ROMA TRE UNIVERSITÀ DEGLI STUDI
Corso di Laurea in Matematica
Dipartimento di Matematica e Fisica

Sistemi per l'elaborazione delle informazioni

7. Applicazioni web based

Dispense del corso IN530 a.a. 2019/2020

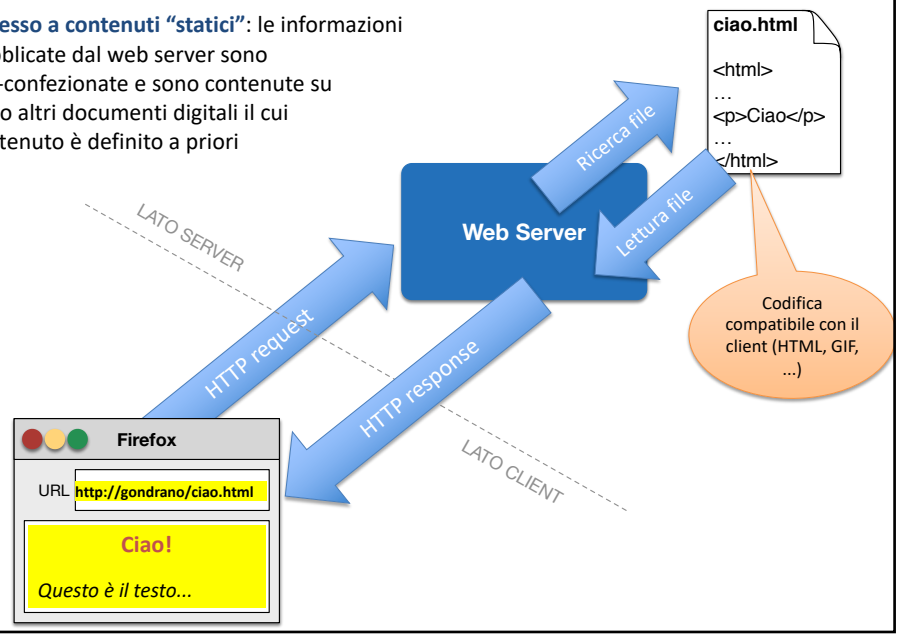
prof. Marco Liverani

Il World Wide Web

- È forse la principale, più importante, più diffusa applicazione informatica dopo i DBMS
- È stato progettato e implementato da **Tim Berners Lee** tra il 1980 e il 1995 quando era un ricercatore presso il CERN
- Nasce per codificare e condividere facilmente documenti, articoli scientifici e materiale multimediale, su una rete TCP/IP
- È basato su tre concetti/componenti chiave:
  - Il **linguaggio HTML** per la codifica dei documenti *ipertestuali*
  - Il **protocollo applicativo HTTP** per la richiesta e l'invio di documenti dai server ai client
  - La codifica delle **URL per identificare univocamente le risorse** (documenti, immagini, ecc.) resi disponibili in rete attraverso il protocollo HTTP
- Un **ipertesto** è un documento che rende possibile una lettura non sequenziale, attraverso percorsi tematici trasversali alle pagine (ai contenuti) del documento stesso
- **HTML** (*HyperText Markup Language*) è un linguaggio di marcatura del testo per definire ipertesti fruibili attraverso il protocollo **HTTP** (*HyperText Transfer Protocol*)
- Alla base di un ipertesto ci sono i collegamenti ipertestuali (*link*) che permettono di collegare singoli termini dell'ipertesto a pagine differenti dello stesso ipertesto o di altri ipertesti
- Il protocollo HTTP definisce il concetto di **URL** (*Uniform Resource Locator*) per costruire i riferimenti ipertestuali alle risorse accessibili mediante lo stesso protocollo HTTP

Navigazione di un documento «ipertestuale»

Accesso a contenuti "statici": le informazioni pubblicate dal web server sono pre-confezionate e sono contenute su file o altri documenti digitali il cui contenuto è definito a priori



Breve cronologia del WWW

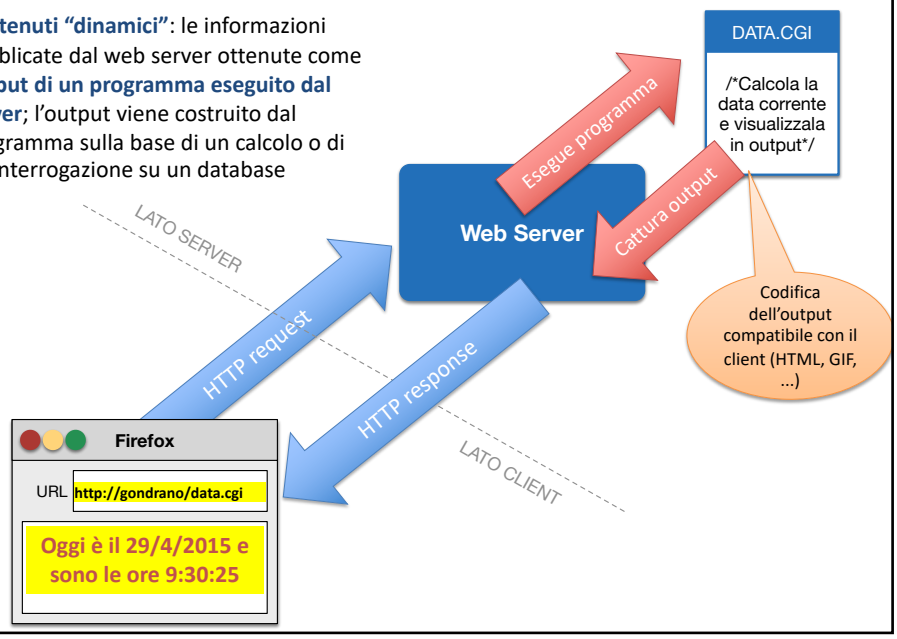
- **1980:** Tim Berners-Lee (CERN) sviluppa il programma *Enquire-Within-Upon-Everything*" che consentiva di effettuare link tra diversi nodi
- **1989:** Tim Berners-Lee diffonde due documenti per accogliere opinioni sul suo lavoro presso il CERN
- **1990:** Il documento di TBL viene riformulato con l'appoggio ufficiale del CERN; viene coniato il nome World Wide Web
- **1991:** Sviluppo dei primi client ed apertura del WWW server del CERN
- **1992:** Sviluppo del client con interfaccia X; viene rilasciata la lista dei primi 26 server WWW
- **1993:** Viene rilasciato il primo browser per Macintosh; viene rilasciato X-Mosaic di Marc Andreessen (NCSA); i server HTTP sono circa 50
- **1993**: Viene implementata l'architettura CGI (formalizzata in RFC 3875 nel 2004)
- **1994:** Viene fondata la Mosaic Corporation (poi Netscape Corp.); i siti WWW sono 1.500; a dicembre si tiene il primo meeting del "W3 Consortium" presso il MIT; il CERN, a causa di problemi di budget, decide di sospendere il supporto del progetto WWW
- **1995:** Viene fondata la "Web Society" a Graz (Austria): partecipano la Technical University di Graz, il CERN, l'Università del Minnesota e l'INRIA (Francia)
- **1996**: Formalizzazione di HTTP/1.0 come RFC 1945 del IETF
- **1997/1999**: Presentazione di HTTP/1.1 (RFC 2616)
- … la storia continua (http://www.internetsociety.org/history)

Applicazioni "web based"

- Un'applicazione web based è un programma eseguito per lo più **lato server**, che offre un'interfaccia utente grafica, codificata in linguaggio HTML e fruita dall'utente mediante un web browser eseguito sul client
- Un'applicazione web based prevede un'interazione tra il programma che gira su un web application server e un web browser che opera su un computer client collegato in rete con il web server
- Il web server e i client devono utilizzare il protocollo TCP/IP; sul protocollo TCP/IP, a livello applicativo, si appoggia il protocollo HTTP
- L'applicazione web implementa un'interazione tra il web client e il web server non molto diversa da quella utilizzata per «navigare» un documento ipertestuale:
  1. Utilizzando il protocollo HTTP il client apre una connessione con il Web Server
  2. Il web client invia al web server la richiesta (HTTP request) di una pagina HTML (contenuto *statico*) o dell'esecuzione di un programma (contenuto *dinamico*) e fornisce in input dei dati al programma, inserendoli nella HTTP request
  3. Il Web Server riceve la richiesta e scorpora i dati in input dal nome del programma da eseguire
  4. Il Web Server esegue il programma e gli fornisce i dati passati in input nella request dell'utente
  5. Il programma esegue la propria elaborazione e produce un output in formato HTML
  6. Il Web Server cattura l'output del programma e lo restituisce al client (HTTP response)
  7. Il Web Server chiude la connessione con il client
  8. Il processo riprende dal passo 1

Applicazioni "web based"

Contenuti "dinamici": le informazioni pubblicate dal web server ottenute come **output di un programma eseguito dal server**; l'output viene costruito dal programma sulla base di un calcolo o di un'interrogazione su un database

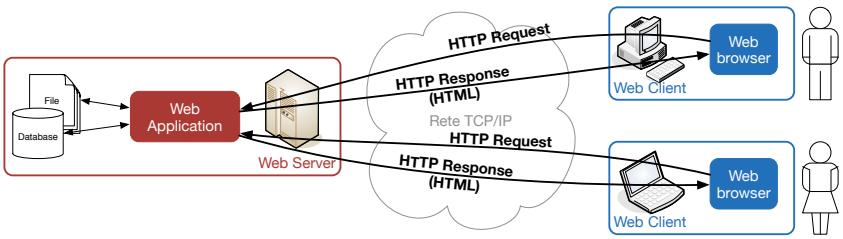


Applicazioni "web based"

- Le applicazioni web based hanno avuto un successo straordinario per numerosissimi motivi, tra i quali:
  - È un'architettura applicativa basata su **standard «aperti»** (noti a tutti e ben documentati) non legata a tecnologie «proprietarie»; è quindi indipendente (entro certi limiti) dall'architettura hardware, dal sistema operativo e da altre caratteristiche di basso livello dei sistemi server e dei sistemi client impiegati dagli utenti
  - Permettono di realizzare un'**architettura applicativa simile a quella client/server**, consentendo a più utenti contemporaneamente di fruire della stesso programma
  - Permettono di utilizzare un **client universale** (il web browser) basato su standard «aperti» e disponibile su tutte le piattaforme client (Microsoft Windows, Linux, Apple OS X, Android, Apple iOS, ecc.); non è necessario *distribuire* e *aggiornare* un client specifico sui client
  - **Utilizzano HTML per la definizione dell'interfaccia con l'utente**, consentendo al tempo stesso di definire interfacce gradevoli, usabili e piuttosto sofisticate, ad un «costo» molto contenuto (l'impegno necessario è inferiore a quello richiesto per definire interfacce utente grafiche per Windows o X11, ad esempio)
  - L'architettura applicativa (entro certi limiti) è **indipendente dal linguaggio di programmazione** adottato per codificare l'applicazione
  - Possono essere facilmente progettate in modo tale da **garantire una «scalabilità orizzontale»** per fronteggiare carichi di lavoro crescenti nel tempo

Applicazioni "web based"

