input type="text" name="expr" size=15 >
  <input type="button" value="Calculate" onClick="compute(this.form)">
  <br/>
  Result:
  <input type="text" name="result" size="15" >
 </form>
</body>
```



Javascript: Event Handler

```
<script language="JavaScript">
  function fun1() {
    ...
  }
  function fun2() {
    ...
  }
</script>


<form name="myForm">
  <input type="button" name="myButton“ onClick="fun1()">
</form>


<script>
  document.myForm.myButton.onclick=fun2;
</script>
```


Javascript: Validare una input form

- Riduce il carico delle applicazioni server side filtrando l’input
- Riduce il ritardo in caso di errori di inserimento dell’utente
- Semplifica le applicazioni server side
- E’ possibile validare un documento in due momenti:
  - Durante l’inserimento utilizzando l’evento onChange
  - Al momento del submit della form

Javascript: Validare una input form

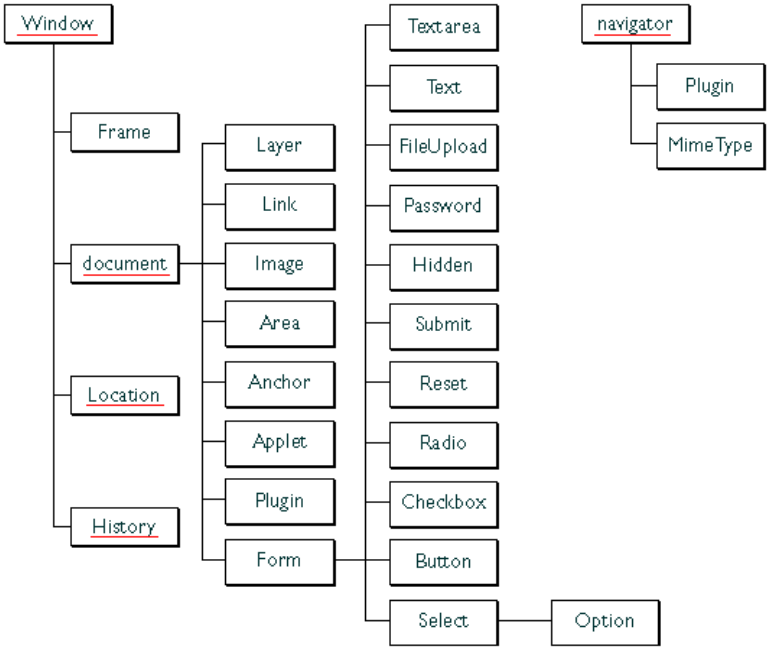
```
<head>
 <script>
  function isaPosNum(s) {
   return (parseInt(s) > 0)
  }

  function qty_check(item, min, max) {
   var returnVal = false;
   if (!isaPosNum(item.value))
    alert("Please enter a positive number");
   else if (parseInt(item.value) < min)
    alert("Please enter a " + item.name + " greater than " + min);
   else if (parseInt(item.value) > max)
    alert("Please enter a " + item.name + " less than " + max);
   else
    returnVal = true;
   return returnVal;
  }

  function validateAndSubmit(theform) {
   if (qty_check(theform.quantity, 0, 999)) {
    alert("Order has been Submitted");
    return true;
   } else {
    alert("Sorry, Order Cannot Be Submitted!");
    return false;
   }
  }
 </script>
</head>
```


Javascript: Validare una input form

```
<body>
 <form name="widget_order" action="lwapp.html" method="post">
  How many widgets today?
  <input type="text" name="quantity" onchange="qty_check(this, 0, 999)">
  <br>
  <input type="submit" value="Enter Order“
         onclick="validateAndSubmit(this.form)">
 </form>
</body>
```

```
<form name="widget_order" action="lwapp.html" method="post“
      onSubmit="return qty_check(theform.quantity, 0, 999)">
 ...
 <input type="submit">
 ...
</form>
```

Javascript: Browser Objects



Javascript: Browser Objects

```
<head>
  <title>A Simple Document</title>
  <script>
    function update(form) {
      alert("Form being updated")
    }
  </script>
</head>
<body>
  <form name="myform" action="foo.cgi" method="get" >
    Enter a value:
    <input type="text" name="text1" value="blahblah" size="20" >
    Check if you want:
    <input type="checkbox" name="Check1" checked="checked"
           onClick="update(this.form)">
    Option #1
    <br/>
    <input type="button" name="button1" value="Press Me"
           onClick="update(this.form)">
  </form>
</body>
```

Property	Value
document.title	"A Simple Document"
document.fgColor	#000000
document.bgColor	#ffffff
location.href	"http://www.royalairways.com/samples/simple.html"
history.length	7

Javascript: Browser Objects

```
<form name="myform" action="foo.cgi" method="get" >
   Enter a value:
   <input type="text" name="text1" value="blahblah" size="20" >
   Check if you want:
   <input type="checkbox" name="Check1" checked="checked"
          onClick="update(this.form)">
   Option #1
   <br/>
   <input type="button" name="button1" value="Press Me"
          onClick="update(this.form)">
 </form>
```

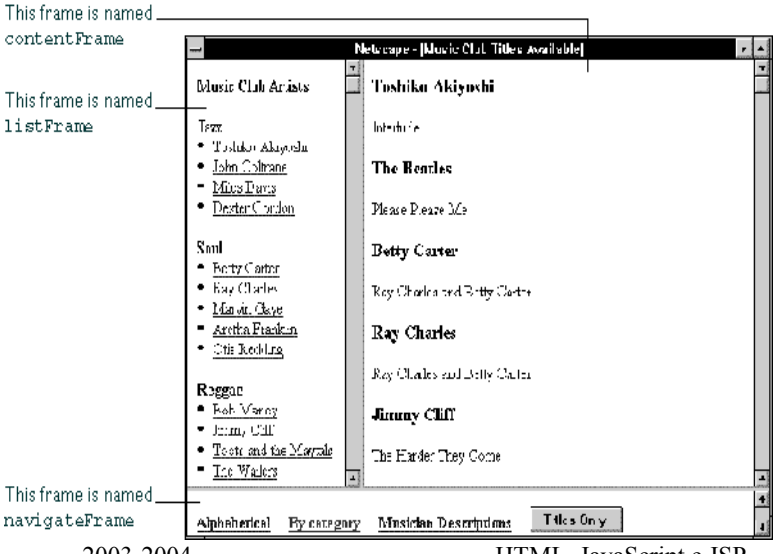
- document.myform is the form
- document.myform.Check1 is the checkbox
- document.myform.button1 is the button
- document.myform.action is http://www.royalairways.com/samples/mycgi.cgi, the URL to which the form is submitted.
- document.myform.method is "get," based on the value of the METHOD attribute.
- document.myform.length is 3, because there are three input elements in the form.
- document.myform.button1.value is "Press Me"
- document.myform.button1.name is "Button1"
- document.myform.text1.value is "blahblah"

Javascript: Window and Frame Object

- L'oggetto window è "padre" di tutti gli oggetti del browser
- Un frame è del tutto identico ad una window
- Alcuni metodi:
  - open e close
  - alert
  - confirm
  - prompt
  - blur e focus
- Alcune proprietà:
  - location
  - status

Javascript: Frames

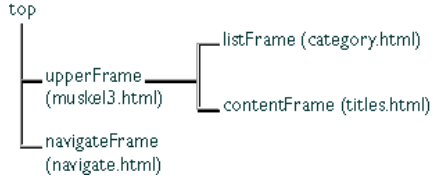
```
<frameset rows="90%,10%">
  <frameset cols="30%,70%">
    <frame src=category.html name="listFrame">
    <frame src=titles.html name="contentFrame">
  </frameset>
  <frame src=navigate.html name="navigateFrame">
</frameset>
```

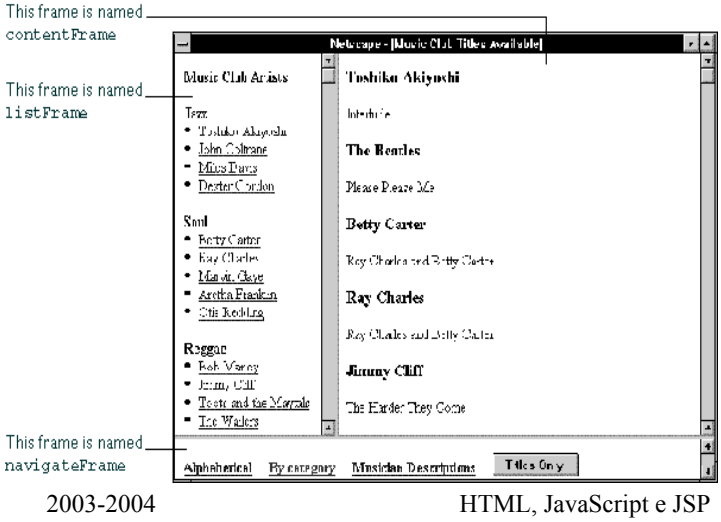


- listFrame is top.frames[0]
- contentFrame is top.frames[1]
- navigateFrame is top.frames[2]

Javascript: Frames

```
<frameset rows="90%,10%">
  <frame src=muskel3.html name="upperFrame">
  <frame src=navigate.html name="navigateFrame">
</frameset>
*** muskel3.html ***
<frameset cols="30%,70%">
  <frame src=category.html name="listFrame">
  <frame src=titles.html name="contentFrame">
</frameset>
```





- upperFrame is top.frames[0]
- navigateFrame is top.frames[1]
- listFrame is upperFrame.frames[0] or top.frames[0].frames[0]
- contentFrame is upperFrame.frames[1] or top.frames[0].frames[1]

Javascript: Frames

- E' possibile fare l'update di un frame mediante la proprietà *location*

```
<input type="button" value="Titles Only"
        onClick="top.frames[0].location='artists.html'">
```

- E possibile riferire un frame o una window in più modi:
  - Proprietà, Metodi e Gestori di eventi (variabile window)
    - self o window: *finestra corrente (default)*
    - top o parent: *finestra origine della gerarchio o frame padre*
    - nome di variabile di tipo window o frame
  - Form e Link (window name)

