

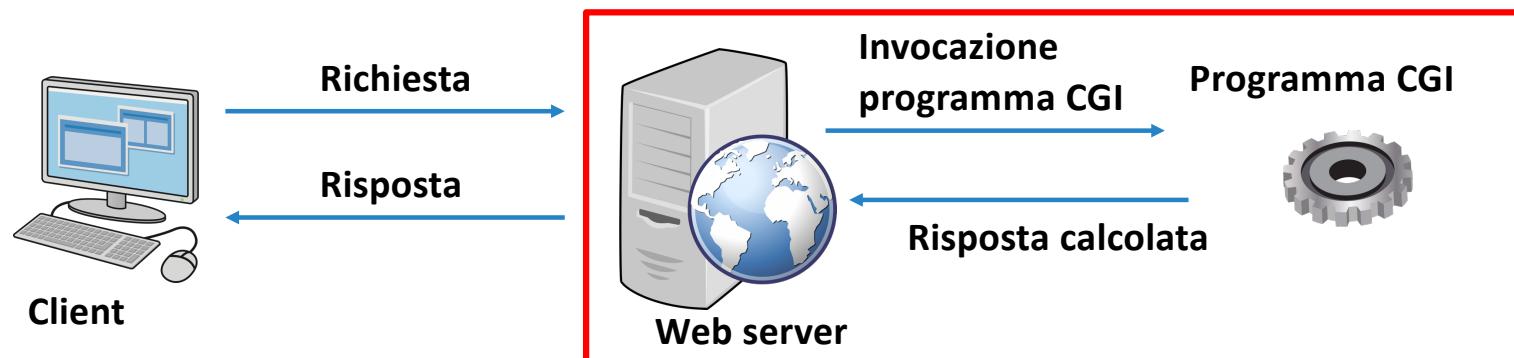
# Servlet e JSP

---

PROF. DIOMAIUTA CRESCENZO

# Common Gateway Interface

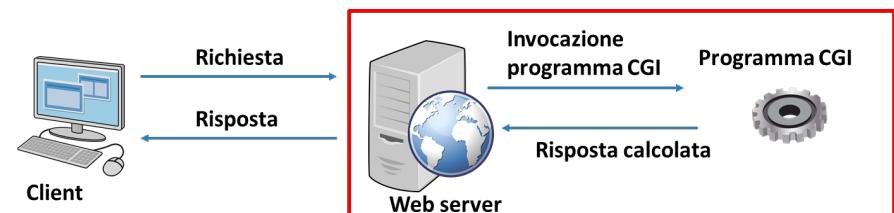
- Common Gateway Interface (CGI) è il primo e più semplice standard architettonico per la creazione dinamica di pagine web
- Il server WWW può richiamare un eseguibile scritto in un qualsiasi linguaggio tradizionale compilato od interpretato



# Common Gateway Interface

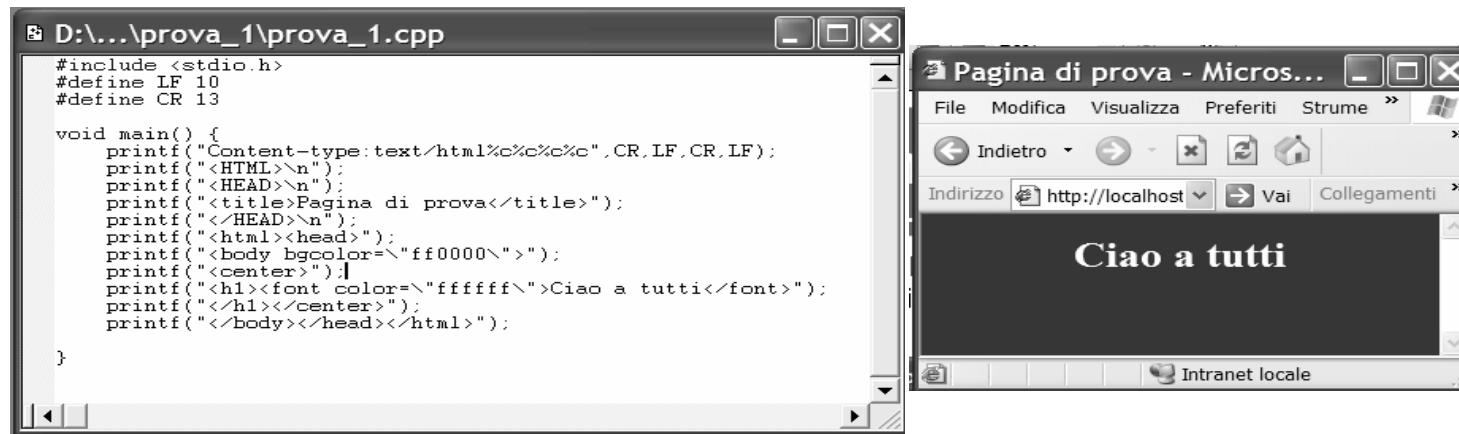
- Il browser invia una richiesta al Web server. L'URL di questa richiesta identifica lo script CGI (può avere anche dei parametri). I parametri inseriti nella parte finale dell'URL sono detti *query string*
- Il Web server riconosce che l'URL fa riferimento ad un programma eseguibile, salva le variabili in memoria (i parametri) e lancia il programma
- Una volta terminato lo script CGI, il Web Server usa la pagina HTML prodotta dinamicamente come oggetto della risposta HTTP da inviare al browser
- Il browser riceve la pagina HTML