- Il World Wide Web è un insieme di tecnologie (computer hardware, reti di computer basate su protocollo TCP/IP, sistemi operativi che offrono servizi applicativi) nato per **pubblicare informazioni distribuite su diversi nodi della rete Internet**
- Le stesse tecnologie sono state poi adottate per costruire l'architettura applicativa che oggi è la più diffusa sul mercato della tecnologia informatica: l'architettura delle **applicazioni web based**
- In questo ambito si sfrutta il **protocollo applicativo HTTP**, il **linguaggio HTML**, l'**interfaccia CGI** o tecnologie più moderne ed ingegnerizzate, come **JEE**, **Microsoft .net**, **Ajax**, i **Web Services SOAP** e **RESTful**, ecc. per costruire programmi in grado di interagire con gli utenti presentandosi come un "sito web" composto da pagine ipertestuali collegate fra loro



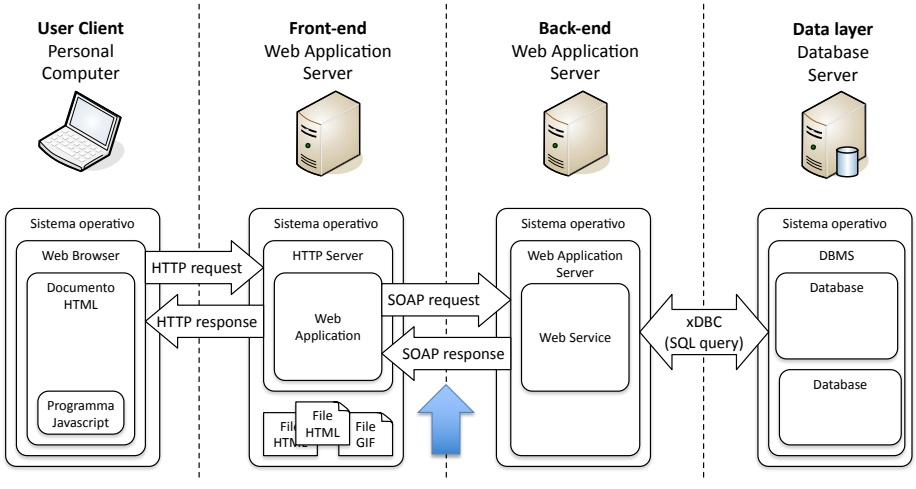
Applicazioni "web based"

- Nel tempo a queste tecnologie di base (HTTP, HTML, CGI) se ne sono aggiunte delle altre, per offrire soluzioni informatiche sempre più ingegnerizzate, robuste e flessibili: linguaggi di programmazione Object Oriented (Java, C#, ...), interi *framework software* in grado di "semplificare" lo sviluppo di applicazioni web based, ecc.
- Per la natura aperta e integrata del modello proposto dal web, le tecnologie che hanno avuto successo sono quelle che hanno sposato gli **standard** o che sono state costruite **proponendo degli standard aperti**, fruibili anche da altri

- **WEB 2.0**: oggi, per aumentare l'interazione con l'utente e realizzare applicazioni compatibili con gli standard dei dispositivi mobili (*smartphone*, *tablet*, *smartwatch*, ...), si aggiunge anche un'importante elaborazione "lato client":
  - l'output prodotto dal programma lato server o il file HTML "statico" **contiene anche istruzioni di programma** in linguaggio **Javascript**
  - **il client esegue le istruzioni Javascript presenti nella pagina** e modifica dinamicamente il contenuto della pagina stessa, anche in base alle azioni compiute dall'utente sugli oggetti presenti nella pagina (bottoni, icone, campi di input, ecc.)
  - il programma Javascript contenuto nella pagina **interagisce con il server acquisendo ed inviando informazioni senza dover ricaricare una nuova URL** nel browser dell'utente (l'interazione avviene mediante Web Services REST e flussi di informazioni JSON)

Web Services

- La stessa tecnologia impiegata per le applicazioni web è stata usata per **mettere in comunicazione tra di loro dei programmi** (e non *utenti* e *programmi*), sviluppando la tecnologia dei Web Services



Web Services

- Protocolli (sopra ad HTTP): **SOAP** (*Simple Object Application Protocol*), **REST** (*REpresentational State Transfer*)
- Le informazioni non sono codificate in HTML (non devono essere presentate ad un utente), ma in **XML**, che offre una codifica più flessibile
  - XML permette di definire "tag" per codificare in modo aperto le informazioni necessarie in uno specifico contesto applicativo
  - sia le richieste del client al server che le risposte del server al client sono codificate in strutture XML denominate **SOAP Envelope** (buste SOAP), costituite da un *header* e da un *body*

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/
soap/envelope/">
 <soap:Body>
  <getProduct xmlns="http://acme.it/ws">
   <productId>827635</productId>
  </getProduct>
 </soap:Body>
</soap:Envelope>
```

```
<soap:Envelope xmlns:soap="http://schemas.../soap/envelope/">
 <soap:Body>
  <getProductResponse xmlns="http://acme.it/ws">
   <getProductResult>
    <productName>Biscotti Gentilini</productName>
    <productId>827635</productId>
    <description>Biscotti secchi da colazione</description>
    <price>1,70</price>
    <inStock>true</inStock>
   </getProductResult>
  </getProductResponse>
 </soap:Body>
</soap:Envelope>
```

Componenti dell'architettura delle applicazioni web based

Per comprendere l'architettura applicativa di un programma web based, è necessario studiarne le componenti:

- **linguaggio HTML** per la codifica delle informazioni presentate al client e la costruzione dell'interfaccia utente della web application
- **protocollo HTTP** e codifica delle **URL** per l'identificazione delle risorse presenti sul web;
- meccanismo di **scambio dei dati in input e in output** tra il web client e il web server (vedremo il meccanismo noto come **CGI**, *Common Gateway Interface*)

Elaborazione automatica di testi

Elaborazione di documenti mediante il computer… Cosa significa? Qual è lo scopo?

- **Comporre** testi con l'ausilio della macchina (è più comodo)
- Definire l'**aspetto del testo** con qualità tipografica
- **Riprodurlo** in più copie rapidamente
- **Modificare il testo** o integrarlo successivamente
- Memorizzare, **archiviare** documenti
- Eseguire delle **ricerche** su un archivio documentale
- **Trasmettere**, **condividere** documenti con altri

Strumenti – Word processor

- Il ***word processor*** è una delle applicazioni più diffuse sui personal computer.
- Consente di comporre e modificare un documento, consente di archiviarlo, stamparlo.
- Il documento può essere condiviso con chi utilizza lo **stesso programma** (e forse anche la stessa macchina, lo stesso sistema operativo, …).
- Devo conoscere il **formato dei dati** per poter costruire strumenti di ricerca di informazioni sull'archivio documentale

Strumenti – Linguaggi di marcatura

- Il linguaggio TEX (e LATEX) permette la costruzione di documenti secondo un ***formato aperto*** (noto a tutti), aumentando così la possibilità di condivisione e scambio
- Necessita di un "**motore di *rendering***" (un programma) per poter produrre una copia leggibile del documento
- Permette di legare tra loro le informazioni secondo una struttura gerarchica e sequenziale

HTML: HyperText Markup Language

- È un "**linguaggio di marcatura**" del testo:
  - Non è un linguaggio di programmazione
  - È costituito da un insieme di parole chiave (***tag*** o marcatori) con cui identifica porzioni del documento
  - È nato con l'obiettivo di consentire la definizione di **documenti ipertestuali distribuiti** su una rete di computer
- HTML viene definito come applicazione di **SGML** (*Standard Generalized Markup Language*)

HTML: Tag per il mark-up del testo

- I tag consentono di contrassegnare (marcare) una porzione del documento per associargli una determinata caratteristica (strutturale, tipografica, ecc.).
- I tag sono costituiti da un elemento di **inizio-marcatura** ed un elemento di **fine-marcatura**; i tag sono ***case insensitive***.
- Gli elementi di inizio marcatura possono essere caratterizzati da **attributi** a cui è associato un particolare **valore**:

  <*tag attributo*$_1$=“*valore*$_1$” *attributo*$_n$=“*valore*$_n$”> … </*tag*>
- Esempio:

```
<p align="center">Paragrafo del documento</p>
```

HTML: Entità

- Per garantire la massima **compatibilità** e **trasportabilità** del documento, HTML utilizza la codifica ASCII standard a 7 bit (caratteri da 1 a 127)
- Per rappresentare **simboli speciali** o **caratteri nazionali** utilizza alcune entità
- Esempi:

  `&agrave;` = “à”, `&egrave;` = “è”, `&eacute;` = “é”,
  `&Egrave;` = “È”, `&uuml;` = “ü”, ecc.
  `&` = “&”, `&quot;` = “ ” ”, `<` = “<“, `>` = “>”, ecc.

Scopo di HTML

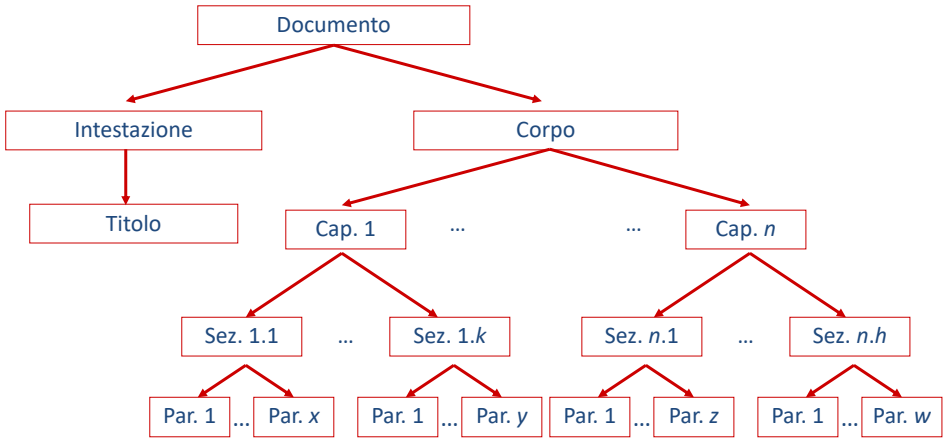
- Con HTML è possibile:
  - Definire la **struttura logica** del documento
  - Definire dei **collegamenti ipertestuali** fra documenti (*link*)
  - Definire l'**interfaccia utente** di una applicazione web
  - Definire la **formattazione tipografica** del testo, fornendo ***indicazioni*** utili per il *rendering*
- L'**errore di sintassi** non è previsto nell'ambito dell'elaborazione di documento HTML: ciò che non è riconducibile ad un tag del linguaggio viene ignorato e visualizzato in quanto tale.

Struttura del documento HTML: il Document Object Model

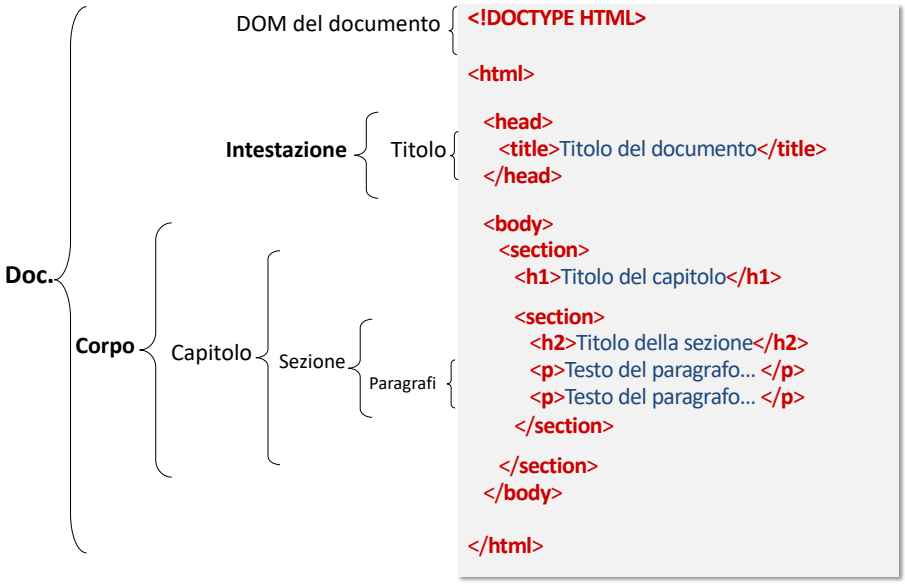
- Definire la struttura logica del documento significa identificarne delle componenti riconducibili ad elementi di una **struttura gerarchica** nell'ambito di un modello denominato **DOM**: *Document Object Model* (specificato con il tag "**DOCTYPE**")
- Il DOM di HTML prevede la seguente struttura gerarchica:

Struttura del documento HTML: il Document Object Model

- Lo stesso modello può essere rappresentato con un albero:



HTML: tag per la definizione della struttura del documento



HTML: rendering del documento

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Primo esempio</title>
    </head>
    <body>
        <article>
        <h1>Titolo del capitolo</h1>
        <section>
        <h2>Titolo della sezione</h2>
        <p>Testo del paragrafo…</p>
        </section>
        </article>
    </body>
</html>
```