```
<form>
  <input type="button" value="Open Second Window"
          onClick="msgWindow=
          window.open('', 'window2' , 'resizable=no,width=200,height=200' )">
<a href="doc2.html" target="window2">Load a file into window2</A>
<input type="button" value="Close Second Window"
        onClick="msgWindow.close()">
</form>
```


Javascript: Document Object

- E' il contenitore di tutti gli elementi della pagina (Form, immagini, Link)
- Alcuni metodi:
  - write
  - writeln
- Alcune proprietà:
  - bgcolor
  - fgcolor
  - lastModified
  - cookie

Javascript: Form Object

- Un documento può contenere più oggetti form
- L'oggetto form può essere referenziato con il suo nome o col vettore forms[]
  document.nomeForm
  document.forms[n]
  document.forms["nomeForm"]
- Gli elementi della form possono essere referenziati con il loro nome o col vettore elements[]
  document.nomeForm.nomeElemento
  document.forms[n].elements[m]
- Ogni elemento della form ha una proprietà che la riferisce

```
<form name="myForm">
 Form name:
 <input type="text" name="text1" value="test">
 <br/>
 <input name="button1" type="button" value="Show Form Name"
        onclick="this.form.text1.value=this.form.name">
</form>
```


Javascript: Form Object

- Alcune Proprietà:
  - action — riflette l'attributo action
  - elements — vettore contenente gli elementi della form
  - length — numero di elementi nella form
  - method — riflette l'attributo method
  - name — nome della form
  - target — riflette l'attributo target
- Alcuni Metodi:
  - reset — resetta la form
  - submit — esegue il submit
- Eventi:
  - onreset — evento scatenato quando la form viene resettata
  - onsubmit — evento scatenato quando viene eseguito il submit della form

Javascript: Form Object

```
…
<head>
  <title>Form object example</title>
  <script>
    function setCase (caseSpec) {
     if (caseSpec == "upper") {
       document.myForm.firstName.value=document.myForm.firstName.value.toUpperCase();
       document.myForm.lastName.value=document.myForm.lastName.value.toUpperCase();
     } else {
       document.myForm.firstName.value=document.myForm.firstName.value.toLowerCase();
       document.myForm.lastName.value=document.myForm.lastName.value.toLowerCase();
     }
    }
  </script>
 </head>
 <body>
  <form name="myForm">
    <b>First name:</b>
    <input type="text" name="firstName" size="20"/><br/>
    <b>Last name:</b>
    <input type="text" name="lastName" size="20"/>
    <p>
     <input type="button" value="Names to uppercase" name="upperButton"
            onClick="setCase('upper')">
     <input type="button" value="Names to lowercase" name="lowerButton"
            onClick="setCase('lower')">
    </P>
  </form>
```


Javascript: Form Object

```
…
<head>
  <title>Form object/onSubmit event handler example</title>
  <script>
    var dataOK=false;
    function checkData () {
      if (document.myForm.threeChar.value.length == 3)
        return true;
      else {
        alert("Enter exactly three characters. " +
              document.myForm.threeChar.value + " is not valid.");
        return false;
      }
    }
  </script>
 </head>
 <body>
  <form name="myForm" onSubmit="return checkData()">
    <b>Enter 3 characters:</b>
    <input type="text" name="threeChar" size=3/>
    <p>
     <input type="submit" value="Done" name="submit1"
            onClick="document.myForm.threeChar.value=
                     document.myForm.threeChar.value.toUpperCase()"/>
    </p>
  </form>
```

Javascript: Form Object

```
…
<head>
   <title>Form object/submit method example</title>
   <script>
     var dataOK=false;
     function checkData () {
       if (document.myForm.threeChar.value.length == 3)
         document.myForm.submit();
       else {
         alert("Enter exactly three characters. " + document.myForm.threeChar.value +
                " is not valid.");
         return false;
       }
     }
   </script>
 </head>
 <body>
   <form name="myForm" onSubmit="alert('Form is being submitted.')">
     <b>Enter 3 characters:</b>
     <input type="text" name="threeChar" size="3"/>
     <p>
       <input type="button" value="Done" name="button1" onClick="checkData()"/>
     </p>
   </form>
…
```


Javascript: Checkbox Object

- Alcune Proprietà:
  - checked — booleano che definisce lo stato del checkbox (on/off)
  - defaultchecked — riflette l'attributo *checked*
  - form — specifica la form che lo contiene
  - name — nome del checkbox
  - value — riflette l'attributo *value*
- Alcuni Metodi:
  - blur — toglie il focus dal checkbox
  - focus — pone il focus sul checkbox
  - click — simula il click del mouse sul checkbox
- Eventi:
  - onblur — evento scatenato quando il checkbox perde il focus
  - onfocus — evento scatenato quando il checkbox prende il focus
  - onclick — evento scatenato quando l'utente clicca sul checkbox

Javascript: Checkbox Object

```
…
<head>
  <title>Checkbox object example</title>
  <script>
    function convertField(field) {
      if (document.form1.convertUpper.checked)
        field.value = field.value.toUpperCase();
    }
    function convertAllFields() {
      document.form1.lastName.value = document.form1.lastName.value.toUpperCase();
      document.form1.firstName.value = document.form1.firstName.value.toUpperCase();
      document.form1.cityName.value = document.form1.cityName.value.toUpperCase();
    }
  </script>
</head>
<body>
  <form name="form1">
    <b>Last name:</b>
    <input type="text" name="lastName" size="20" onChange="convertField(this)"/>
    <br/><b>First name:</b>
    <input type="text" name="firstName" size="20" onChange="convertField(this)"/>
    <br/><b>City:</b>
    <input type="text" name="cityName" size="20" onChange="convertField(this)"/>
    <p>
      <input type="checkBox" name="convertUpper"
             onClick="if (this.checked) {convertAllFields()}"> Convert fields to upper case
    </p>
  </form>
…
```


Javascript: Riferimenti

- Client-Side JavaScript Guide:

  http://developer.netscape.com/docs/manuals/js/client/jsguide/index.htm

- Client-Side JavaScript Reference:

  http://developer.netscape.com/docs/manuals/js/client/jsref/index.htm

Progetto Rubrica: 06JS

- Realizzazione di una applicazione web: Rubrica

- 06JS

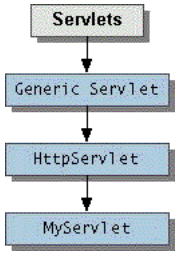
  Implementazione di alcune tecniche di controllo dei dati in ingresso realizzate mediante javascript per la pagina di inserimento di un nuovo contatto nel progetto rubrica

Servlet

- Le Servlet sono moduli che estendono le funzionalità dei server basati sul modello request/response.
- Le Servlet sono una tecnologia cross-platform in quanto le API non fanno nessuna assunzione sull'ambiente di esecuzione o sul protocollo.
- L'uso più diffuso è con i server HTTP.
- Le Servlet possono sostiutire compltamente gli script CGI (maggior semplicità, efficienza, portabilità)

Servlet: Java Servlet Package

- java.servet.*
- *Servlet* interface



- ServletRequest, HttpServletRequest
  - Parametri, protocollo, client info
  - ServletInputStream, per ricevere dati dal client
- ServletResponse, HttpServletResponse
  - Definisce la natura dei contenuti della risposta
  - ServletOutputStream, stream di invio della risposta

Servlet: Esempio

```
public class SimpleServlet extends HttpServlet {
  /**
   * Handle the HTTP GET method by building a simple web page.
   */
  public void doGet (HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
    PrintWriter out;
    String title = "Simple Servlet Output";
    // set content type and other response header fields first
    response.setContentType("text/html");
    // then write the data of the response
    out = response.getWriter();
    out.println("<html><head><title>");
    out.println(title);
    out.println("</title></head><body>");
    out.println("<h1>" + title + "</h1>");
    out.println("<p>This is output from SimpleServlet.</p>");
    out.println("</body></html>");
    out.close();
  }
}
```

Servlet: HttpServletRequest

- Consente l'accesso agli header HTTP:
  - Accesso ai cookie
  - Accesso al request method (GET,POST)
  - Accesso ai parametri della request
- Accesso ai dati della richiesta del client:
  - getParameter, getParameterValues, getParameterNames
  - GET: getQueryString
  - POST: getReader, getImputStream

Servlet: HttpServletResponse

- Costruzione del response header:
  - content type
- Costruzione della risposta per il client:
  - text,html: getWriter
  - binary: getOutputStream

Servlet: GET Request

```
public class BookDetailServlet extends HttpServlet {
  public void doGet (HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {

    ...
    // set content-type header before accessing the Writer
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    // then write the response
    out.println("<html>" + "<head><title>Book Description</title></head>" + ...);
    //Get the identifier of the book to display
    String bookId = request.getParameter("bookId");
    if (bookId != null) {
      // and the information about the book and print it
      ...
    }
    out.println("</body></html>");
    out.close();
  }
  ...
}
```


Servlet: POST Request

```
public class ReceiptServlet extends HttpServlet {

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

    ...
    // set content type header before accessing the Writer
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    // then write the response
    out.println("<html>" + "<head><title> Receipt </title>" + ...);
    out.println("<h3>Thank you for purchasing your books from us " +
                request.getParameter("cardname") + ...);
    out.close();

  }
  ...
}
```

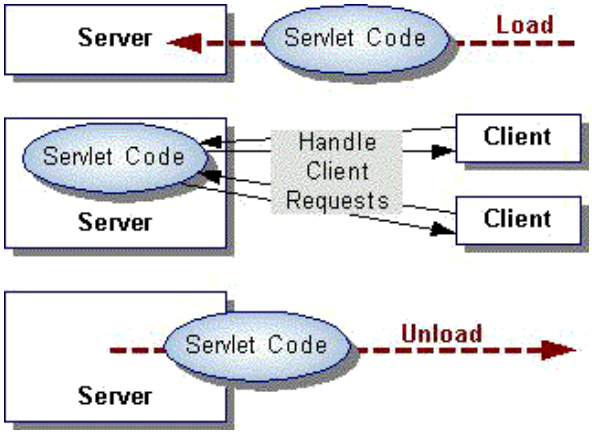
Servlet: Concorrenza

- Le servlet sono in grado di rispondere a richieste concorrenti.
- Se la servlet accede a risorse condivise è necessario sincronizzare gli accessi alle suddette risorse.
- E' altresì possibile creare servlet che gestiscono una sola richiesta alla volta.