`http://www.server.com/nomeScript?par1=val1&par2=val2`



# Common Gateway Interface

---



D:\...\prova\_1\prova\_1.cpp

```
#include <stdio.h>
#define LF 10
#define CR 13

void main() {
    printf("Content-type:text/html%c%c%c%c",CR,LF,CR,LF);
    printf("<HTML>\n");
    printf("<HEAD>\n");
    printf("<title>Pagina di prova</title>");
    printf("</HEAD>\n");
    printf("<html><head>");
    printf("<body bgcolor=\"ff0000\">");
    printf("<center>");
    printf("<h1><font color=\"ffffff\">Ciao a tutti</font>");
    printf("</h1></center>");
    printf("</body></head></html>");
}


```

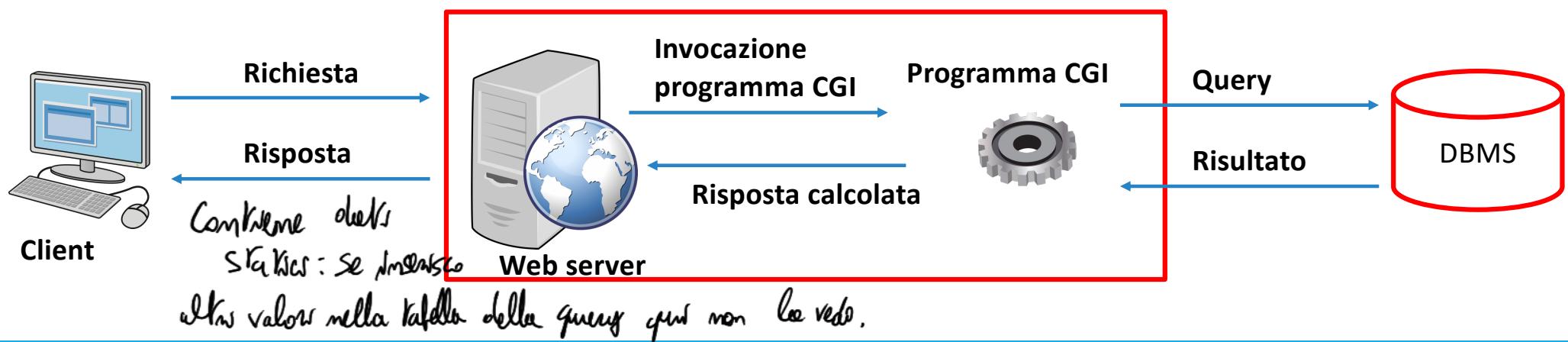
Pagina di prova - Microsoft Internet Explorer

Indirizzo: http://localhost Vai Collegamenti

Ciao a tutti

# Common Gateway Interface

- È possibile utilizzare anche un'interrogazione con una base dati utilizzando uno script CGI