HTML: ancora sulla struttura "logica" del testo

- I tag che definiscono la struttura del documento consentono al programma che lo visualizza di **evidenziare graficamente** le porzioni di testo caratterizzate da un determinato "ruolo" (titoli, sottotitoli, ecc.)
- Oltre alla gerarchia delle parti che compongono il documento (**article**, **section** anche nidificate) e dei titoli (**h1**, **h2**, …, **h6**) esistono altri tag per caratterizzare il testo; es.: ****…****: enfatizza (in qualche modo) una frase *importante* evidenziandone i caratteri
- Anche la spaziatura tra i paragrafi e il tipo di carattere da utilizzare viene **stabilito autonomamente** dal programma che visualizza il documento

HTML: liste ed elenchi

- Elenchi puntati:

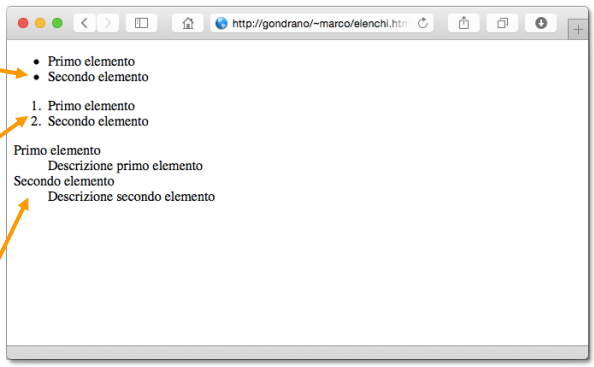
```
<ul>
 <li> Primo elemento</li>
 <li> Secondo elemento</li>
</ul>
```

- Liste numerate:

```
<ol>
 <li> Primo elemento</li>
 <li> Secondo elemento</li>
</ol>
```

- Liste con descrizione:

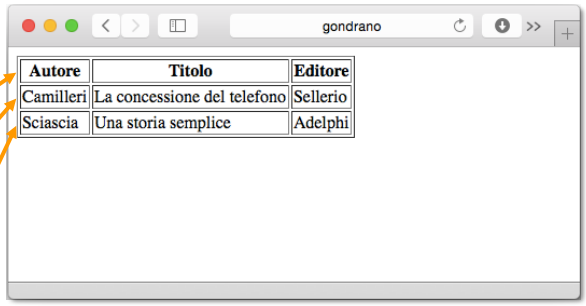
```
<dl>
 <dt> Primo elemento</dt>
 <dd> Descrizione primo elemento</dd>
 <dt> Secondo elemento</dt>
 <dd> Descrizione secondo elemento</dd>
</dl>
```



HTML: tabelle

- Tabella:

```
<table border="1">
 <tr>
  <th>Autore</th>
  <th>Titolo</th>
  <th>Editore</th>
 </tr>
 <tr>
  <td>Camilleri</td>
  <td>La concessione del telefono</td>
  <td>Sellerio</td>
 </tr>
 <tr>
  <td>Sciascia</td>
  <td>Una storia semplice</td>
  <td>Adelphi</td>
 </tr>
</table>
```



Spesso le tabelle sono utilizzate (impropriamente) anche per disporre e allineare gli elementi sulla pagina (es.: i campi e le etichette di una form)

HTML: documenti ipertestuali

- I documenti HTML possono contenere riferimenti (***link* ipertestuali**) ad altri documenti
- In questo modo la lettura del testo può **non** essere rigidamente sequenziale: invece di una sequenza di pagine (come in un libro) ottengo una **rete di pagine** (una sorta di *grafo* con pagine fra loro correlate)
- Il **riferimento ad una risorsa** (documento o altro) viene espresso mediante un **URL** (*Uniform Resource Locator*)
- Con un URL è possibile descrivere la **modalità** con cui deve essere raggiunto il documento correlato e la sua **collocazione**:

```
http://www.mat.uniroma3.it/users/liverani/index.html
```

protocollo *hostname* *path* *filename*

HTML: tag per la creazione di link

- Il tag per la definizione di riferimenti ipertestuali (***h*ypertextual *ref*erence**) è

```
<a href="URL" target="finestra">testo…</a>
```

- Il riferimento ipertestuale può spingersi fino ad un punto particolare del documento indirizzato da un URL: possono essere definite delle "**ancore**" interne al documento per indicare punti di aggancio di un link:

```
<a href="URL#etichetta">testo…</a>
```

e

```
<a name="etichetta">altro testo…</a>
```

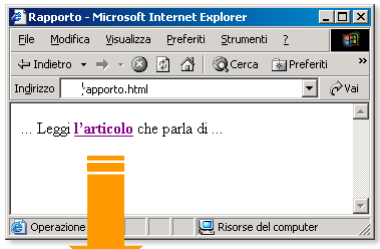
HTML: collegamenti ipertestuali

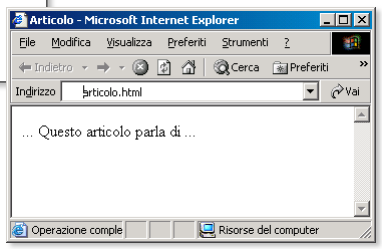
rapporto.html

… Leggi l'articolo che parla di …

articolo.html

… Questo articolo parla di …

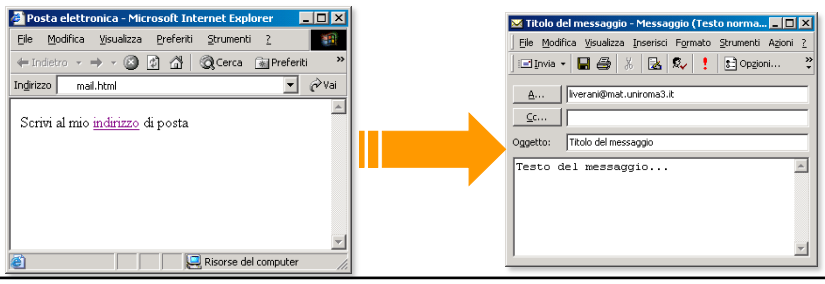




Le due pagine collegate possono essere su server differenti: il documento può essere fisicamente ***distribuito*** (dislocato in più punti)

HTML: altri tipi di link

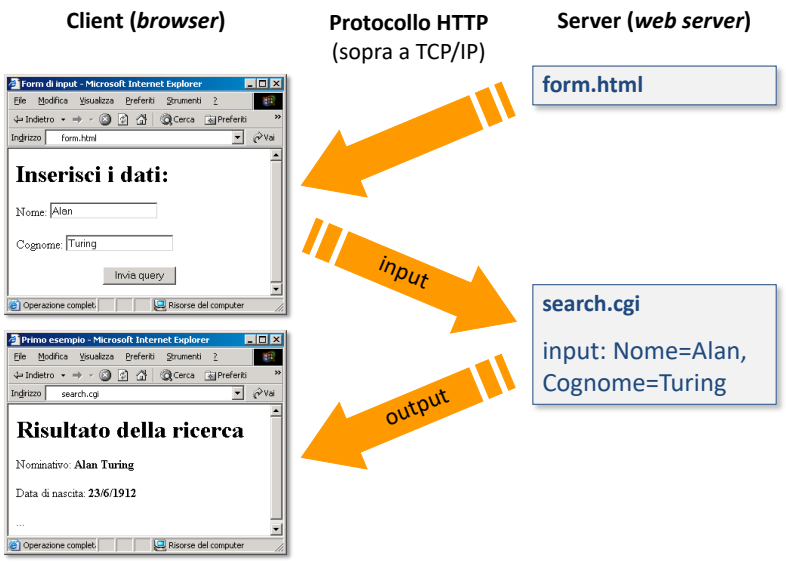
- Definendo un link è possibile indicare il **protocollo** con cui potrà essere raggiunta la risorsa collegata (documento, file, …).
- Se il protocollo è differente da HTTP (es.: FTP, MAILTO, …) verrà attivato un **programma adatto** a trattarlo (a volte lo stesso browser gestisce più protocolli).
- Se il tipo di documento richiesto non può essere visualizzato dal browser (es.: documento Word, PDF, Postscript, ecc.), tipicamente viene attivato un programma adeguato, ovvero il file viene salvato sul computer.
- Esempi:
  - Scarica il file file.zip…
  - Scrivi al mio indirizzo di posta



HTML: form di inserimento dati

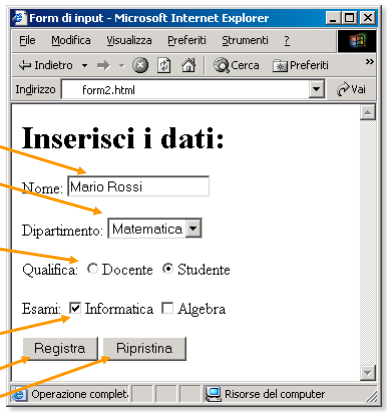
- Le pagine web possono essere usate anche come "interfaccia" verso un programma applicativo web based (es.: **programmi CGI**)
- Il programma produrrà un **output in formato HTML** per poterlo visualizzare nel browser e **acquisirà l'input** da parte dell'utente attraverso delle maschere di inserimento dati denominate ***form***
- Una *form* è composta da
  - un insieme di **campi di input** caratterizzati da un **nome** (univoco nell'ambito della form) ed un **tipo**, che ne determina il comportamento funzionale
  - un'**azione**, ossia la **URL del programma** che riceverà i dati inseriti dall'utente e si occuperà della loro elaborazione restituendo a sua volta un output (ad esempio in formato HTML)
  - l'indicazione del **metodo HTTP** (GET o POST) con cui i dati inseriti dall'utente nei campi della form saranno inviati al server web e all'applicazione che implementa l'azione
- Quando l'utente ha terminato la compilazione della *form* viene eseguito il programma specificato come "azione" della *form* passandogli come input i dati inseriti dall'utente nei campi