```
public class ReceiptServlet extends HttpServlet
                                  implements SingleThreadModel {
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
                throws ServletException, IOException {
    ...
  }
  ...
}
```


Servlet: Life Cycle

- Il server **carica** e **inizializza** la servlet
- La servlet **gestice** zero o più richieste dei client
- Il server **rilascia** la servlet



Servlet: Inizializzazione

- Il server **carica** e **inizializza** la servlet
- Viene chiamato il metodo *init* (una sola volta)
- E' possibile costruire un metodo *init* in overriding
- Se vi sono errori nell'inizializzazione bisogna lanciare una Unavailable Exception
- La Servlet potrà essere reinizializzata solo dopo la sua distruzione

Servlet: Distruzione

- Una Servlet vive finche il server non la distrugge
- Viene chiamato il metodo *destroy*
- E' possibile costruire un metodo *destroy* in overriding

Servlet: Session Tracking

- Meccanismo per mantenere nel tempo uno stato su più richieste provenienti dallo stesso browser
- Per usare la Session Tracking è necessario:
  - Ottenere una sessione (HttpSession Class) per l'utente
  - Memorizzare e richiamare informazioni dalla Sessione
  - Invalidare la sessione (opzionale)

Servlet: Ottenere una sessione

- E' necessario chiamare il metodo getSession dell'oggeto Request
- L'accesso alla sessione va fatto prima di inviare qualsiasi risposta

```
public class CatalogServlet extends HttpServlet {
  public void doGet (HttpServletRequest request,
                     HttpServletResponse response)
          throws ServletException, IOException {

    // Get the user's session and shopping cart
    HttpSession session = request.getSession(true);
    ...
    out = response.getWriter();
    ...
  }
}
```

Servlet: Utilizzare la sessione

- E' possibile memorizzare e ottenere:
  - Proprietà standard della sessione (Session identifier)
  - Coppie nome/valore (nome: stringa; valore: oggetto)

```
public class CatalogServlet extends HttpServlet {
  public void doGet (HttpServletRequest request,
                     HttpServletResponse response)
          throws ServletException, IOException {
    // Get the user's session and shopping cart
    HttpSession session = request.getSession(true);
    ShoppingCart cart=
       (ShoppingCart)session.getAttribute("examples.bookstore.cart");

    // If the user has no cart, create a new one
    if (cart == null) {
      cart = new ShoppingCart();
      session.putAttribute("examples.bookstore.cart", cart);
    }
    ...
  }
}
```


Servlet: Invalidare la sessione

- E' possibile invalidare una servlet
  - Manualmente
  - In modo automatico, dal server, dopo un determinato timeout

```
public class ReceiptServlet extends HttpServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse
response)
          throws ServletException, IOException {
    ...
    scart = (ShoppingCart)session.getAttribute("examples.bookstore.cart");
    ...
    // Clear out shopping cart by invalidating the session
    session.invalidate();
    // set content type header before accessing the Writer
    response.setContentType("text/html");
    out = response.getWriter();
    ...
  }
}
```

Servlet: Cookies

- Le servlet inviano i cookie al client attraverso l'HTTP response header
- Possono essere utlizzati fino a 20 cookie di al massimo 4Kb ciascuno
- Per inviare un cookie occorre:
  - Istanziare un oggetto cookie
  - Settare tutti gli attributi dell'oggetto cookie
  - Inviare il cookie
- Per recuperare informazioni da un cookie occorre:
  - Recuperare dall'HTTP request tutti i cookie dell'utente
  - Trovare il cookie che interessa attraverso il suo nome
  - Recuperare il valore del cookie selezionato

Servlet: Creare un cookie

- Un cookie viene creato attraverso il costruttore della classe *Cookie*
- Per settarne il valore (stringa) si può utilizzare il metodo *setValue*

```
public void doGet (HttpServletRequest request,
                   HttpServletResponse response)
        throws ServletException, IOException {
  // Check for pending adds to the shopping cart
  String bookId = request.getParameter("Buy");
  //If the user wants to add a book, remember it by adding a cookie
  if (bookId != null) {
    Cookie getBook = new Cookie("Buy", bookId);
    ...
  }
  // set content-type header before accessing the Writer
  response.setContentType("text/html");
  // now get the writer and write the data of the response
  PrintWriter out = response.getWriter();
  out.println("<html>" + "<head><title> Book Catalog </title></head>" + ...);
  ...
}
```

Servlet: Inviare un Cookie

- Un cookie viene inviato col metodo addCookie di HttpServletResponse

```
public void doGet (HttpServletRequest request,
                   HttpServletResponse response)
        throws ServletException, IOException {
  ...
  //If the user wants to add a book, remember it by adding a cookie
  if (values != null) {
    bookId = values[0];
    Cookie getBook = new Cookie("Buy", bookId);
    getBook.setComment("User has indicated a desire " +
                       "to buy this book from the bookstore.");
    response.addCookie(getBook);
  }
  ...
}
```


Servlet: Ottenere il valore di un Cookie

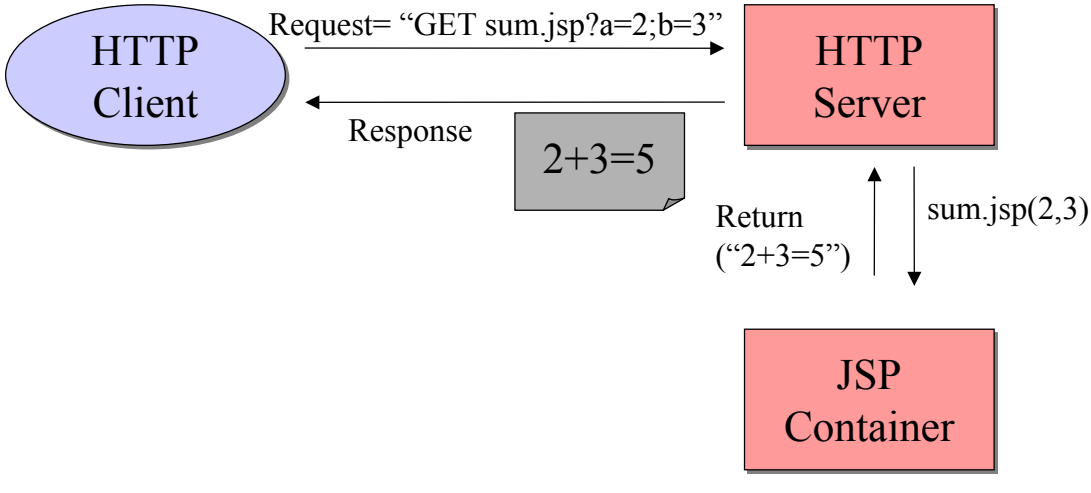
- Per ottenere il valore di un cookie è necessario recuperarli tutti

```
public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
  ...
  /* Handle any pending deletes from the shopping cart */
  String bookId = request.getParameter("Remove");
  ...
  if (bookId != null) {
    // Find the cookie that pertains to that book
    Cookie[] cookies = request.getCookies();
    for(i=0; i < cookies.length; i++) {
      Cookie thisCookie = cookie[i];
      if (thisCookie.getName().equals("Buy") &&
          thisCookie.getValue().equals(bookId)) {
        // Delete the cookie by setting its maximum age to zero
        thisCookie.setMaxAge(0);
      }
    }
  }
  // also set content type header before accessing the Writer
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  //Print out the response
  out.println("<html> <head>" + "<title>Your Shopping Cart</title>" + ...);
}
```

Java Server Pages

- Tecnologia Java-Based
- Semplifica lo sviluppo
- E' assimilabile a un linguaggio di scripting server-side
- Le Java Server Pages contengono sia HTML che codice:
  - Il client effettua una richiesta per una pagina JSP
  - La parte HTML viene passata senza trasformazione
  - Il codice viene eseguito sul server e viene generato il contenuto dinamico
  - La pagina così creata viene restituita al client
- L'esecuzione delle JSP è compito del *JSP Container*

Java Server Pages: Architettura



http://myserver/sum.jsp?a=2;b=3

Java Server Pages: Hello World!

```
<html>
  <body>
    Hello, World!
  </body>
</html>
```


Java Server Pages: Hello World!

```
<html>
  <body>
  <% String visitor=request.getParameter("name");
     if (visitor == null) visitor = "World";
  %>
     Hello, <%= visitor %>!
  </body>
</html>
```

http:// server/webdev/fundamentals/helloScript.jsp

Hello, World!

http://server/webdev/fundamentals/helloScript.jsp?name=Flynn

Hello, Flynn!

Java Server Pages: Hello World!

```
package com.taglib.wdjsp.fundamentals;

public class HelloBean implements java.io.Serializable {

  String name;

  public HelloBean () {
    this.name = "World";
  }
  public String getName () {
    return name;
  }
  public void setName (String name) {
    this.name = name;
  }

}
```

```
<html>
  <body>

    <jsp:useBean id="hello" class="com.taglib.wdjsp.fundamentals.HelloBean"/>
    <jsp:setProperty name="hello" property="name" param="name"/>

    Hello, <jsp:getProperty name="hello" property="name"/>!