# CGI: Limiti

---

- Il web server genera un nuovo processo cgi ad ogni richiesta
- Il processo viene terminato alla fine dell'elaborazione della risposta
- Elevato sovraccarico di esecuzione per la creazione e distruzione di processi
- Impossibile:
  - Tenere informazioni sulla sessione dell'utente
  - Tenere allocate risorse condivise tra più richieste o più utenti (es. pool di connessioni a database)

# Il Web dinamico attuale

---

- Forte interazione con l'utente
- Contenuti attivi lato server e lato client
- Diverse tecnologie disponibili:
  - Lato client
  - Lato server
- Architetture multi-tier e distribuite

↳ multilivello

# Lato client

---

- Offrono interazione mediante:
  - codice precompilato che viene scaricato ed eseguito sulla macchina client (Microsoft ActiveX, Applet Java, Macromedia Flash)  
*→ compilato mentre lo stai eseguendo*
  - codice di scripting, visibile all'interno della pagina web ed interpretato dal browser (VBScript, JavaScript) o eventualmente compilato Just In Time-JIT (JavaScript, ASP.NET)  
*↳ Basta fare click e browser interpreta su parte di codice JS*
- Richiedono estensioni lato client (plug-in per il browser, Java Virtual Machine (JVM), interpreti di script)

# Tecnologie lato client ad oggetti precompilati

## • Microsoft ActiveX

- Tecnologia proprietaria Microsoft
- Supportata solo su Microsoft Internet Explorer
- Permette di scrivere controlli in Visual C++ o Visual Basic, gestiti con VBScript

→ Possono richiedere Framework (chiamate oggetto) da usare

## • Applet JAVA (im dursus: pesante e difficile per smartphone. Oggi molto ha presupposto recompilazione)

- Tecnologia a standard aperto sviluppata da Sun Microsystem
- Supportata tramite JVM che interpreta il ByteCode, un codice compilato portabile
- Supporta nativamente ambienti multipiattaforma e scalabili

## • Macromedia Flash (Non usata)

- Tecnologia proprietaria per lo sviluppo di pagine molto leggere tramite grafica vettoriale
- Supportata tramite un plugin
- Linguaggio orientato esclusivamente allo sviluppo di presentazioni multimediali

# Tecnologie lato client con script

---

- Microsoft VBScript

↳ Pensata per app. desktop

- Tecnologia proprietaria derivata da Microsoft Visual Basic
- Supportata esclusivamente da Microsoft Internet Explorer , per creare piccole app complete
- Permette di controllare sia il browser sia altre applicazioni, può essere impiegato per controllare oggetti OLE e applet Java contenuti in una pagina web

# Tecnologie lato client con script

---

- **JavaScript**

- Linguaggio di programmazione interpretato e leggero, creato dalla Netscape
- Supportato anche da Microsoft Internet Explorer ma nella variante Microsoft JScript
- Permette il controllo dell'aspetto e del contenuto del documento e del browser e l'interazione con l'utente e con i moduli

{anche versioni bundle su JS ma con diff. struttura: framework base) su JS

# Tecnologie lato server

---

- Offrono interazione mediante codice che viene eseguito sulla macchina server (Microsoft Active Server Pages (ASP), Java Servlet, Java Server Pages (JSP), applicazioni CGI)

*Ci interessano soluz. standard non propriez.*

*- Il codice può essere interpretato (ASP), precompilato (Servlet), o compilato JIT (JSP) e produce una pagina HTML che viene inviata al client, senza traccia del codice originario*

- Richiedono estensioni lato server (moduli per il server web che gestiscano la tecnologia e le funzionalità richieste)

# Tecnologie lato server

---

- Microsoft Active Server Pages (ASP)
  - Tecnologia proprietaria Microsoft
  - Supportata solo su Microsoft Personal Web Server 4 e superiori, Microsoft Internet Information Server (IIS), Microsoft Internet Information Server e Advanced Server
  - Permette di scrivere script in VBScript - lato server che accedano ad oggetti ASP e ADO, direttamente nella pagina web, detta pagina ASP (.asp)



- Java Servlet
  - Tecnologia standard progettata da Sun Microsystems
  - Supportata tramite le estensioni Java dei più diffusi server web o tramite appositi Application Server
  - Permette di scrivere classi Java che estendono le funzionalità del server web e generano pagine HTML che vengono inviate al client

# Tecnologie lato server

---



- **Java Server Pages (JSP)**

- Tecnologia standard progettata da Sun Microsystem
- Supportata tramite un container JSP, disponibile per i più diffusi server web
- E' usato scrivendo pagine miste in HTML e JavaScript, interpretate alla richiesta generando pagine HTML pure che vengono inviate al client in risposta
- L'HTML è separato dal codice.