Flusso di un programma web based



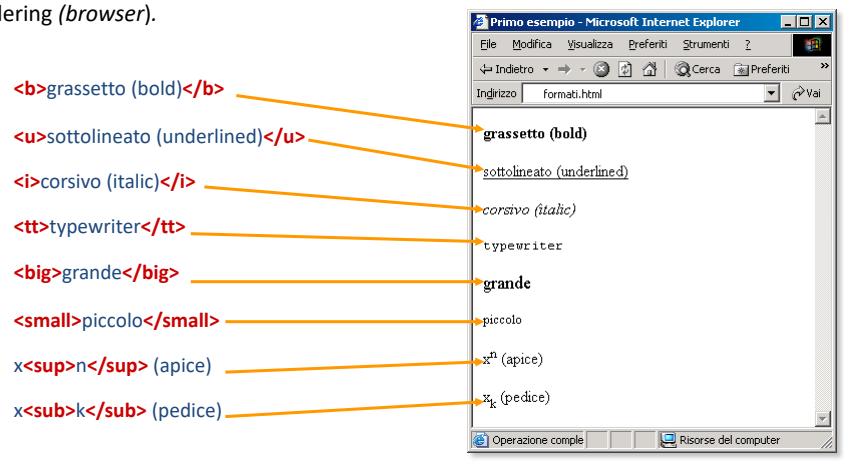
HTML: form di inserimento dati

```
<html>
 <head><title>Form di input</title></head>
 <body>
  <h1>Inserisci i dati:</h1>
  <form action="search.cgi" method="GET">
  <p>Nome: <input type="text" name="nome"></p>
  <p>Dipartimento: <select name="dip">
                     <option value="mat">Matematica
                     <option value="fis">Fisica
                   </select></p>
  <p>Qualifica: <input type="radio" name="qualifica"
    value="prof" checked>Docente <input type="radio"
    name="qualifica" value="stud">Studente </p>
  <p>Esami: <input type="checkbox" name="esami"
    value="in1">Informatica <input type="checkbox"
    name="esami" value="al1">Algebra</p>
  <p><input type="submit" value="Registra">
    <input type="reset" value="Ripristina"></p>
  </form>
 </body>
</html>
```



HTML: formattazione tipografica

- È possibile fornire delle *indicazioni* per il ***rendering*** del documento HTML (per la sua rappresentazione tipografica).
- Tali indicazioni **possono essere trascurate** dal programma visualizzatore, in modo da adattare al meglio la visualizzazione della pagina allo strumento fisico su cui viene eseguito il rendering *(browser)*.



HTML: caratteri, font, colori

- È possibile definire in dettaglio il tipo di carattere che deve essere utilizzato:

  <**font face**="*tipo*" **size**="*dim.*" **color**="*colore*">…</**font**>
- Questo rende **poco "trasportabile"** il documento: è legato alla disponibilità di caratteri e colori sulla macchina dell'utente
- I colori vengono definiti specificandone le componenti **RGB** (*red*, *green*, *blue*) in notazione esadecimale (valori da 00 a FF):

  #rrggbb
- Esempio: **#AB205B**

HTML: caratteri, font, colori

Con l'uso di questi tag anche una pagina molto semplice diventa complicata e meno leggibile e manutenibile; diventa difficile garantire la coerenza grafica e l'omogeneità delle diverse pagine del sito


```
<html>
 <head>
  <title>Font e colori</title>
 </head>
 <body bgcolor="#3090AA">
  <h1>Pagina colorata</h1>
  <p>
   <font color="#FFCC00" size="+1">Testo giallo</font> con il carattere
   <font face="Arial" color="#FF0000" size="+2">Arial</font> ...
  </p>
 </body>
</html>
```

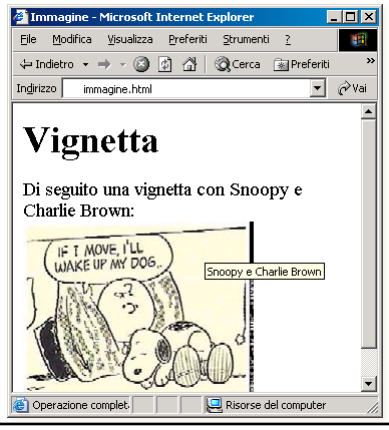
HTML: immagini grafiche

- È possibile includere ***immagini*** in un documento HTML
- L'immagine verrà visualizzata compatibilmente con le capacità del browser e dell'ambiente ospite

<**img src**="*URL*" **align**="*allineam.*" **alt**="*testo alternativo*">

- Esempio:

<**img src**="snoopy.jpg" **alt**="Snoopy e Charlie Brown">



Fogli di stile: Cascading Style Sheet (CSS)

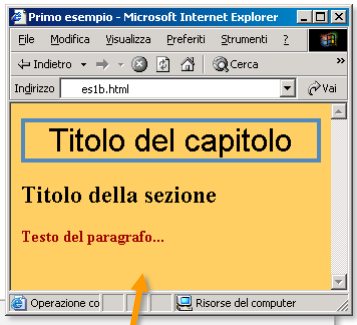
- Negli anni HTML si è arricchito di numerosi tag di caratterizzazione grafica degli elementi della pagina fino a **perdere la sua originaria eleganza e portabilità**
- Per semplificare la costruzione di pagine HTML con una forte connotazione tipografica e cromatica, ritornando all'obiettivo originale del linguaggio (definire una struttura logica del documento) sono stati introdotti i **fogli di stile** (*Cascading Style Sheet* – CSS)
- Nel CSS vengono definite le caratteristiche tipografiche dei tag presenti nella pagina
- I tag possono essere raggruppati in insiemi omogenei definendo delle **classi** nel foglio di stile CSS; lo stesso tag può essere inserito in più classi, caratterizzandolo in modi differenti
- Uno stesso foglio di stile può essere utilizzato su più documenti (o sull'output di un programma, codificato in HTML) in modo da garantire
  - **omogeneità "tipografica"** di presentazione di tutte le pagine del sito o dell'applicazione
  - **facilità di modifica** dell'impaginazione del sito/applicazione: basta modificare **un** foglio di stile per cambiare l'aspetto di tutte le pagine del sito

HTML+CSS: fornire uno stile al documento

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Primo esempio</title>
  </head>

  <body>
    <h1>Titolo del capitolo</h1>
    <h2>Titolo della sezione</h2>
    <p>Testo del paragrafo...</p>
  </body>
</html>
```

HTML+CSS: fornire uno stile al documento

```
H1 {
        font-family: Arial, Helvetica, sans-serif;
        text-align: center;
        font-weight: normal;
        border-color: #5588bb;
        border-style: solid;
}
p {
        font-family: Times New Roman, serif;
        text-align: justify;
        color: #990000;
        font-weight: bold;
}
body {
        background: #FFCC66;
}
```




```
<!DOCTYPE HTML>
<html>
 <head>
  <title>Primo esempio</title>
  <link rel=stylesheet type="text/css" href="style.css">
 </head>
 <body>
        <h1>Titolo del capitolo</h1>
        <h2>Titolo della sezione</h2>
        <p>Testo del paragrafo...</p>
 </body>
</html>
```

Server Web (server HTTP)

- Un server Web è un programma che implementa la **componente server del protocollo HTTP**
- Il protocollo HTTP è un protocollo applicativo che opera al livello 4 dello stack TCP/IP (utilizza tipicamente la porta TCP 80)
- Il protocollo **HTTPS** (porta TCP 443) è il medesimo protocollo reso sicuro mediante la cifratura dei dati trasmessi, mediante l'uso di **SSL** (*Secure Socket Layer*); il protocollo HTTPS garantisce anche l'identità del server
- Il protocollo **HTTP/1.1** implementa i seguenti comandi:
  - **GET**: il client richiede una specifica "risorsa" al server indicandone la URL
  - **HEAD**: il client richiede l'header della risposta HTTP per la richiesta di una risorsa indicata attraverso una URL, ma non il contenuto della risorsa stessa
  - **POST**: il client richiede una specifica "risorsa" al server ed invia delle informazioni al server, che seguono la richiesta HTTP
  - **PUT**: il client invia al server una risorsa che deve essere memorizzata nella URL specificata
  - **DELETE**: il client richiede al server la cancellazione della risorsa specificata con una URL
  - **TRACE**, **OPTIONS**, **CONNECT**, **PATCH**: altri comandi per funzioni di servizio, spesso non implementati per ragioni di sicurezza
- I comandi più utilizzati sono **GET** e **POST**

Server Web (server HTTP)

- Il protocollo HTTP è un protocollo ***stateless***: al termine della risposta ad una richiesta del client la connessione viene terminata dal server
- Il server HTTP non conserva memoria dello stato della sessione utente; per gestire informazioni di sessione si ricorre all'uso di ***cookies***, piccoli file di dati che è possibile memorizzare sul web client
- Il server Web esegue questa sequenza di operazioni:
  - **Accetta la connessione** da parte di un client mediante il protocollo HTTP
  - Verifica se il client o l'utente ha l'**autorizzazione** per eseguire il collegamento
  - **Accetta una richiesta** dal client mediante il protocollo HTTP (GET, POST, ...)
  - Per richieste di **risorse statiche** (file HTML, immagine JPEG, ecc.): il server carica la risorsa e la invia al client
  - Per richieste di **risorse dinamiche**: il server esegue il programma passandogli attraverso la modalità CGI i parametri ricevuti dal client ed invia al client l'output prodotto dal programma
  - **Chiude la connessione** con il client

Server Web (server HTTP)

Sessione HTTP mediante il programma Telnet (connessione sulla porta TCP 80)

HTTP request (GET) del client

HTTP response del server: *header*

HTTP response del server: *body*

```
marco@aquilante ~$ telnet gondrano 80
Trying 10.211.55.42...
Connected to freebsd.
Escape character is '^]'.
GET /~marco/index.html http/1.1
Host: gondrano

HTTP/1.1 200 OK
Date: Sun, 26 Apr 2015 13:57:05 GMT
Server: Apache/2.4.12 (FreeBSD)
Last-Modified: Sat, 25 Apr 2015 14:54:30 GMT
ETag: "9e-5148db0a37580"
Accept-Ranges: bytes
Content-Length: 158
Content-Type: text/html

<!DOCTYPE HTML>

<html>
<head><title>Marco Home Page</title></head>
<body>
<h1>Benvenuti!</h1>
<p>Questa &egrave; la home page di Marco.</p>
</body>
</html>

Connection closed by foreign host.
```

Programmi CGI

- **CGI**, ***Common Gateway Interface***, è una delle modalità (la prima, in ordine di tempo, forse tutt'ora la più semplice, ma anche meno efficiente) con cui è possibile eseguire dei programmi su un server Web
- L'interfaccia CGI definisce il modo in cui è possibile **inviare delle informazioni** (parametri) da un client web (un web browser) all'applicazione invocata sul server web e come sia possibile **restituire al client l'output** prodotto dal programma
- I programmi che operano secondo tale modalità vengono definiti **programmi CGI**
- I programmi CGI **vengono eseguiti sul server**, non sul client
- Non esiste un linguaggio di programmazione esclusivo per la codifica di programmi CGI
  - il linguaggio Perl è uno dei principali strumenti per lo sviluppo di applicazioni CGI, ma non l'unico
  - il linguaggio Python è oggi un linguaggio di scripting molto diffuso
  - programmi CGI possono essere scritti come script della shell del sistema operativo (Bash shell script, ad esempio) o come programmi in linguaggio C compilati in linguaggio macchina

Passaggio di parametri da web browser ad applicazione CGI

- I parametri vengono forniti con un'unica stringa in formato **URL encoded**:

  $nome_1$=$valore_1$&$nome_2$=$valore_2$&…&$nome_n$=$valore_n$

- Esempio:

```
nome=Silvia&cognome=Di+Stefano&eta=23
```

- È il server HTTP che si occupa di passarli al programma CGI secondo due modalità:
  - **POST**: la stringa viene fornita sul canale standard input, il programma leggerà la stringa da **STDIN** (*standard input*)
  - **GET**: la stringa viene memorizzata nella variabile di ambiente **QUERY_STRING**
- La modalità viene specificata dal server (sulla base delle indicazioni del client) mediante la variabile di ambiente **REQUEST_METHOD**

Compiti del programma CGI

- **Acquisire** la stringa con i parametri
- Eseguire il ***parsing* dei parametri** ricevuti in input, ossia una suddivisione della stringa per isolare le coppie "nome=valore" e successivamente per separare il nome del parametro dal suo valore
- **Eseguire l'elaborazione** richiesta
- **Produrre un output** compatibile con le capacità di visualizzazione del browser (tipicamente output in formato HTML)

- L'output deve contenere una intestazione che ne indichi il **formato secondo la codifica MIME** (*Multipurpose Internet Mail Extensions*), seguito da una riga vuota
- Esempio:

```
Content-type: text/html
```

Programmi CGI in Perl

- In Perl esiste una libreria per implementare le principali funzioni della programmazione CGI: **cgi-lib.pl**
- Con l'istruzione **require** è possibile caricare una libreria nel programma Perl

```
require "cgi-lib.pl";
```

- Tra le subroutine messe a disposizione dalla libreria c'è "**ReadParse**":
  - Legge la stringa dei parametri (ricevuti indifferentemente con metodo GET o POST)
  - Suddivide i nomi dei parametri dai valori
  - Costruisce l'array associativo **%in**:

```
$in{nome} = valore
```

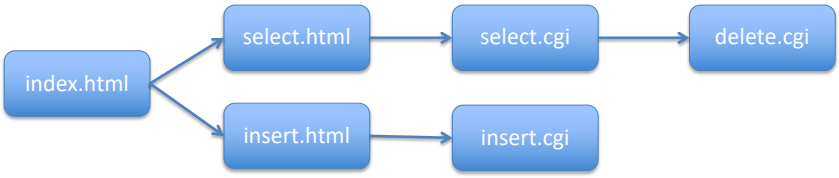
CGI in Perl: un esempio elementare

```
<html>
 <head><title>Input</title></head>
 <body>
  <form action="ciao.cgi" method="get">
  Come ti chiami? <input name="nome">
  <input type="submit">
  </form>
 </body>
</html>
```

```
#!/usr/local/bin/perl
# Programma di esempio "ciao.cgi"

require "cgi-lib.pl";
&ReadParse;
print "Content-type: text/html\n\n";
print "<html><body>\n";
print "<p>Ciao $in{nome}</p>";
print "</body></html>\n";
```

Esempio: indirizzario on-line, schema dell'applicazione

- Programma Web per l'archiviazione di indirizzi e la ricerca sul catalogo
- L'archivio è su un DBMS relazionale: database "marcodb", tabella "rubrica"
- Schema dell'applicazione:



- Sono necessari sei file:
  - **index.html**: menù principale dell'applicazione
  - **select.html**: form di input dei criteri di ricerca per selezionare i dati presenti sul DB
  - **select.cgi**: programma CGI che esegue la selezione dei dati dal DB in base ai parametri specificati
  - **delete.cgi**: programma CGI che elimina un record dal DB in base al suo "id"
  - **insert.html**: form di input dei dati da inserire come nuovo record nella tabella "rubrica" sul DB
  - **insert.cgi**: programma CGI che esegue l'inserimento dei dati in un nuovo record sul DB

Indirizzario on-line: menù principale (index.html)

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Rubrica degli indirizzi - Men&ugrave; principale</title>
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>

  <body>
    <h1>Rubrica degli indirizzi</h1>
    <h2>Men&ugrave; principale</h2>

    <dl>
      <dt><a href="select.html" title="Selezione dei dati presenti nella
          rubrica">Selezione dati</a></dt>
      <dd>Ricerca e visualizza i dati presenti nella rubrica degli indirizzi</dd>

      <dt><a href="insert.html" title="Inserimento dati in rubrica">Inserimento
          dati</a></dt>
      <dd>Inserimento di un nuovo record nell'archivio degli indirizzi</dd>
    </dl>
  </body>
</html>
```

Indirizzario on-line: inserimento criteri di ricerca (select.html)

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Rubrica degli indirizzi - Consultazione dati</title>
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>

  <body>
    <h1>Rubrica degli indirizzi</h1>
    <h2>Consultazione dati</h2>

    <form action="select.cgi" method="GET">
      <table>
       <tr><th>Nome</th><td><input name="nome" type="text" size="20"></td></tr>
       <tr><th>Cognome</th><td><input name="cognome" type="text" size="30"></td></tr>
       <tr><th>E-mail</th><td><input name="mail" type="text" size="30"></td></tr>
       <tr><td colspan="2" align="center"><input type="submit" value="Cerca”>
           <input type="reset" value="Ripristina"></td></tr>
      </table>
    </form>
  </body>
</html>
```

Indirizzario on-line: script CGI di selezione dei dati (select.cgi)

```
#!/usr/local/bin/perl
require "cgi-lib.pl";
&ReadParse;
print "Content-Type: text/html\n\n";
print "<!DOCTYPE HTML>\n";
print "<html>\n<head><title>Rubrica degli indirizzi - Consultazione dati</title>\n";
print "<link rel=\"stylesheet\" type=\"text/css\" href=\"style.css\">\n</head>\n";
print "<h1>Rubrica degli indirizzi</h1>\n";
print "<h2>Consultazione dati</h2>\n";
use DBI;
$db = DBI->connect("dbi:mysql:dbname=marcodb", "marco", "marco") or die DBI::errstr;
$query = $db->prepare("select id, nome, cognome, email, tel, data_nascita from rubrica
    where nome like '%$in{nome}%' and cognome like '%$in{cognome}%' and email like '%$in{email}%’
    order by cognome, nome");
$query->execute();
if ($query->rows() == 0) {
  print "<p>Nessun record selezionato.</p>";
} else {
  print "<table>\n<tr><th>Nome</th><th>Cognome</th><th>E-mail</th><th>Telefono</th><th>Data di
      nascita</th><th>Canc.</th></tr>\n";
  for ($i=1; $i <= $query->rows(); $i++) {
    ($id, $nome, $cognome, $email, $tel, $data) = $query->fetchrow();
    print "<tr><td>$nome</td><td>$cognome</td><td><a
      href=\"mailto:$email\">$email</a></td><td>$tel</td><td>$data</td><td><a href=\"delete.cgi?id=$id\"
      title=\"Elimina n. $id\">X</a></td></tr>\n";
  }
  print "</table>\n";
}
$query->finish();
$db->disconnect();
print "<p>Torna <a href=\"index.html\" title=\"Men&ugrave; principale\">men&ugrave; principale</a></p>\n";
print "</body>\n</html>\n";
```

Indirizzario on-line: script CGI di cancellazione dei dati (delete.cgi)

```
#!/usr/local/bin/perl
require "cgi-lib.pl";
&ReadParse;
print "Content-Type: text/html\n\n";
print "<!DOCTYPE HTML>\n";
print "<html>\n<head><title>Rubrica degli indirizzi-Cancellazione dati-OK</title>\n";
print "<link rel=\"stylesheet\" type=\"text/css\" href=\"style.css\">\n</head>\n";
print "<h1>Rubrica degli indirizzi</h1>\n";
print "<h2>Cancellazione di un contatto</h2>\n";
use DBI;
$db = DBI->connect("dbi:mysql:dbname=marcodb", "marco", "marco") or die DBI::errstr;
$query = $db->prepare("delete from rubrica where id='$in{id}'");
$query->execute();
$query->finish();
$db->disconnect();

print "<p>Cancellazione del record n. $in{id} eseguita correttamente.</p>\n";
print "<p>Torna al <a href=\"index.html\" title=\"Men&ugrave;
principale\">men&ugrave; principale</a>.</p>\n";
print "</body>\n</html>\n";
```

Indirizzario on-line: Form inserimento dati (insert.html)

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Rubrica degli indirizzi - Inserimento dati</title>
    <link rel=\"stylesheet\" type="text/css" href="style.css">
  </head>

  <body>
    <h1>Rubrica degli indirizzi</h1>
    <h2>Inserimento di un nuovo contatto</h2>

    <form action="insert.cgi" method="GET">
      <table>
        <tr><th>Nome</th><td><input name="nome" type="text" size="20"></td></tr>
        <tr><th>Cognome</th><td><input name = "cognome" type="text" size="30"></td></tr>
        <tr><th>E-mail</th><td><input name = "mail" type="text" size="30"></td></tr>
        <tr><th>Telefono</th><td><input name = "telefono" type="text" size="15"></td></tr>
        <tr><th>Data di nascita</th><td><input name = "giorno" type="text" size="2">/
          <input name="mese" type="text" size="2">/<input name="anno" type="text"
          size="4"></td></tr>
        <tr><td colspan="2" align="center"><input type="submit" value="Salva">
          <input type="reset" value="Ripristina"></td></tr>
      </table>
    </form>
  </body>
</html>
```

Indirizzario on-line: script CGI di inserimento dati

```
#!/usr/local/bin/perl
require "cgi-lib.pl";
&ReadParse;
print "Content-Type: text/html\n\n";
print "<!DOCTYPE HTML>\n";
print "<html>\n<head><title>Rubrica degli indirizzi - Inserimento dati - OK</title>\n";
print "<link rel=\"stylesheet\" type=\"text/css\" href=\"style.css\">\n</head>\n";
print "<h1>Rubrica degli indirizzi</h1>\n";
print "<h2>Inserimento di un nuovo contatto</h2>\n";
use DBI;
$db = DBI->connect("dbi:mysql:dbname=marcodb", "marco", "marco") or die DBI::errstr;
$query = $db->prepare("insert into rubrica (nome, cognome, email, tel, data_nascita)
  values ('$in{nome}', '$in{cognome}', '$in{mail}', '$in{telefono}', '$in{anno}-$in{mese}-$in{giorno}')");
$query->execute();
$query->finish();
$db->disconnect();
print "<p>L'inserimento dei dati di $in{nome} $in{cognome} &egrave; stato eseguito correttamente.</p>\n";
print "<p>Torna al <a href=\"index.html\" title=\"Men&ugrave; principale\">men&ugrave; principale</a>.
 </p>\n";
print "</body>\n</html>\n";
```

Indirizzario on-line: foglio di style (style.css)

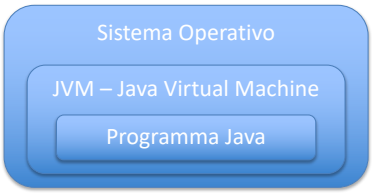
```
body {
  padding: 0pt;
  margin: 0pt;
}

p {
  margin-left: 8pt;
}

h1 {
  font-family: sans-serif;
  font-size: 20pt;
  color: white;
  background: #E0E0E0;
  display: block;
  padding: 8pt;
  margin: 0pt;
  box-shadow: 0px -5px 5px #B0B0B0
inset;
}

h2 {
  font-family: sans-serif;
  font-size: 18pt;
  display: block;
  padding: 8pt;
  color: green;
}
```

```
a {
  text-decoration: none;
}

a:hover {
  color: red;
}

dt a {
  text-decoration: none;
  padding: 3pt;
  color: green;
  border: solid 2pt white;
}

dt a:hover {
  background: #EEFFEE;
  color: green;
  border: solid 2pt green;
}

dt {
  font-weight: bold;
  font-family: sans-serif;
  padding: 5pt;
}

dd {
  padding-bottom: 20pt;
}
```

```
table {
  margin-left: 8pt;
}

form th {
  text-align: right;
  background: white;
  border-bottom: none;
}

form td {
  border-bottom: none;
}


th {
  background: #EEFFEE;
  border-bottom: 3pt solid green;
  padding: 3pt;
}

td {
  border-bottom: 1pt solid green;
  padding: 3pt;
}
```