  </body>
</html>
```

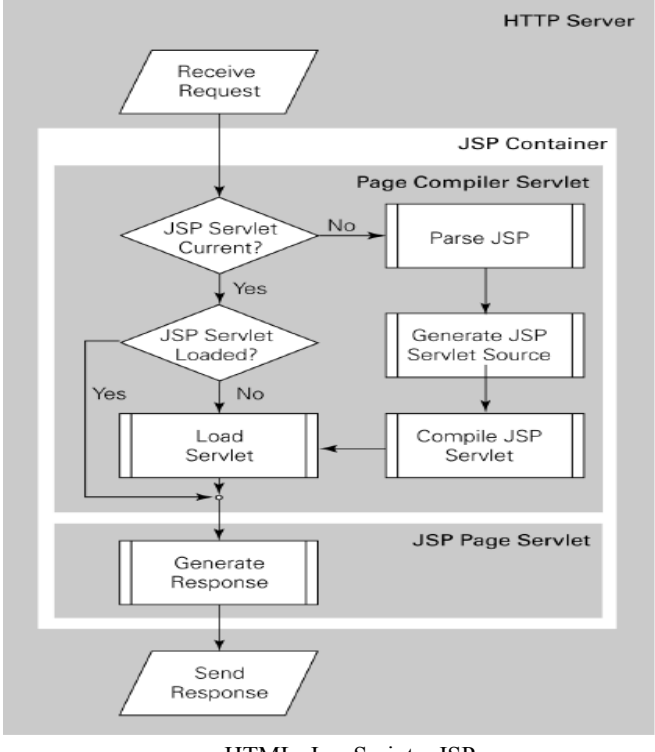

Java Server Pages: Scripting-Oriented Tag

- <% *Script* %>
- Sono disponibili altri delimitatori di apertura: <%!, <%=, <%@
- Esempi:
  - <%! double radius = 7.5; %>
  - <%= 2 * Math.PI * radius %>
  - <% if (radius > 10.0) {
    out.println("Exceeds recommended maximum. Stress analysis advised.");
    } %>
  - <%@ include file="copyright.html" %>

Java Server Pages: XML-Based Tag

```
<jsp:forward page="admin.jsp"/>

<jsp:useBean id="login“
             class="com.taglib.wdjsp.fundamentals.UserBean">
   <jsp:setProperty name="login" property="group" value="admin"/>
</jsp:useBean>
```


Java Server Pages: JSP Container



Java Server Pages: Output Buffering

- Limiti del protocollo HTTP:
  - Il Client si aspetta di ricevere tutto il *response header* prima del *response body: l*a JSP deve effettuare tutte le modifice all'header (es: modifica di un cookie) prima di cominciare a creare il body
  - Una volta che il web server comincia a servire la risposta non può più interrompere il processo, altrimenti il browser mostra solo la frazione parziale che ha ricevuto: se ho iniziato a eseguire la JSP non posso più effettuare un forward ad un'altra JSP (se per esempio voglio gestire il verificarsi di eventuali errori)
- Buffering dell'output:
  - Quando si processa una JSP l'output prodotto non viene immediatamente restituito dal container, ma viene bufferato fino a quando la JSP non viene completamente processata e la response è completa sia nella parte header che nella parte body (è possibile effettuare redirect in qualsiasi momento)
  - Tutto questo con il limite della dimensione del buffer

Java Server Pages: Sessione

- Possibilità di gestire efficientemente i cookie (come per le servlet)
- Esiste un oggetto *session* implicito che può essere utilizzato per la sessione:
  - utilizza lo stesso sistema delle servlet
  - sessione server side
  - la sessione scade dopo un determinato time-out

Java Server Pages: Direttive

- Definiscono informazioni particolari per la pagina:
  - Definiscono il linguaggio di scripting utilizzato
  - Includono il contenuto di altre pagine
  - Definiscono i parametri di utilizzo di una Tag Library
  - etc.
- Non producono nessun output visibile
- Generano side effects che cambiano il modo in cui il JSP container processa la pagina

Java Server Pages: Page Directive

```
<%@ page attribute1="value1" attribute2="value2" attribute3=… %>
```

```
<%@ page info="This is a valid set of page directives." %>
<%@ page language="java" import="java.net.*" %>
<%@ page import="java.util.List, java.util.ArrayList" %>
```

```
<%@ page info="This is an invalid page directive" session="false“
          buffer="16k" autoFlush="false" session="false" %>
```

```
<%@ page info="This is not a valid set of page directives." %>
<%@ page extends="com.taglib.wdjsp.MyJspPage“
          info="Use my superclass." %>
```

Java Server Pages: Page Directive

Attribute	Value	Default	Examples
info	Text string	None	info="Registration form."
language	Scripting language name	"java"	language="java"
contentType	MIME type, character set	See first example	contentType="text/html; charset=ISO-8859-1" contentType="text/xml"
extends	Class name	None	extends="com.taglib.wdjsp.MyJspPage"
import	Class and/or package names	None	import="java.net.URL" import="java.util.*, java.text.*"
session	Boolean flag	"true"	session="true"
buffer	Buffer size, or false	"8kb"	buffer="12kb" buffer="false"
autoFlush	Boolean flag	"true"	autoFlush="false"
isThreadSafe	Boolean flag	"true"	isThreadSafe="true"
errorPage	Local URL	None	errorPage="results/failed.jsp"
isErrorPage	Boolean flag	"false"	isErrorPage="false"

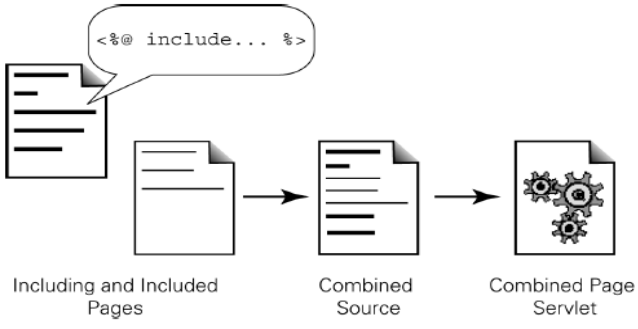
Java Server Pages: Include Directive

<%@ include file="localURL" %>

- Serve ad includere il contenuto del file specificato
- E' possibile nidificare un numero qualsiasi di inclusioni
- Esempi:

<%@ include file="includes/navigation.jsp" %>

<%@ include file="/shared/epilogue/copyright.html" %>



Java Server Pages: Taglib Directive

<%@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>

Esempi:

<%@ taglib uri="/EncomTags" prefix="mcp" %>

<mcp:endProgram/>

Java Server Pages: Script Tag

- Consentono di inserire codice in una pagina JSP
- Ci sono tre tipi di Script Tag:
  - Dichiarazioni
    Definizione di variabili e mteodi per una pagina.
  - Espressioni
    Singole espressioni che generano un valore che viene visualizzato.
  - Scriptlet
    Blocchi di codice che vengono eseguiti ogni volta che la pagina viene processata.
- Nessun tipo di Script Tag ha attributi

Java Server Pages: Dichiarazioni

<%! declaration(s) %>

- Serve a definire variabili e metodi per una JSP
- E' possibile effettuare dichiarazioni multiple
- Le variabili definite diventano **variabili di istanza**

```
<%! private int x = 0, y = 0; private String units = "ft"; %>
```

- E' possibile dichiarare anche uno o più metodi

```
<%! public long fact (long x) {
      if (x == 0) return 1;
      else return x * fact(x-1);
    } %>
```


Java Server Pages: Espressioni

<%= expression %>

- Scopo principale delle espressioni e quello di generare output
- Il risultato dell'espressione viene valutato e visualizzato
- Esempi:

```
<%= Math.PI %>
<%= Math.PI * Math.pow(radius, 2) %>
<%= fact(12) %>
```

- Il risultato dell'espressione può essere di qualsiasi tipo che viene poi convertito in stringa per la visualizzazione
- Le espressioni non sono terminate da un ; perché non sono *statement*
- E molto utilizzato con le espressioni l'operatore ternario condizionale:

test_expr ? *true_expr* : *false_expr*

Java Server Pages: Scriptlets

<% scriptlet %>

- Possono contenere qualsiasi statement del linguaggio di scripting
- E' possibile lasciare aperto un blocco di istruzioni utilizzando **{**
- Gli scriptlet di una pagina vengono eseguiti ad ogni chiamata
- Esempio:

```
<%   GameGrid grid = GameGrid.getGameGrid();
     Recognizer r1 = new Recognizer(new Coordinates(grid, 0, 0));
     Recognizer r2 = new Recognizer(new Coordinates(grid, 100, 100));
     r1.findProgram("Flynn");
     r2.findProgram("Flynn"); %>
 <%= r1.statusReport() %>
```

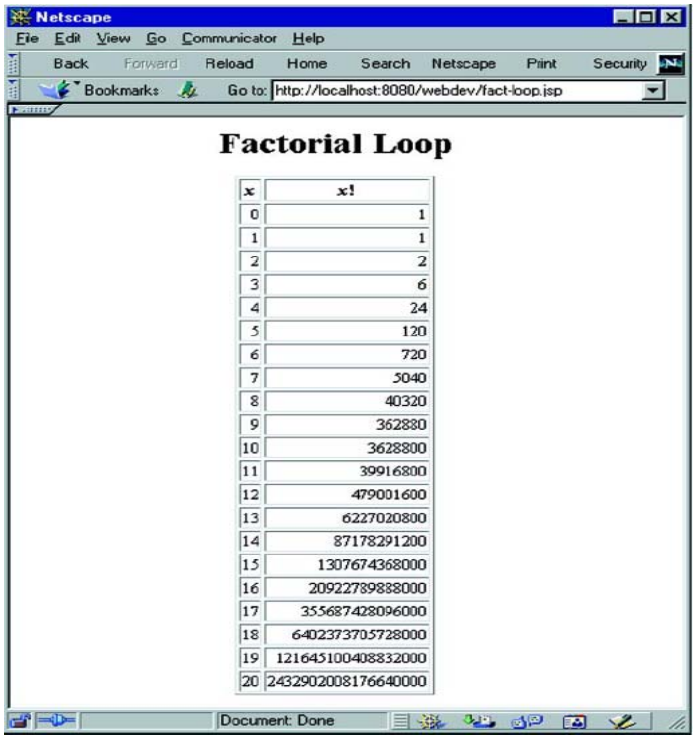

Java Server Pages: Condizioni

```
<% if (x < 0) { %>
  <p>
    Sorry, can't compute the factorial of a negative number.
  </p>
<% } else if (x > 20) { %>
  <p>
    Sorry, arguments greater than 20 cause an overflow error.
  </p>
<% } else { %>
  <p align=center><%= x %>! = <%= fact(x) %></p>
<% } %>
```

Java Server Pages: Iterazioni

```
<table>
  <tr>
    <th><i>x</i></th><th><i>x</i>! </th>
  </tr>
  <% for (long x = 0l; x <= 20l; ++x) { %>
    <tr>
      <td><%= x %></td><td><%= fact(x) %></td>
    </tr>
  <% } %>
</table>
```


Java Server Pages: Iterazioni

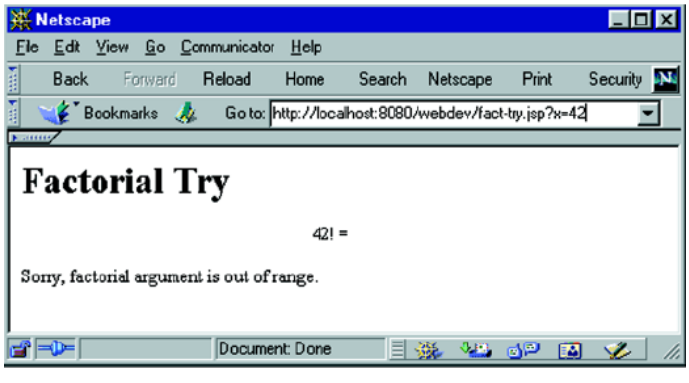


Java Server Pages: Gestione delle Eccezioni