# Tecnologie lato server

---

- CGI

- Protocollo di interazione di programmi esterni con un server web
  - Supportata tramite reindirizzamento delle richieste del client al programma CGI da parte del server web
  - Permette al server web di invocare programmi in C, Perl ed altri linguaggi in risposta a richieste di pagine CGI (.cgi) da parte del client, generando pagine HTML che vengono inviate in risposta



# Tecnologie lato server

---

## • PHP

- Linguaggio open source di scripting lato server, orientato prevalentemente all'interfacciamento con basi di dati
- Supportato tramite un modulo del server web che funge da interprete
- Permette di scrivere pagine miste in HTML e PHP (.php, .php3, .phtml), interpretate alla richiesta generando pagine HTML pure che vengono inviate al client in risposta

Posso sfruttare il tag <hr>

# Tecnologie lato server: Confronto



- CGI: è efficace ma non efficiente (un processo separato per applicazione) e richiede un programmatore esperto
- ASP: è efficiente (ogni pagina è un thread del server web) ma è proprietario e supportato solo da Microsoft
- Java Servlet: è efficiente (è compilato e ogni pagina è un thread del server web ) ma richiede un programmatore Java
- JSP: è efficiente (è compilato alla prima invocazione e ogni pagina è un thread del server web ) ma è interpretato alla prima invocazione e richiede anche un programmatore JAVA
- PHP: è facile e ha funzionalità di alto livello, ma è interpretato (anche se è integrato in maniera nativa in Apache)

# Tecnologie lato server: Confronto

---

- I PRO di JSP *Java Server Pages*

- Completamente multi-piattaforma
- Supportato da numerosi Application Server
- Utilizzo di Java come linguaggio per gli scriptlet
- Aderenza alla piattaforma J2EE
- Le pagine JSP sono compilate, garantendo generalmente delle ottime prestazioni

- I CONTRO di JSP

- Di non immediata comprensione per chi non è pratico di Java
- Latenza di risposta alla prima richiesta che si inoltra verso una pagina
- Attualmente, supporta solo Java come linguaggio per gli scriptlet

# Tecnologie lato server: Confronto

---

- I PRO di PHP

- Conosciutissimo ed usatissimo
- Possiede una libreria di funzioni completissima
- Integrabile in numerosi Web server

- I CONTRO di PHP

- Poco orientato agli oggetti: le librerie standard hanno aspetto procedurale
- Caratteristiche importanti sono supportate solo dalle versioni più recenti
- Linguaggio di scripting un po' criptico

# Tecnologie lato server: Confronto

---

- I PRO di ASP (Active Server Pages)

- Semplice da apprendere e facile da utilizzare
- Permette l'uso di diversi linguaggi di scripting
- L'accesso ai database è davvero semplice

- I CONTRO di ASP

- Supportato principalmente solo dai web server Microsoft, su piattaforma prettamente Windows
- Completamente dipendente dagli ActiveX installati nel sistema
- Difficilmente espandibile

# Servlet

---

- Un'alternativa a CGI può essere il linguaggio Java per realizzare le Servlet, che sono delle classi con affinità particolari alle applet Java.
- La differenza principale tra CGI e Servlet è che gli script CGI sono eseguiti dal S.O., mentre le Servlet sono eseguite dalla JVM integrata nel Web Server
- Le Servlet possono essere utilizzate qualunque sia il protocollo di interazione client-server espletato dal server (es: sia HTTP che FTP)

Amche HTTPS/FTPS

# Servlet: creare servlet: avere diversi con dei metodi particolari

---

- **Efficienti**

- Con l'approccio CGI-BIN un nuovo programma è avviato per ogni richiesta. Con le servlets ogni nuova richiesta è gestita tramite threads.