Il linguaggio Java, l'ambiente di sviluppo e di esecuzione

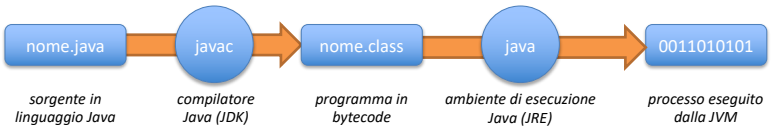
- **Java** è un linguaggio **Object Oriented** molto diffuso a partire dagli anni '90
- È uno dei linguaggi più diffusi per lo sviluppo di applicazioni web based e per dispositivi mobili
- Il linguaggio Java è **indipendente dalla piattaforma** su cui viene eseguito
- L'indipendenza dalla piattaforma viene realizzata mediante la **Java Virtual Machine**, un ambiente di esecuzione per i programmi Java che fornisce un'astrazione del computer e di alcuni dei servizi del sistema operativo: in questo modo il programma sfrutta le API della JVM e non quelle del sistema operativo ospite



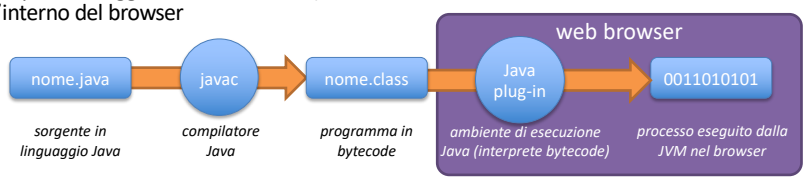
- **JDK** (*Java Development Kit*) è l'ambiente di sviluppo per Java: il compilatore **javac** traduce il sorgente Java in **bytecode**, una sorta di linguaggio macchina per la JVM
- **JRE** (*Java Runtime Environment*) è l'ambiente di esecuzione (la JVM) per i programmi Java codificati in *bytecode*

Java Application e Java Applet

- Un programma in linguaggio Java è una **Java Application**: è eseguita su un computer mediante una Java Virtual Machine che offre l'ambiente di esecuzione del *bytecode*



- Una Java Application può implementare un'interfaccia grafica (GUI) indipendente dall'interfaccia del sistema operativo ospite, mediante i package SWING e AWT (*abstract window toolkit*)
- Un **Applet** è un programma Java eseguito all'interno del web browser mediante un plug-in (componente aggiuntivo del browser) che realizza un ambiente di esecuzione e una JVM all'interno del browser



- Con un Applet vengono impegnate le risorse del client; l'applet ha numerosi vincoli imposti dalle politiche di sicurezza del browser e della JVM realizzata come plug-in

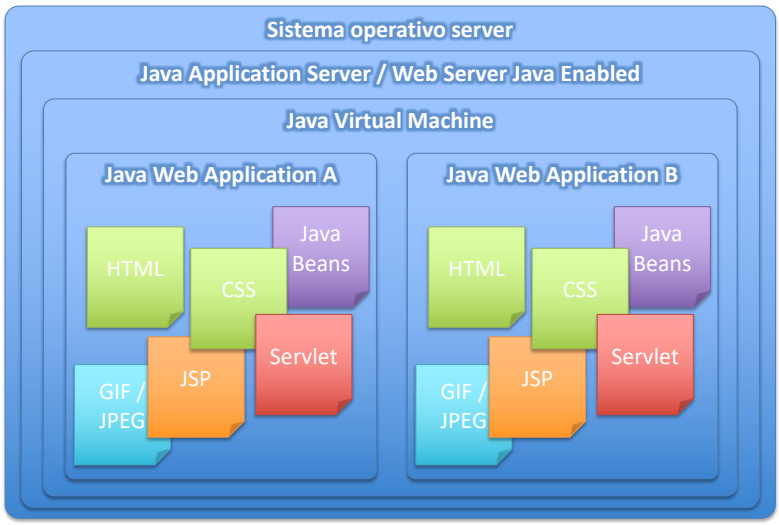
JEE: Java Enterprise Edition

- **JEE** (*Java Enterprise Edition*) è una specifica aperta per la realizzazione di un'architettura per applicazioni web basate sul **linguaggio Java** e su una serie di componenti software rese disponibili alle web application rese disponibili nell'ambito dell'architettura
- L'architettura è basata su un **Application Server JEE** che rende disponibili le componenti JEE e fornisce un ambiente di esecuzione per le applicazioni Java sviluppate secondo la specifica
- Esistono numerosi prodotti di mercato o open source di tipo Application Server JEE (es.: Glassfish, JBoss, IBM WebSphere, ecc.)
- Gli *Application Server JEE* ed i *Web Server Java Enabled* implementano il protocollo HTTP offrendo delle classi di oggetti alle applicazioni Java per ricevere i dati in input e restituire in output una pagina codificata in HTML (al pari di quanto offerto da CGI)
- Le **componenti JEE** sono di tre tipi
  1. **Componenti web**: Servlet e JSP, MVC (Model View Controller, una specifica per l'implementazione della parte di front-end dell'applicazione), JSF (Java Server Faces), ecc.
  2. **Componenti enterprise**: EJB (Enterprise Java Beans), JMS (Java Message Service), JNDI (Java Naming and Directory Interface), ecc.
  3. **Componenti per l'interazione con il database**: JDBC (Java Database Connectivity), Java Transaction API, Java Persistence API, ecc.
- Ad eccezione delle componenti Enterprise una web application Java può essere resa disponibile anche tramite un **Web Server Java Enabled** (es.: Apache Tomcat)

JEE: Java Enterprise Edition

- I due tipi di programma Java per la realizzazione di una web application sulla base della specifica JEE sono le **Servlet** e le **JSP**
- **Servlet**
  - è un programma Java che viene compilato in bytecode dall'Application Server JEE la prima volta che viene eseguito
  - sfrutta appositi oggetti e metodi messi a disposizione dall'Application Server JEE per ricevere i dati passati tramite i metodi GET o POST di HTTP
  - di fatto è l'equivalente di un programma CGI nel contesto JEE
- **JSP (Java Server Pages)**
  - si presenta come una pagina HTML contenente dei tag speciali che delimitano istruzioni scritte in linguaggio Java
  - la pagina JSP viene compilata e trasformata nel bytecode di un servlet la prima volta che viene richiamata dal browser di un utente
  - anche la pagina JSP sfrutta appositi oggetti e metodi messi a disposizione dall'Application Server JEE per ricevere i dati passati tramite i metodi GET o POST di HTTP
- Rispetto ad uno script CGI un programma JEE
  - è molto più efficiente perché compilato in bytecode
  - è più efficiente e meno oneroso in termini di risorse del sistema operativo perché non viene aperto un processo per ciascuna chiamata al programma (come avviene per i CGI)
  - è portabile da una piattaforma ad un'altra (anche su sistemi operativi diversi)
- Servlet e JSP non usano AWT e SWING, ma HTML per produrre l'interfaccia con l'utente

Java Web Application



Java Servlet

- Una **servlet** è una classe Java eseguibile da un *Web Server Java Enabled*, ossia dotato di una componente denominata *servlet container*
- Una **servlet** è un programma pubblicato in rete dal Web Server come una qualsiasi altra risorsa web (una pagina HTML, un'immagine, uno script CGI, ecc.): anche una servlet è identificata da una specifica URL
- Le servlet operano come gli script CGI in una **modalità request/response**:
  - il client (browser) dell'utente ne fa richiesta tramite la URL associata
  - il server attiva la servlet e la esegue
  - il server ne restituisce il risultato, ciò che il server restituisce come "output", come contenuto della risorsa (es.: formattato in HTML) e chiude la connessione
- Le servlet API di Java permettono di programmare una servlet in maniera completamente indipendente dalle caratteristiche del server, del client e del protocollo di trasferimento
- Il codice del programma servlet è normale codice Java: può usare tutte le librerie e le utility del linguaggio, tra cui la connessione a DBMS tramite JDBC, l'uso di XML tramite JAXP, ecc.
- Le servlet sostituiscono i classici script CGI fornendo un grado di sicurezza maggiore, e un livello di astrazione superiore per il programmatore

Le slide su Java Servlet sono tratte in parte dalle dispense del prof. Giuseppe Della Penna del Dipartimento di Informatica dell'Università degli Studi dell'Aquila (http://www.di.univaq.it/gdellape/)

Java Servlet e Web Applications

- Una **Java Web Application** è un insieme di servlet, JSP, pagine HTML, immagini, file di stile, librerie (package) Java raccolti in una directory del web application server
- I file che compongono la Java Web Application devono essere organizzati in **sottodirectory specifiche** in modo tale da consentire al server di accedere alle risorse pubblicate nell'ambito della *web application*:
  - La sottodirectory **`WEB-INF`**, contiene alcuni file di configurazione, tra cui il *web application deployment descriptor* (`web.xml`)
  - La sottodirectory **`WEB-INF/classes`**, contiene le classi Java dell'applicazione, comprese le servlet; secondo le convenzioni Java, le singole classi andranno poste all'interno di un albero di directory corrispondente al loro package
  - La sottodirectory **`WEB-INF/lib`**, contiene le librerie JAR necessarie all'applicazione, comprese quelle di terze parti, come i driver JDBC
  - Tutte le altre sottodirectory del contesto, compresa la stessa *root directory*, conterranno normali file come pagine HTML, fogli di stile, immagini o pagine JSP
- Spesso per la pubblicazione di una Java Web Application si utilizza raccogliere tutti i file e le sottodirectory dell'applicazione in un file di tipo **WAR** (*Web Archive*): viene poi eseguito il *deploy* del WAR sull'Application Server che eseguirà l'estrazione dei file dal WAR e la loro collocazione nella sottodirectory che identifica il contesto
- Il file WAR spesso viene prodotto mediante un ambiente di sviluppo **IDE** (*Integrated Development Environment*) come Eclipse o NetBeans