```
<%! public long fact (long x) throws IllegalArgumentException {
        if ((x < 0) || (x > 20))
          throw new IllegalArgumentException("Out of range.");
        else if (x == 0) return 1;
        else return x * fact(x-1);
      } %>
```

```
<% try { %>
  <p align=center> <%= x %>! = <%= fact(x) %></p>
<% } catch (IllegalArgumentException e) { %>
  <p>Sorry, factorial argument is out of range.</p>
<% } %>
```


Java Server Pages: Gestione delle Eccezioni




```
<% try {
       long result = fact(x); %>
  <p align=center> <%= x %>! = <%= result %></p>
<% } catch (IllegalArgumentException e) { %>
  <p>Sorry, factorial argument is out of range.</p>
<% } %>
```

Java Server Pages: Commenti ai contenuti

<!-- comment -->

- Commenti in HTML (o XML)
- Vengono inseriti nell'output esattamente senza essere processati
- Possono anche essere generati dinamicamente
- Esempio:

```
<!-- Java longs are 64 bits, so 20! = <%= fact(20) %> is the upper limit. -->
```


Java Server Pages: Commenti JSP

<%-- comment --%>

- Sono indipendenti dal tipo di contenuto generato
- Sono indipendenti dal linguaggio di scripting utilizzato
- Non vengono trasferiti sull'output
- Il JSP container ignora il contenuto quando processa la pagina
- Esempio:

```
  5! = <%= fact(5) %><br>
<%--
 6! = <%= fact(6) %><br>
 7! = <%= fact(7) %><br>
 8! = <%= fact(8) %><br>
--%>
 9! = <%= fact(9) %><br>
```

Java Server Pages: Commenti del linguaggio

<% /* comment */ %>

- Viene utilizzata la sintassi nativa del linguaggio di scripting
- Non appaiono nell'output della pagina
- Appaiono nel codice sorgente della servlet generata dalla JSP
- Esempi:

```
<% long valid = fact(20);
     long overflow = fact(21); /* Exceeds 64-bit long! */
 %>

<%= /* Comment before expression */ fact(5) %>
<%= fact(7) /* Comment after expression */ %>
<%= fact(9 /* Comment inside expression */) %>

<% long valid = fact(20);// This one fits in a 64-bit long.
     long overflow = fact(21);// This one doesn't.
%>

Lora's brother is over <%= fact(3) // Strange ruler... %> feet tall!
```


Java Server Pages: Implicit Objects

- Oggetti automaticamente disponibili nelle pagine JSP
- Gestiti e forniti dal JSP Container
- Ci sono quattro tipi di oggetti impliciti:
  - oggetti legati alla servlet relativa alla pagina JSP
  - oggetti legati all'input e all'output della pagina JSP
  - oggetti che forniscono informazioni sul contesto in cui la JSP viene eseguita
  - oggetti risultanti da eventuali errori

Java Server Pages: Implicit Objects

Object	Class or *Interface*	Description
page	*javax.servlet.jsp.HttpJspPage*	Page's servlet instance.
config	*javax.servlet.ServletConfig*	Servlet configuration data.
request	*javax.servlet.http.HttpServletRequest*	Request data, including parameters.
response	*javax.servlet.http.HttpServletResponse*	Response data.
out	javax.servlet.jsp.JspWriter	Output stream for page content.
session	*javax.servlet.http.HttpSession*	User-specific session data.
application	*javax.servlet.ServletContext*	Data shared by all application pages.
pageContext	javax.servlet.jsp.PageContext	Context data for page execution.
exception	java.lang.Throwable	Uncaught error or exception.

JSP implicit objects and their API's for HTTP applications

Java Server Pages: page Object

- Oggetto legato alla servlet relativa alla pagina JSP
- L'oggetto page rappresenta l'istanza della servlet
- Poco usato nella pratica
- Esempio:

```
<%@ page info="Page implicit object demonstration." %>
  Page info:
<%= ((javax.servlet.jsp.HttpJspPage)page).getServletInfo() %>
```

Java Server Pages: config Object

- Oggetto legato alla servlet relativa alla pagina JSP
- L'oggetto contiene la configurazione della servlet (parametri di inizializzazione)
- Poco usato in pratica (poco usati i parametri di inizializzazione)

Method	Description
`getInitParameterNames()`	Retrieves the names of all initialization parameters.
`getInitParameter(name)`	Retrieves the value of the named initialization parameter.

Java Server Pages: request Object

- Oggetto legato all'I/O della pagina JSP
- L'oggetto request rappresenta la richiesta per la pagina JSP
- Consente l'accesso a tutte le informazioni della request:
  - l'indirizzo di provenienza
  - l'URL richiesto
  - tutti gli headers
  - i cookies
  - i parametri associati alla richiesta
- Esempio:

```
<%    String xStr = request.getParameter("num");
      try {
        long x = Long.parseLong(xStr); %>
        Factorial result: <%= x %>! = <%= fact(x) %>
<%    } catch (NumberFormatException e) { %>
        Sorry, the <b>num</b> parameter does not specify an integer value.
<%    } %>
```

Java Server Pages: request Object

Method	Description
`getParameterNames()`	Returns the names of all request parameters
`getParameter(name)`	Returns the first (or primary) value of a single request parameter
`getParameterValues(name)`	Retrieves all of the values for a single request parameter.

Method	Description
`getHeaderNames()`	Retrieves the names of all of headers associated with the request.
`getHeader(name)`	Returns the value of a single request header, as a string.
`getHeaders(name)`	Returns all of the values for a single request header.
`getIntHeader(name)`	Returns the value of a single request header, as an integer.
`getDateHeader(name)`	Returns the value of a single request header, as a date.
`getCookies()`	Retrieves all of the cookies associated with the request.

Java Server Pages: request Object

Method	Description
`getMethod()`	Returns the HTTP (e.g., `GET`, `POST`) method for the request.
`getRequestURI()`	Returns the request URL, up to but not including any query string.
`getQueryString()`	Returns the query string that follows the request URL, if any.
`getSession(flag)`	Retrieves the session data for the request (i.e., the `session` implicit object), optionally creating it if it doesn't already exist.
`getRequestDispatcher(path)`	Creates a request dispatcher for the indicated local URL.
`getRemoteHost()`	Returns the fully qualified name of the host that sent the request.
`getRemoteAddr()`	Returns the network address of the host that sent the request.
`getRemoteUser()`	Returns the name of user that sent the request, if known.
`getSession(flag)`	Retrieves the session data for the request (i.e., the `session` implicit object), optionally creating it if it doesn't already exist.

Java Server Pages: request Object

```
<%  String nome=new String();
    String _HERE= request.getRequestURI();
    java.util.Enumeration eNames=request.getParameterNames();

    util.Debug.println("--"+_HERE+" (BEGIN)----------------------");
    while (eNames.hasMoreElements()) {
      nome=(String)eNames.nextElement();
      String[] values=request.getParameterValues(nome);
      for (int pc=0;pc<values.length;pc++)
        util.Debug.println(nome+": ["+pc+"]:"+values[pc]+" ");
      }
    util.Debug.println("--"+_HERE+" (END)------------------------"); %>
```


Java Server Pages: response Object

- Oggetto legato all'I/O della pagina JSP
- Rappresenta la risposta che viene restituita al client
- Consente di inserire nella risposta diverse informazioni:
  - il content type e l'enconding
  - eventuali header di risposta
  - URL Rewriting
  - i cookies
- Esempio:

```
<%response.setDateHeader("Expires", 0);
    response.setHeader("Pragma", "no-cache");
    if (request.getProtocol().equals("HTTP/1.1")) {
      response.setHeader("Cache-Control", "no-cache");
    }%>
```

Java Server Pages: response Object

Method	Description
`setContentType()`	Set the MIME type and, optionally, the character encoding of the response's contents.
`getCharacterEncoding()`	Returns the character encoding style set for the response's contents.

Method	Description
`addCookie(cookie)`	Adds the specified cookie to the response.
`containsHeader(name)`	Checks whether the response includes the named header.
`setHeader(name, value)`	Assigns the specified string value to the named header.
`setIntHeader(name, value)`	Assigns the specified integer value to the named header.
`setDateHeader(name, date)`	Assigns the specified date value to the named header.
`addHeader(name, value)`	Adds the specified string value as a value for the named header.
`addIntHeader(name, value)`	Adds the specified integer value as a value for the named header.
`addDateHeader(name, date)`	Adds the specified date value as a value for the named header.

Java Server Pages: response Object

Method	Description
setStatus(code)	Sets the status code for the response (for non-error circumstances).
sendError(status, msg)	Sets the status code and error message for the response.
sendRedirect(url)	Sends a response to the browser indicating it should request an alternate (absolute) URL.

Method	Description
encodeRedirectURL(url)	Encodes a URL for use with the `sendRedirect()` method to include session information.
encodeURL(name)	Encodes a URL used in a link to include session information.

Java Server Pages: out Object

- Oggetto legato all'I/O della pagina JSP
- Rappresenta lo stream di output della pagina
- Esempio:

```
<P>Counting eggs
<% int count = 0;
      while (carton.hasNext()) {
        count++;
        out.print(".");
      } %>
<BR>
There are <%= count %> eggs.</P>
```


Java Server Pages: out Object

Method	Description
`isAutoFlush()`	Returns `true` if the output buffer is automatically flushed when it becomes full, `false` if an exception is thrown.
`getBufferSize()`	Returns the size (in bytes) of the output buffer.
`getRemaining()`	Returns the size (in bytes) of the unused portion of the output buffer.
`clearBuffer()`	Clears the contents of the output buffer, discarding them.
`clear()`	Clears the contents of the output buffer, signaling an error if the buffer has previously been flushed.
`newLine()`	Writes a (platform-specific) line separator to the output buffer.
`flush()`	Flushes the output buffer, then flushes the output stream.
`close()`	Closes the output stream, flushing any contents.

Java Server Pages: session Object

- Oggetto che fornisce informazioni sul contesto di esecuzione della JSP
- Rappresenta la sessione corrente per un utente
- La direttiva *session* deve essere *true*
- Esempio:

```
<% UserLogin userData = new UserLogin(name, password);
   session.setAttribute("login", userData); %>

<% UserLogin userData = (UserLogin) session.getAttribute("login");
   if (userData.isGroupMember("admin")) {
     session.setMaxInactiveInterval(60*60*8);
   } else {
     session.setMaxInactiveInterval(60*15);
   } %>
```


Java Server Pages: session Object

Method	Description
`getId()`	Returns the session ID.
`getCreationTime()`	Returns the time at which the session was created.
`getLastAccessedTime()`	Returns the last time a request associated with the session was received.
`getMaxInactiveInterval()`	Returns the maximum time (in seconds) between requests for which the session will be maintained.
`setMaxInactiveInterval(t)`	Sets the maximum time (in seconds) between requests for which the session will be maintained.
`isNew()`	Returns `true` if user's browser has not yet confirmed the session ID.
`invalidate()`	Discards the session, releasing any objects stored as attributes.

Java Server Pages: application Object

- Oggetto che fornisce informazioni sul contesto di esecuzione della JSP
- Rappresenta l'applicazione a cui la JSP appartiene
- Consente di interagire con l'ambiente di esecuzione:
  - versione del JSP Container
  - accesso a risorse server-side
  - supporto al log
  - accesso ai parametri di inizializzazione relativi all'applicazione
  - possibilità di gestire gli attributi di una applicazione

Java Server Pages: pageContext Object

- 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
- Consente il trasferimento del controllo ad altre pagine
- Poco usato per lo scripting, utile per costruire custom tags

Java Server Pages: exception Object

- 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 (solo nelle Error Page)
- Esempio:

```
<%@ page isErrorPage="true" %>
<H1>Warning!</H1>
The following error has been detected:<BR>
<B><%= exception %></B><BR>
<% exception.printStackTrace(out); %>
```


Java Server Pages: exception Object

Method	Description
`getMessage()`	Returns the descriptive error message associated with the exception when it was thrown.
`printStackTrace(out)`	Prints the execution stack in effect when the exception was thrown to the designated output stream.
`toString()`	Returns a string combining the class name of the exception with its error message (if any).

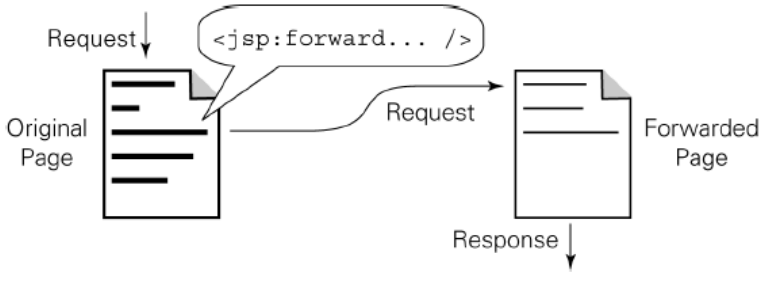
Java Server Pages: Actions

- Tag che consentono di eseguire particolari azioni
- Consentono di trasferire il controllo dalla pagina corrente ad altre pagine
- Consentono di dialogare con i componenti (JavaBeans)
- Viene usata la sintassi XML

Java Server Pages: Forward

<jsp:forward page="*localURL*" />

- Consente il trasferimento del controllo dalla pagina JSP corrente ad un'altra pagina sul server locale
- L'attributo *page* definisce l'URL della pagina a cui trasferire il controllo
- La request viene completamente trasferita in modo trasparente per il client



Java Server Pages: Forward

<jsp:forward page="*localURL*" />

- E' possibile generare dinamicamente l'attributo *page*

  <jsp:forward page='<%= "message" + statusCode + ".html" %>' />

- Gli oggetti *session* e *request* della pagina d'arrivo sono gli stessi della pagina chiamante, ma viene istanziato un nuovo contesto
- E' possibile aggiungere parametri all'oggetto request della pagina chiamata utilizzando il tag <jsp:param>

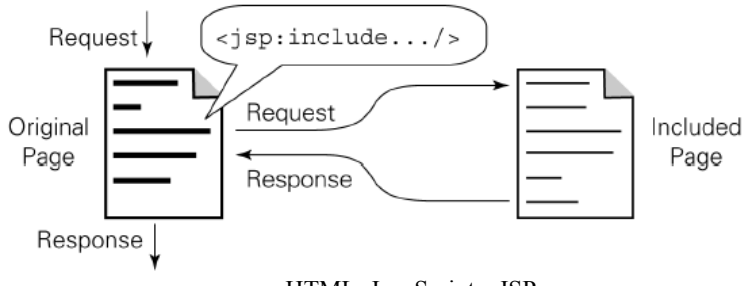
```
<jsp:forward page="localURL">
<jsp:param name="parameterName1" value="parameterValue1"/>
   …
<jsp:param name="parameterNameN" value="parameterValueN"/>
</jsp:forward>
```

- La redirezione è possibile soltanto se non è già stato fatto un flush del buffer di output o se la chiamata avviene pirma della creazione di qualsiasi output

Java Server Pages: Include

<jsp:include page="*localURL*" flush="true" />

- Consente di includere il contenuto generato dinamicamente da un'altra pagina locale all'interno dell'output della pagina corrente
- Questo tag trasferisce **temporaneamente** il controllo ad un'altra pagina
- L'attributo *page* definisce l'URL della pagina da includere
- L'attributo *flush* stabilisce se sul buffer della pagina corrente viene eseguito un flush prima di effettuare l'inclusione (deve essere *true*)
- Gli oggetti *session* e *request* per la pagina da includere sono gli stessi della pagina chiamante, ma viene istanziato un nuovo contesto



Java Server Pages: Include

<jsp:include page="*localURL*" flush="true" />

•E' possibile aggiungere parametri all'oggetto request della pagina che viene inclusa utilizzando il tag <jsp:param>

```
<jsp:include page="localURL“ flush=“true”>
  <jsp:param name="parameterName1“
             value="parameterValue1"/>
  …
  <jsp:param name="parameterNameN“
             value="parameterValueN"/>
</jsp: include >
```


Java Server Pages: Modello a componenti

- Scriptlet e espressioni consentono uno sviluppo centrato sulla pagina
- Lo sviluppo centrato sulla pagina non consente una forte separazione tra sviluppo dell'applicazione e presentazione dei contenuti
- Un applicazione molto complessa necessità di un Architettura a più livelli
- Le JSP consentono uno sviluppo basato sul modello a componenti
- Il Modello a Componenti consente di avere una maggiore separazione fra logica dell'applicazione e contenuti
- Il Modello a Componenti consente architetture molto più articolate

Java Server Pages: JavaBeans

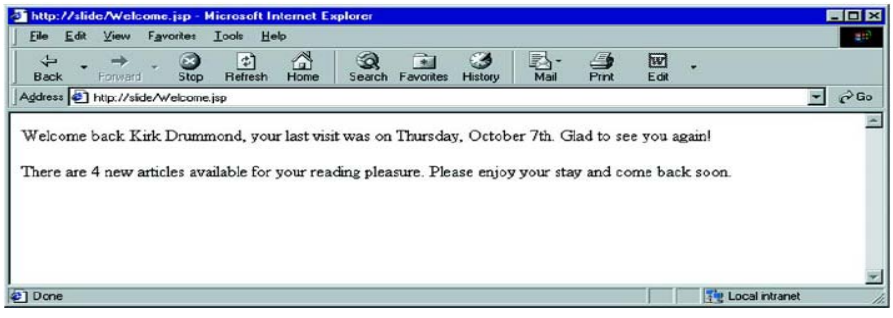
- I Componenti solo elementi software autonomi, riusabili che incapsulano un certo insieme di funzionalità
- I Componenti dialogano fra loro e con l'esterno attraverso una interfaccia ben definita
- I **JavaBeans** sono componenti software scritti in Java
- I JavaBeans hanno delle **proprietà** che ne definiscono lo stato:
  - possono essere *read-only*, *write-only* o *readable and writable*
  - possono essere *trigger* o *linked*
  - possono essere *indexed*
  - possono contenere qualsiasi tipo di dato Java

Java Server Pages: Bean Tags

- Esistono dei tag per agganciare un bean e le sue proprietà
- Ci sono tre tipi di tag:
  - Tag per creare un riferimento al bean
  - Tag per settare il valore delle proprietà del bean
  - Tag per leggere il valore delle proprietà del bean

Java Server Pages: Bean Tags

```
<jsp:useBean id="user" class="RegisteredUser" scope="session"/>
<jsp:useBean id="news" class="NewsReports" scope="request">
  <jsp:setProperty name="news" property="category" value="financial"/>
  <jsp:setProprety name="news" property="maxItems" value="5"/>
</jsp:useBean>

<html>
  <body>
    Welcome back
    <jsp:getProperty name="user" property="fullName"/>,
    your last visit was on
    <jsp:getProperty name="user" property="lastVisitDate"/>.
    Glad to see you again!
    <P>
      There are <jsp:getProperty name="news" property="newItems"/>
      new articles available for your reading pleasure. Please enjoy your
      stay and come back soon.
  </body>
</html>
```



Java Server Pages: <jsp:useBean>

<jsp:useBean id="beanName" class="class name"/>

- Inizializza e crea la reference al bean
- E possibile specificare il tipo di bean ed assegnarli un nome
- Gli attributi principali sono *id* e *class* ma è possibile specificarne altri

Attribute	Value	Default	Example Value
id	Java identifier	none	myBean
scope	page, request, session, application	page	session
class	Java class name	none	java.util.Date
type	Java class name	same as class	com.manning.jsp.AbstractPerson
beanName	Java class or serialized Bean	none	com.manning.jsp.USCurrency.ser

Java Server Pages: <jsp:useBean>

- Attributo *id*
  - identificatore unico per il bean (Java identifier)
- Attributo *class*
  - indica il nome della classe del bean
  - la classe deve essere nel classpath

```
<% @page import="com.taglib.wdjsp.*" %>
<jsp:useBean name="user" class="RegisteredUserBean" />
        oppure
<jsp:useBean name="user" class="com.taglib.wdjsp.RegisteredUserBean" />
```


Java Server Pages: <jsp:useBean>