Java Servlet e Web Applications

- Per rendere una classe Java disponibile come risorsa di tipo *servlet*, è necessario configurarne le caratteristiche tramite un file detto ***web application deployment descriptor***
- Questo file, denominato **`web.xml`** e codificato in XML, deve essere posto nella directory WEB-INF dell'applicazione
- Ciascuna servlet è configurata in un elemento `<servlet>` distinto; l'elemento `<servlet-class>` deve contenere il nome completo della classe che implementa la servlet
- È necessario mappare ciascuna servlet su una URL tramite l'elemento `<servlet-mapping>`
  La componente `<url-pattern>` specificata andrà a comporre la URL per la servlet nel seguente modo:
  http://*indirizzo*/*contesto*/*url pattern*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
 <display-name>Progetto Web</display-name>
 <description>Bla bla bla...</description>
 <servlet>
  <servlet-name>Servlet1</servlet-name>
  <description>Implementa la funzione Y</description>
  <servlet-class>org.iw.project.class1</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>Servlet1</servlet-name>
  <url-pattern>/funzione1</url-pattern>
 </servlet-mapping>
</web-app>
```

Java Servlet

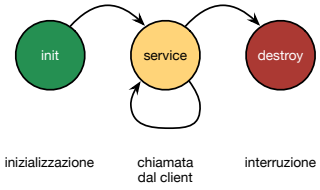
- Alla base dell'implementazione di una servlet c'è l'**interfaccia `Servlet`**, che è implementata da una serie di classi base come **`HttpServlet`**; tutte le servlet che vengono create sono derivate (*extends*) da quest'ultima
- Le altre due classi base per la creazione di una servlet sono **`ServletRequest`** e **`ServletResponse`**
  - Un'istanza di **`ServletRequest`** viene passata dal contesto alla servlet quando viene invocato, e contiene tutte le informazioni inerenti la richiesta effettuata dal client: queste comprendono, ad esempio, i parametri GET e POST inviati dal client, le variabili di ambiente del server, gli header e il payload della richiesta HTTP
  - Un'istanza di **`ServletResponse`** viene passata alla servlet quando si richiede di restituire un contenuto da inviare al client; i metodi di questa classe permettono di scrivere su uno *stream* che verrà poi indirizzato al client, modificare gli header della risposta HTTP, ecc.

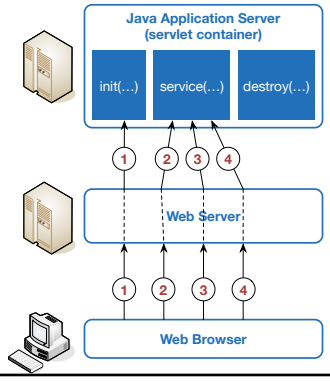
Ciclo di vita di una Servlet Java

- Il ***ciclo di vita*** di una servlet è scandito da una serie di chiamate, effettuate dal container, a particolari metodi dell'interfaccia Servlet
  1. **Inizializzazione**: quando il container carica la servlet, chiama il suo metodo **`init`**. Tipicamente questo metodo viene usato per stabilire connessioni a database e preparare il contesto per le richieste successive. A seconda dell'impostazione del contesto e/o del contenitore, la servlet può essere caricata immediatamente all'avvio del server, ad ogni richiesta, ecc.
  2. **Servizio**. Le richieste dei client vengono gestite dal container tramite chiamate al metodo service. Richieste concorrenti corrispondono a esecuzioni di questo metodo in ***thread*** distinti. Il metodo service riceve le richieste dell'utente sotto forma di una **`ServletRequest`** e invia la risposta tramite una **`ServletResponse`**
  3. **Finalizzazione**. Quando il container decide di rimuovere la servlet, chiama il suo metodo **`destroy`**. Quest'ultimo viene solitamente utilizzato per chiudere connessioni a database, o scaricare altre risorse persistenti attivate dal metodo `init`.
     È necessario sovrascrivere il metodo `destroy` solo se esistono effettivamente operazioni da fare prima della distruzione della servlet
- La classe **`HttpServlet`** specializza questo sistema per la comunicazione HTTP, fornendo in particolare due metodi: **`doGet`** e **`doPost`**, corrispondenti alle due request più comuni in HTTP. Il metodo **`service`** delle classe **`HTTPServlet`** provvede automaticamente a smistare le richieste al metodo opportuno

Ciclo di vita di una Servlet Java

- Il Java Application Server invoca alcuni metodi standard implementati dalla servlet in alcuni momenti specifici del suo ciclo di vita:
  - **`init(...)`**: il metodo viene invocato una sola volta quando l'applicazione Java viene avviata per la prima volta (inizializzazione);
  - **`service(...)`**: il metodo viene invocato ogni volta che il Java Application Server riceve una richiesta dal client per la URL corrispondente alla servlet;
  - **`doGet(...)`**: è il metodo utilizzato per gestire la richiesta HTTP mediante i comandi GET e HEAD;
  - **`doPost(...)`**: è il metodo utilizzato per gestire la richiesta HTTP mediante il comando POST;
  - **`destroy(...)`**: è il metodo invocato una sola volta dal Java Application Server quando viene terminata la web application; conclude il ciclo di vita della servlet





La classe HttpServlet

- Una servlet estende la classe `javax.servlet.http.HttpServlet`
- Per gestire le richieste HTTP, si sovrascrivono opportunamente i metodi corrispondenti: in questo esempio, il metodo **`doGet`** verrà chiamato per gestire le richieste HTTP GET.

```
package org.pippo.project;

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

public class pippo extends HttpServlet {
  public String getServletInfo() {
    return "Servlet di esempio, versione 1.0";
  }
  public void doGet(HttpServletRequest in, HttpServletResponse out) {
    //...
  }
}
```

Inizializzazione del Servlet

- Il metodo **`init`**, dopo aver chiamato lo stesso metodo della classe superiore, può procedere con l'inizializzazione della servlet
- Se ci sono problemi di inizializzazione, viene generata una **`ServletException`**

```
package org.pippo.project;

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

public class pippo extends HttpServlet {

  private int parameter1;
  public void init(ServletConfig c) throws ServletException {
    super.init(c);
    parameter1 = 1;
  }

  public String getServletInfo() {
    return "Servlet di esempio, versione 1.0";
  }

  public void doGet(HttpServletRequest in, HttpServletResponse out) {
    //...
  }
}
```

Acquisire la richiesta del client

- I metodi **`doGet`** e **`doPost`** possono leggere i parametri interpretati della richiesta tramite **`getParameter()`** e **`getParameterValues()`**
- `doGet` può leggere la stringa di query in maniera *raw* chiamando `getQueryString()`
- `doPost` può leggere il *payload* del messaggio (HTTP request) in maniera *raw* tramite gli *stream* restituiti da
  - `getReader()` (testuale) e
  - `getInputStream()` (binario)

```
package org.pippo.project;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class pippo extends HttpServlet {
  ...
  public void doGet(HttpServletRequest in, HttpServletResponse out) {
    String nome = in.getParameter("nominativo");
  }

  public void doPost(HttpServletRequest in, HttpServletResponse out) {
    try {
      Reader r = in.getReader();
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
  ...
}
```

Inviare la risposta al client

- L'oggetto **`HttpServletResponse`** fornito a tutti i metodi di servizio permette di costruire la risposta da inviare al client
  - Il metodo **`setContentType`**(*String*) permette di dichiarare il tipo MIME restituito dalla servlet (ad esempio "`text/html`")
  - Il metodo **`setHeader`**(*String*,*String*) permette di aggiungere *headers* alla risposta
  - Il metodo **`sendRedirect`**(*String*) permette di reindirizzare il browser verso una nuova URL
  - I metodi **`getWriter()`** e **`getOutputStream()`** permettono di aprire un canale, testuale o binario, su cui scrivere il contenuto della risposta; vanno chiamati dopo aver (eventualmente) impostato il *content type* e gli altri *header*
  - Altri metodi permettono di gestire, ad esempio, i cookie

```
public void doGet(HttpServletRequest in, HttpServletResponse out) {
 String nome = in.getParameter("nominativo");
 out.setContentType("text/html");
 try {
  Writer w = out.getWriter();
  w.write("<html><body><p>Ciao " + nome + "</p></body></html>");
 } catch(Exception e) {
  e.printStackTrace();
 }
}
```

Java Server Pages

- Le **Java Server Pages (JSP)** semplificano il processo di costruzione di siti web permettendo l'inclusione di contenuti dinamici e l'interazione con le classi Java che incapsulano la logica di business che sta alla base di una web application
- Le JSP sono costituite da una **combinazione di testo e tag**:
  - il **testo** (e i tag HTML) rappresenta il contenuto informativo di una pagina web
  - i **tag JSP** invece partecipano alla costruzione delle pagine dinamiche producendo i contenuti mediante istruzioni di programma in linguaggio Java
- Quando una Java Server Page viene richiesta, il container JSP provvede a **convertirla in una servlet**, **compilarlo** e quindi **eseguire la servlet** per elaborare la richiesta HTTP
- La conversione avviene soltanto nei casi in cui il sorgente della pagina ha subito modifiche rispetto all'ultima conversione effettuata dal container
- In questo modo si riducono i tempi di elaborazione delle richieste e si ottiene un incremento generale delle prestazioni

Java Server Pages

I **tag JSP** si dividono in quattro tipologie:

1. **direttive**: servono a comunicare al container JSP informazioni sulla pagina utili per il processo di traduzione in un servlet
   <%@ ***tag*** *attributo1* =“*valore1*” *attributo2*=“*valore2*” …%>
   es.: `<%@ page language=“java” contentType=“text/html” %>`
   `<%@ include file=“...” %>`
   `<%@ taglib uri=“...” prefix=“...” %>`

2. **script**: includono delle istruzione nel linguaggio di *scripting* associato alla pagina (Java); in questo tipo di tag sono contenute le istruzioni che implementano le tipiche strutture algoritmiche di sequenza, condizione e iterazione, con l’uso di variabili, oggetti e metodi
   dichiarazioni: <%! … *dichiarazione di variabili e metodi per tutte le istanze della pagina*… %>
   espressioni: <%= *espressione in output* %>
   scriptlet: <% … *istruzioni in linguaggio Java* … %>

3. **commenti**: sono usati per descrivere la logica con cui è stata progettata la JSP
   commenti: <%-- *commento al codice JSP* --%>

4. **action**: supportano diversi comportamenti e possono trasferire il controllo ai vari componenti della web application, come servlet, Java beans o altre Java Server Pages
   <jsp:**tag** *attributo*=“*valore*” />

Java Server Pages

- Il motore JSP dell’Application Server mette implicitamente a disposizione del programmatore degli **oggetti Java** che possono essere utilizzati nelle pagine JSP senza la necessità di dichiararli

- **`request`**: è l’oggetto che contiene le intestazioni della richiesta HTTP e quindi anche i parametri passati alla pagina JSP attraverso i metodi GET o POST
  - Il metodo `getParameter(“X”)` dell’oggetto request consente di acquisire il parametro fornito alla pagina JSP identificato con il nome “X” (ad esempio il valore di un campo di una form)
  - Es.: `<p>Ciao <%=request.getParameter(“nome”)%></p>`

- **`response`**: è l’oggetto che riceve l’output prodotto dalla pagina JSP e lo restituisce all’application server per l’invio a client
  - L’oggetto viene utilizzato implicitamente attraverso l’oggetto **`out`**
  - Es.: `<% out.println(“Importo:” + valore); %>`
  - oppure: `<%=valore%>`

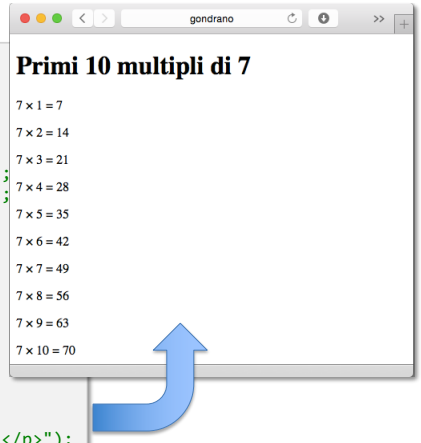
Java Server Pages: un esempio elementare

Applicazione web JSP per il calcolo dei primi *n* multipli di *k*


```
<!DOCTYPE HTML>

<html>
  <head>
    <title>Calcolo dei multipli</title>
  </head>
  <body>
    <h1>Calcolo dei multipli di un intero</h1>
    <p>Inserisci l'intero k e il numero n di
    multipli che desideri calcolare.</p>
    <form action="multipli.jsp" method="GET">
      <p>k: <input type="text" name="k" size="3">
      n: <input type="text" name="n" size="3">
      <input type="submit" value="Ok"></p>
    </form>
  </body>
</html>
```

Java Server Pages: un esempio elementare

Applicazione web JSP per il calcolo dei primi *n* multipli di *k*