```
<jsp:useBean id="myclock" class="com.manning.jsp.ClockBean"/>

<html>
  <body>
    There is a Bean hiding in this page!
  </body>
</html>
```

- Questa JSP userà il bean definito dalla classe **com.manning.jsp.ClockBean**
- Si farà riferimento al bean mediante l'identificatore **myclock**
- Un bean va definito prima di essere usato
- **<jsp:useBean>** crea un'istanza del bean e gli assegna l'id
- Il bean, quando istanziato, esegue determinate operazioni definite dallo sviluppatore
- Ciascun tag crea **<jsp:useBean>** una diversa istanza di un bean

```
<jsp:useBean id="clock1" class="com.manning.jsp.ClockBean" />
<jsp:useBean id="clock2" class="com.manning.jsp.ClockBean" />
```

Java Server Pages: <jsp:getProperty>

```
<jsp:getProperty name="bean name"
                 property="property name"/>
```

- Consente l'accesso alle proprietà del bean
- Produce come output il valore della proprietà del bean
- Il tag non ha mai body e ha solo 2 attributi:
  - *name,* definisce il nome del bean a cui faccio riferimento
  - *property,* definisce il nome della proprietà di cui voglio visualizzare il valore

Java Server Pages: <jsp:getProperty>

```
<jsp:useBean id="style" class="beans.CorporateStyleBean"/>
<html>
 <body bgcolor="<jsp:getProperty name="style" property="color"/>">
  <center>
   <img src="<jsp:getProperty name="style" property="logo"/>">
   Welcome to Big Corp!
  </center>
 </body>
</html>
```

```
<html>
 <body bgcolor="pink">
  <center>
   <img src="http://imageserver/logo.gif">
   Welcome to Big Corp!
  </center>
 </body>
</html>
```

Java Server Pages: <jsp:setProperty>

```
<jsp:setProperty name="bean name“
                 property="property name“
                 value=“property value”/>
```

- Consente di modificare il valore delle proprietà del bean
- Il bean eseguirà determinate operazioni in funzione del valore assegnato alle proprietà
- Esempi:

```
<jsp:setProperty name="user" property="daysLeft" value="30"/>
<jsp:setProperty name="user" property="daysLeft"
                             value="<%= 15 * 2 %>"/>
```


Java Server Pages: Indexed Properties

- I bean tag non supportano le proprietà indicizzate
- **Un bean è un normale oggetto Java**:
  è possibile accedere a variabili e metodi.
- Esempio:

```
<b>Tomorrow’s Forecast</b>: <%= weather.getForecasts(0) %>
<br/>
<b>The Rest of the Week</b>
<ul>
  <% for (int index=1; index < 5; index++) { %>
    <li><%= weather.getForecasts(index) %></li>
  <% } %>
</ul>
```

Java Server Pages: Inizializzare un bean

- E' possibile inizializzare un bean settando il valore delle proprietà al momento dell'inizializzazione:

```
<jsp:useBean id="clock" class="com.manning.jsp.ClockBean">
  <jsp:setProperty name="clock" property="timezone" value="CST"/>
</jsp:useBean>
```

- E' possibile inizializzare un bean con i parametri della request HTTP:

```
<jsp:setProperty name="calculator" property="interestRate“
                 value="<%= request.getParameter(“interestRate") %>"/>
<jsp:setProperty name="calculator" property="years“
                value="<%= request.getParameter(“years") %>"/>

<jsp:setProperty name="calculator" property="interestRate“ param="interestRate"/>
<jsp:setProperty name="calculator" property="years" param="years"/>

<jsp:setProperty name="calculator" property="interestRate"/>
<jsp:setProperty name="calculator" property="years"/>

<jsp:setProperty name="calculator" property="*">
```


Java Server Pages: Bean Life Cycle

- Ogni volta che una pagina JSP viene richiesta e processata il bean viene istanziato
- E' possibile estendere la vita del bean oltre la singola richiesta
- *scope* è l'attributo che consente di estendere la vita del bean
- Il bean viene identificato nel suo ciclo vitale dall'attributo id del tag <jsp:useBean>

Scope	Accessibility	Life span
page	current page only	until page is displayed or control is forwarded to a new page
request	current page and any included or forwarded pages	until the request has been completely processed and the response has been sent back to the user
session	the current request and any subsequent request from the same browser window	life of the user's session
application	the current and any future request that is part of the same web application	life of the application

Java Server Pages: Bean Life Cycle

<jsp:useBean id="beanName" class="class“
scope="page|request|session|application"/>

- Page Beans (default scope)
  - Ogni volta che viene richiesta la pagina un nuovo bean viene istanziato
  - bean transienti e non persistenti
- Request Beans
  - Utilizzato per estendere la vita del bean alle JSP richiamate mediante i tag <jsp:include> e <jsp:forward>

Java Server Pages: Bean Life Cycle

<jsp:useBean id="beanName" class="class“
scope="page|request|session|application"/>

- Session Beans
  - Il bean è persistente, viene inserito nell’oggetto di sessione
  - Nella stessa sessione, qualsiasi altra JSP avrà accesso al bean
  - Il bean scade allo scadere della sessione
- Application Beans
  - Persistenza totale: il bean esiste finché esiste il JSP Container
  - E’ disponibile per tutte le JSP e per ogni utente/sessione
  - Esiste solo un’istanza del bean per il server

Java Server Pages: Cos'è un bean?

- Un JavaBean è una normale classe Java
- Una classe Java, per essere un bean, deve seguire un determinato insieme di regole e convenzioni di naming e di struttura
- E' sempre possibile accedere agli elementi di un bean secondo il normale standard Java
- Un bean container è in grado di interfacciarsi con i bean utilizzando i meccanismi Java standard della *introspection* e della *reflection*

Java Server Pages: Bean Constructor

- E' necessario implementare un costruttore senza argomenti
- Tale costruttore è quello utilizzato dal JSP container per istanziare il bean richiesto con il tag <jsp:useBean>
- Ciascuna classe Java ha un costruttore di default senza argomenti
- Esempi:

```
public class DoNothingBean { }

package com.taglib.wdjsp.components;
import java.util.*;
public class CurrentTimeBean {
  private int hours;
  private int minutes;
  public CurrentTimeBean() {
    Calendar now = Calendar.getInstance();
    this.hours = now.get(Calendar.HOUR_OF_DAY);
    this.minutes = now.get(Calendar.MINUTE);
  }
}
```

Java Server Pages: Bean Properties

- Per creare una proprietà è necessario creare una variabile *private* e i metodi di accesso
- I metodi di accesso (*access methods*) consentono di accedere (*getter method*) e di modificare (*setter method*) il valore di una proprietà
- I metodi di accesso sono metodi pubblici che hanno il nome della proprietà preceduta dal prefisso *get* o *set*
- Deve essere rispettata la tipizzazione
- Esempi:

```
private String rank;
public void setRank(String rank);
public String getRank();

private int age;
public void setAge(int age);
public int getAge();
```


Java Server Pages: Bean Properties

```
package com.taglib.wdjsp.components;
import java.util.*;
public class CurrentTimeBean {
  private int hours;
  private int minutes;
  public CurrentTimeBean() {
    Calendar now = Calendar.getInstance();
    this.hours = now.get(Calendar.HOUR_OF_DAY);
    this.minutes = now.get(Calendar.MINUTE);
  }
  public int getHours() {
    return hours;
  }
  public int getMinutes() {
    return minutes;
  }
}
```

```
<jsp:useBean id="time" class="CurrentTimeBean"/>
<html><body>
It is now <jsp:getProperty name="time" property="minutes"/>
minutes past the hour.
</body></html>
```

Java Server Pages: Bean Properties

```
package com.taglib.wdjsp.components;
import java.util.*;
public class DiceBean {
  private Random rand;
  public DiceBean() {
    rand = new Random();
  }

  public int getDieRoll() {
    // return a number between 1 and 6
    return rand.nextInt(6) + 1;
  }

  public int getDiceRoll() {
    // return a number between 2 and 12
    return getDieRoll() + getDieRoll();
  }

}
```


Java Server Pages: Indexed Properties

```
package com.taglib.wdjsp.components;
import java.util.*;
public class StatBean {
  private double[] numbers;
  public StatBean() {
    numbers = new double[2];
    numbers[0] = 1;
    numbers[1] = 2;
  }
  public double getAverage() {
    double sum = 0;
    for (int i=0; i < numbers.length; i++)
      sum += numbers[i];
    return sum/numbers.length;
  }
  public double[] getNumbers() {
    return numbers;
  }
  public double getNumbers(int index) {
    return numbers[index];
  }
  public void setNumbers(double[] numbers) {
    this.numbers = numbers;
  }
  public void setNumbers(int index, double value) {
    numbers[index] = value;
  }
  public void setNumbersList(String values) {
    Vector n = new Vector();
    StringTokenizer tok =
       new StringTokenizer(values, ",");
    while (tok.hasMoreTokens())
      n.addElement(tok.nextToken());
    numbers = new double[n.size()];
    for (int i=0; i < numbers.length; i++)
      numbers[i] =
        Double.parseDouble((String) n.elementAt(i));
  }
  public String getNumbersList() {
    String list = new String();
    for (int i=0; i < numbers.length; i++) {
      if (i != numbers.length)
        list += numbers[i] + ",";
      else
        list += "" + numbers[i];
    }
    return list;
  }
  public int getNumbersSize() {
    return numbers.length;
  }
}
```

Java Server Pages: Indexed Properties

```
<jsp:useBean id="stat" class="com.taglib.wdjsp.StatBean">
  <% double[] mynums = {100, 250, 150, 50, 450};
        stat.setNumbers(mynums); %>
</jsp:useBean>

<html>
  <body>
    The average of
<% double[] numbers = stat.getNumbers();
       for (int i=0; i < numbers.length; i++) {
         if (i != numbers.length)
           out.print(numbers[i] + ",");
         else
           out.println("" + numbers[i]);
       } %>
    is equal to <jsp:getProperty name="stat" property="average"/>
  </body>
</html>
```


Java Server Pages: Boolean Properties

private boolean enabled;
public boolean isEnabled();
public void setEnabled(boolean b);

private boolean authorized;
public boolean isAuthorized();
public void setAuthorized(boolean b);

Java Server Pages: Type Conversion

- Ogni volta si accede ad una proprietà del bean col tag <jsp:getProperty> questa viene convertita in una stringa
- Ogni volta che viene settata una proprietà col tag <jsp:setProperty> avviene la conversione inversa

Property Type	Conversion to String
boolean	java.lang.Boolean.toString(boolean)
byte	java.lang.Byte.toString(byte)
char	java.lang.Character.toString(char)
double	java.lang.Double.toString(double)
int	java.lang.Integer.toString(int)
float	java.lang.Float.toString(float)
long	java.lang.Long.toString(long)

Property Type	Conversion from String
boolean or Boolean	java.lang.Boolean.valueOf(String)
byte or Byte	java.lang.Byte.valueOf(String)
char or Character	java.lang.Character.valueOf(String)
double or Double	java.lang.Double.valueOf(String)
int or Integer	java.lang.Integer.valueOf(String)
float or Float	java.lang.Float.valueOf(String)
long or Long	java.lang.Long.valueOf(String)

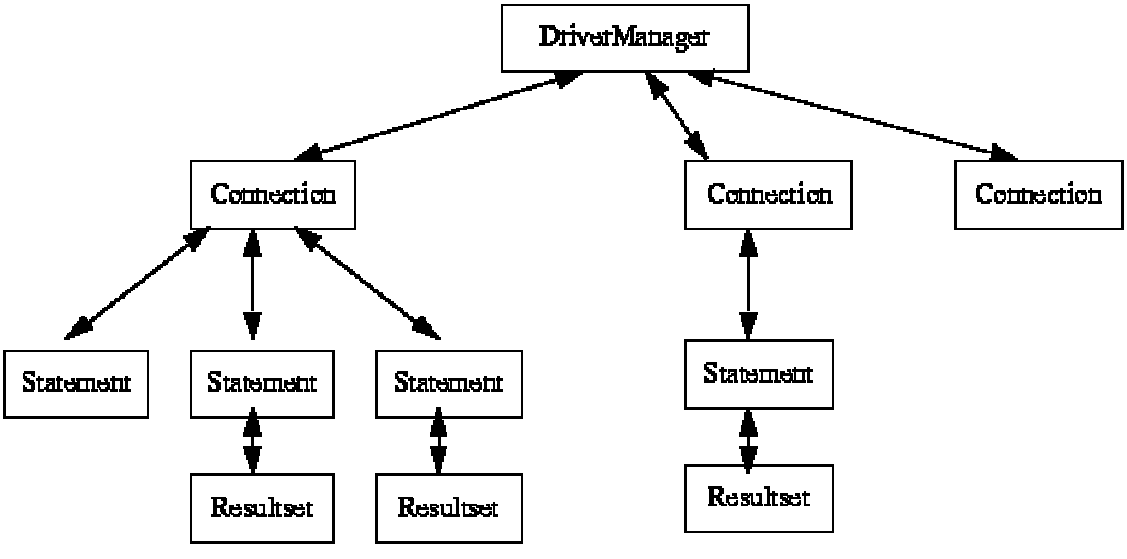
JDBC: Accesso alle Basi di dati

- I Sistemi Informativi hanno il compito di gestire in modo efficiente ed affidabile i dati, mantenuti in una forma strutturata e possibilmente normale.
- La struttura dei dati viene definita attraverso il **modello Entity/Relationship**
- L'accesso e la manipolazione dei dati è standardizzata a livello di linguaggio attraverso lo standard di fatto che è **SQL (Simple Query Language)**
- Ogni base di dati può essere gestita quindi in modo standard, senza conoscere quali sono i dettagli del software e dell'hardware che si occupa della gestione del dato fisico

JDBC: Java Database Connection

- Nasce come strumento standard per gestire le connessioni alle basi di dati DBMS
- È costituito da un insieme di package standard:
  - java.sql
  - javax.sql
- I package standard definiscono una interfaccia al quale deve sottostare il package DataBase dependent detto “driver”

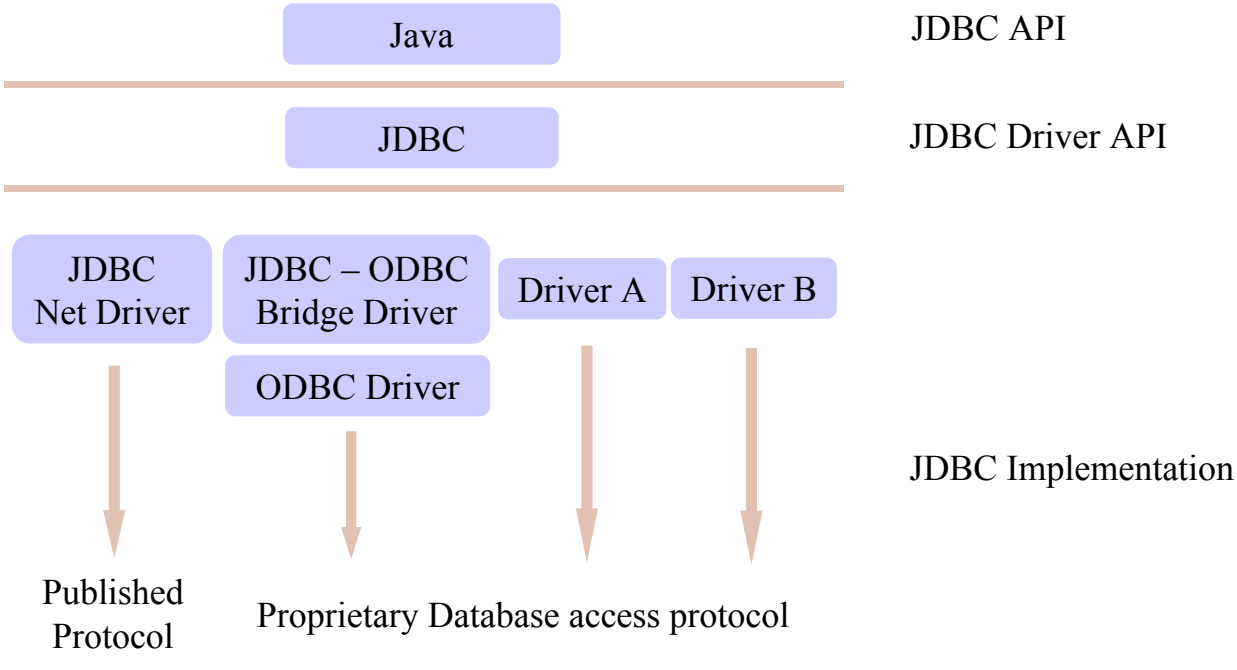
JDBC: Struttura delle API



JDBC: Struttura delle API

- **Driver Manager**: (java.sql.DriverManager) si occupa della gestione dei Driver per accedere ai diversi sistemi informativi, ha il compito di gestire l'apertura e la chiusura delle connessioni verso questi sistemi
- **Connection**: (java.sql.Connection) rappresenta unaconnessione diretta alla base di dati. Mantiene lo stato della interazione tra applicazione e databse.
- **Statement**: (java.sql.Statement) rappresenta l'accesso (via SQL) ad un insieme di dati attraverso una specifica connessione. Lo statement rappresenta quindi una richiesta SQL
- **ResultSet**: (java.sql.ResultSet) contiene il risultato di una specifica richiesta SQL; contiene in generale l'insieme delle righe (record) ottenuti dalla richiesta.

JDBC: Driver Interface



JDBC: Apertura di una connessione

```
try {

    // Create a URL specifying an ODBC data source name.
    String url = "jdbc:odbc:NomeDB";
    String user="scott";
    String pwd="tiger"

    // Connect to the database at that URL.
   Connection con = DriverManager.getConnection(url, user,pwd);

} catch (java.lang.SQLException ex) {
… … … …
}
```


JDBC: Esecuzione di una query SQL

```
try {

    String sSQL="SELECT a, b, c, d, key FROM Table1";

    // Create a Statement
    Statement stmt = con.createStatement();

    // Execute a SELECT statement
    ResultSet rs = stmt.executeQuery(sSQL);

} catch (java.lang.SQLException ex) {
… … … …
}
```

JDBC: Lettura del Resultset

```
try {

    while (rs.next()) {
       // get the values from the current row:
       int a = rs.getInt("a");
       BigDecimal b = rs.getBigDecimal("b");
       char c[] = rs.getString("c").tocharArray();
       boolean d = rs.getBoolean("d");
       String key = rs.getString("key");
       … … … … …
    }
    stmt.close();
    con.close();
} catch (java.lang.SQLException ex) {
… … … …
}
```


JDBC: Lettura del Resultset

SQL → Java

SQL type	Java Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Java → SQL

Java Type	SQL type
String	VARCHAR or LONGVARCHAR
java.math.BigDecimal	NUMERIC
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
byte[]	VARBINARY or LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

JDBC: Lettura del Resultset

		SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
getByte	X	x	x	x	x	x	x	x	x	x	x	x	x						
getShort	x	X	x	x	x	x	x	x	x	x	x	x	x						
getInt	x	x	X	x	x	x	x	x	x	x	x	x	x						
getLong	x	x	x	X	x	x	x	x	x	x	x	x	x						
getFloat	x	x	x	x	X	x	x	x	x	x	x	x	x						
getDouble	x	x	x	x	x	X	X	x	x	x	x	x	x						
getBigDecimal	x	x	x	x	x	x	x	X	X	x	x	x	x						
getBoolean	x	x	x	x	x	x	x	x	x	X	x	x	x						
getString	x	x	x	x	x	x	x	x	x	x	X	X	x	x	x	x	x	x	x
getBytes														X	X	x			
getDate											x	x	x				X		x
getTime											x	x	x					X	x
getTimestamp											x	x	x				x		X
getAsciiStream											x	x	X	x	x	x			
getUnicodeStream											x	x	X	x	x	x			
getBinaryStream														x	x	X			
getObject	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

JDBC: Transazioni

- Associata ad ogni connessione esiste sempre una transazione
- Di default, ogni connessione è configurata in *autocommit*, automaticamente, al completamento di ogni query la tranzazione viene *chiusa con successo* (*commit*) e ne viene riaperta una nuova
- È possibile configurare la connessione in manual *commit*:
  - Connection.setAutoCoomit(false);
- Quando il commit è manuale, per chiudere le transazioni è necessario specificare se la chiusura è con successo o con rollback:
  - Connection.commit();
  - Connection.roolback();

Progetto Rubrica: 07JSP

- Realizzazione di una applicazione web: Rubrica

- 07JSP

  Realizzazione del progetto rubrica in tutte le sue funzionalità mediante una web application dinamica, questa implementazione non è però multilingua.