```
<!DOCTYPE HTML>
<%--  multipli.jsp:
      calcola i primi n multipli di k --%>
<%
  int n, k, m, i;
  n = Integer.parseInt(request.getParameter("n"));
  k = Integer.parseInt(request.getParameter("k"));
%>
<html>
  <head>
    <title>Multipli di un intero</title>
  </head>
  <body>
    <h1>Primi <%=n%> multipli di <%=k%></h1>
    <%
    for (i=1; i<=n; i++) {
      m = k*i;
      out.println("<p>"+k+" &times; "+i+" = "+m+"</p>");
    }
    %>
  </body>
</html>
```

Programmazione "lato client" e Web 2.0

- Il modello architetturale e funzionale delle applicazioni web (JEE, CGI o realizzate con altre tecnologie e framework) è basato sul paradigma "**request/response**":
  - ogni risposta costruita dinamicamente dal server mediante un programma CGI o un programma Java a fronte di una richiesta del client costituisce una pagina della web application
  - L'utente può selezionare un semplice link o compilare una form estremamente complessa, costituita da decine o centinaia di campi di input, in ogni caso il server raccoglie tutti i dati della richiesta, elabora una risposta e la invia al client che deve eseguire il *rendering* di una pagina web completamente nuova e differente dalla pagina precedente
- Per quanto le pagine web siano gradevoli e basate su una rappresentazione grafica o un'impaginazione tipografica complessa delle informazioni, questo modello non differisce molto da quello utilizzato nelle maschere alfanumeriche tipiche delle applicazioni centralizzate su un host o su un mainframe
- Le applicazioni dotate di un'interfaccia utente grafica (le applicazioni per Microsoft Windows o per X11 in ambiente UNIX) ci hanno invece abituato ad una **diversa e più complessa interazione tra l'utente ed il programma**: *ad ogni interazione dell'utente con l'interfaccia corrisponde una reazione da parte del programma che si manifesta con una variazione di alcune parti del contenuto della schermata che stiamo utilizzando*

Programmazione "lato client" e Web 2.0

- Il cosiddetto "**Web 2.0**" introduce un livello di complessità superiore sulle web application per realizzare un'interazione con l'utente paragonabile a quella delle applicazioni client
- Con le tecnologie introdotte con il Web 2.0 si gettano le basi anche per la realizzazione di applicazioni per il mondo "***mobile***": non semplici applicazioni web fruibili mediante il web browser presente su *tablet* e *smartphone*, ma vere e proprie applicazioni che operano (anche / in parte) sul sistema operativo del *device mobile*
- Gli elementi che sono alla base di questo salto in avanti nella progettazione di applicazioni web based sono sintetizzati nei seguenti punti:
  - **capacità del web browser di eseguire del codice di un programma** in linguaggio Javascript
  - possibilità di **operare sugli elementi della pagina HTML da programma Javascript** caricato nella pagina stessa, indirizzando gli elementi presenti all'interno del DOM del documento, identificati con nomi univoci (Dynamic HTML)
  - possibilità di **stabilire un dialogo tra il client e il server** nell'ambito della stessa pagina attraverso procedure Javascript inserite nella pagina stessa (AJAX)
  - mediante l'uso di CSS combinato con Javascript si può ottenere la costruzione di interfacce utente **responsive**, ossia in grado di adattarsi alla dimensione e alle caratteristiche del device che le sta visualizzando all'utente

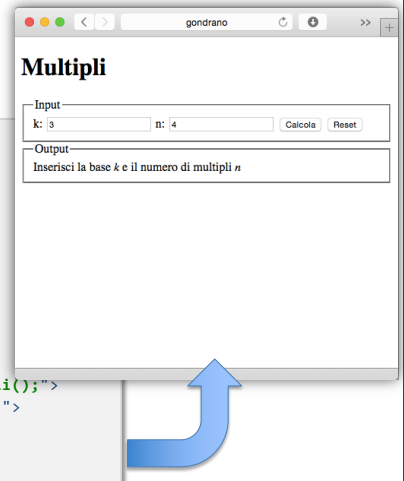
Javascript

- È un **linguaggio Object Oriented** la cui sintassi ricorda quella di Java
- È supportato da un interprete e da un ambiente di esecuzione presente in quasi tutti i web browser
- **Il codice di un programma Javascript può essere inserito in una pagina HTML**: nell'intestazione della pagina, in modalità "inline" all'interno dei tag della pagina, o in un file separato richiamato da un tag posto nell'intestazione della pagina
- Il browser crea un ambiente di esecuzione del programma Javascript con un ***namespace*** legato al DOM del documento HTML per collegare l'esecuzione delle funzioni Javascript agli eventi a cui sono sensibili gli oggetti (componenti del DOM) che compongono il documento stesso
- **L'onere di eseguire la procedura Javascript ricade sul client**: è la CPU del client che è impegnata nell'elaborazione e nel calcolo, è la memoria del client che viene impegnata per gestire le variabili, le strutture dati e gli oggetti utilizzati dal programma
- L'ambiente di esecuzione Javascript interno al browser impone un insieme di **vincoli per ragioni di sicurezza**: con il programma Javascript non è possibile utilizzare pienamente le risorse del client (es.: non è possibile scrivere un file sul client o accedere ad un DBMS sul client)

Javascript: un esempio elementare

Applicazione web Javascript per il calcolo dei primi *n* multipli di *k*

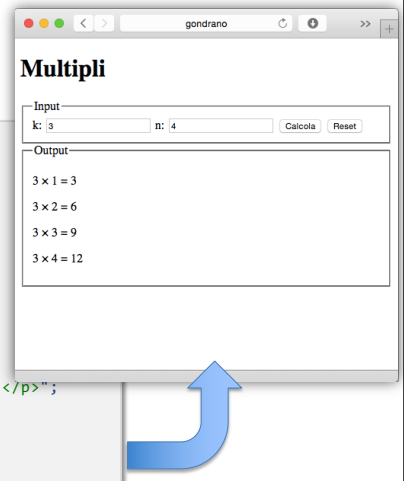
```
<!DOCTYPE HTML>
<html>
<head>
  <title>Prova in Javascript</title>
  <script language="Javascript" src="multipli.js">
  </script>
</head>
<body>
  <h1>Multipli</h1>
  <fieldset>
    <legend>Input</legend>
    k: <input id="k"> n: <input id="n">
    <input type="button" value="Calcola" onClick="multipli();">
    <input type="button" value="Reset" onClick="azzera();">
  </fieldset>
  <fieldset>
    <legend>Output</legend>
    <div id="output">Inserisci la base <var>k</var> e il numero
    di multipli <var>n</var></div>
  </fieldset>
</body>
</html>
```



Javascript: un esempio elementare

Applicazione web Javascript per il calcolo dei primi *n* multipli di *k*

```
function multipli() {
  var i, m, n, k, s;
  n = document.getElementById("n").value;
  k = document.getElementById("k").value;
  if (!n) {
    alert("ERRORE: inserire un valore per n!");
  } else if (!k) {
    alert("ERRORE: inserire un valore per k!");
  } else {
    s = "";
    for (i=1; i<=n; i++) {
      m = k*i;
      s = s + "<p>" + k + " &times; " + i + " = " + m + "</p>";
    }
    document.getElementById("output").innerHTML = s;
  }
}

function azzera() {
  document.getElementById("output").innerHTML = "Inserisci
  la base <var>k</var> e il numero di multipli <var>n</var>";
}
```



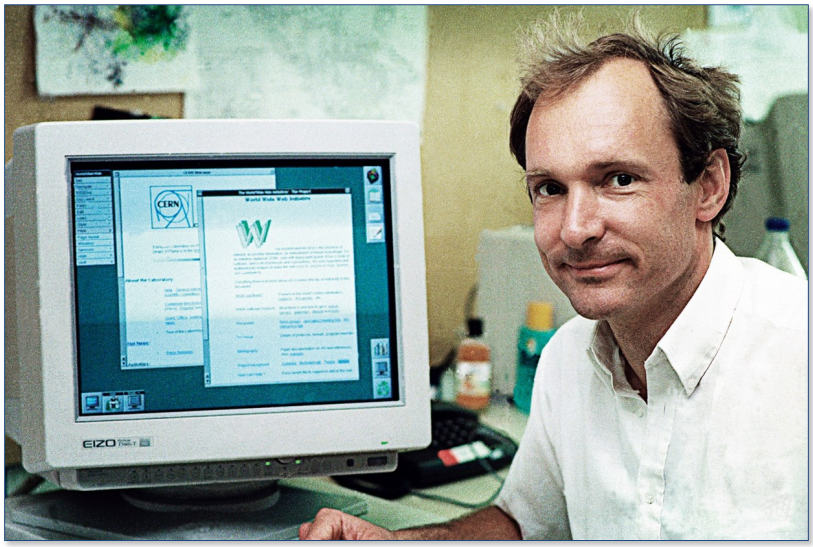
AJAX e Dynamic HTML

- **AJAX** (*Asynchronous JavaScript and XML*), è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive
- Lo sviluppo di applicazioni web con AJAX si basa su uno scambio di dati in background fra web browser e web server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento della pagina stessa da parte dell'utente
- AJAX è **asincrono**: i dati extra sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente
- Il formato per lo scambio di dati tra il client e il server è spesso **JSON** (*JavaScript Object Notation*), che offre un formato standard molto semplice per rappresentare sequenze di coppie di tipo "chiave:valore", in cui il "valore" può a sua volta essere una collezione di coppie "chiave:valore"
- Tipicamente le funzioni richiamate sono scritte con il linguaggio JavaScript e sono presenti nella pagina HTML

- **JQuery** è oggi la più famosa libreria Javascript che implementa AJAX, ma anche molti altri aspetti del Web 2.0; è una libreria di funzioni progettate per semplificare la costruzione di pagine web dinamiche, con una gestione degli eventi molto efficace
- È supportata da numerosissimi framework di sviluppo (Microsoft .net, JEE, ecc.)

Riferimenti

1. W3 Consortium: http://www.w3.org
2. W3Schools, "*HTML5 Tutorial*": http://www.w3schools.com/html/default.asp
3. W3Schools, "*CSS Tutorial*": http://www.w3schools.com/css/default.asp
4. Tim Berners-Lee "*Style Guide for online hypertext*": http://www.w3.org/Provider/Style/
5. Kevin Werbach, "*The Bare Bones Guide to HTML*": http://werbach.com/barebones/ (in italiano: http://werbach.com/barebones/it/it_barebone.html)
6. Judy Bishop, "*Java Gently – Corso Introduttivo*", Seconda Edizione, Addison Wesley
7. Marco Bertacca, Andrea Guidi, "*Introduzione a Java*", Seconda Edizione, McGraw-Hill
8. Larry Wall, "*Programming Perl*", O'Reilly
9. Altre risorse disponibili su Internet:
   - http://www.perl.org
   - http://www.perl.com
   - http://cgi-lib.berkeley.edu/ (CGI-lib home page, presso University of California at Berkeley)
   - http://www.aquilante.net/perl/



Tim Berners-Lee

Sir Timothy John Berners-Lee è uno degli inventori del World Wide Web, autore del primo server HTTP e del primo client per accedere a risorse distribuite in rete mediante il protocollo HTTP; nella foto è ritratto negli anni '90 accanto al monitor della sua workstation NeXT al CERN di Ginevra, che successivamente ha abbandonato per trasferirsi al prestigioso Laboratorio di Computer Science del MIT di Boston