- **Convenienti (Sviluppo in Java)**

- Esiste un significativa infrastruttura per leggere e decodificare forms, mantenere i dati di una sessione, gestire cookies, gestire headers HTTP, interagire con DB, etc.
- Possibilità di riutilizzare opportunamente l'enorme quantità di utilities JAVA presenti nel mondo freeware/opensource

- **Portabili**

- Sono scritte in Java rispettando uno standard ben preciso.

- **Sicure** costituzione di un protocollo obbligatorio che delle richieste

- Non sono eseguibili a se stanti, per cui non sono utilizzabili per attaccare un server.

- **Economiche**

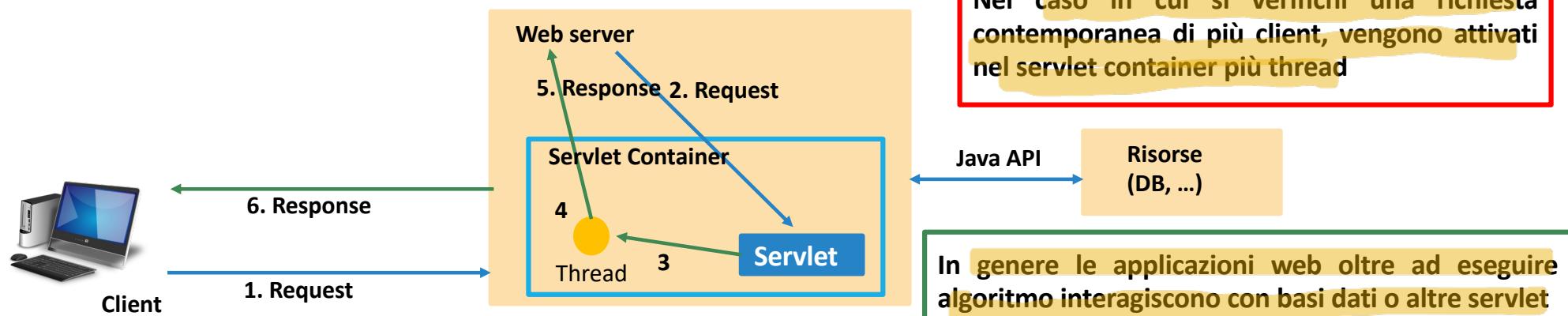
- Tutta l'infrastruttura necessaria esiste (anche) nel mondo open source.

# Servlet: Struttura

Un oggetto servlet gestisce una richiesta da un client.

Una Servlet è un componente software scritto in Java, gestito da un "container" che produce contenuto web dinamico

- Ogni istanza di una Servlet è un oggetto Java caricato ed eseguito dal Web Server.
- Il controllo di una Servlet avviene ad opera di un container (il Web server stesso) che regola il ciclo di vita di una Servlet



# Servlet: Container *Open Source*

---

- Il servlet container open source più diffuso e utilizzato è **Tomcat**, del gruppo **"Apache"**
- Tomcat è un servlet container scritto in Java che si integra con i più diffusi web server
- L'esecuzione delle servlet richiede inoltre che sulla macchina server sia installata una versione di Java (Java Runtime Environment (**JRE**) per poter funzionare, almeno la versione 1.5.0 per Tomcat 5.0)

# Realizzazione di una Servlet

- Nella sua forma più generale, le servlet usate nel Web sono estensioni della classe `javax.servlet.http.HttpServlet` che a sua volta implementa l'interfaccia `javax.servlet.Servlet`.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletHelloWWW extends HttpServlet {
```

**Indipendenti dal protocollo di comunicazione**  
**Specifiche per il protocollo HTTP**

- Il package `javax.servlet` è la base per le API Servlet e contiene la definizione dell'interfaccia `servlet` e le classi utili alla comunicazione fra client e server
- Il package `javax.servlet.http` fornisce classi che estendono le funzionalità base di una `servlet`, adottando le caratteristiche del protocollo HTTP come la gestione dei metodi `GET` e `POST` o degli header HTTP
- Tutte le servlet estendono `HttpServlet` o un'altra classe sono classi che implementa l'interfaccia `servlet`

Naturalmente è necessaria la libreria `javax` (file `javax.servlet.jar`)

# Classe HTTPServlet

---

- La classe **HTTPServlet** implementa **service()** e invoca un insieme di metodi, tra i quali quelli necessari per servire le richieste dal Web
- I due metodi principali sono:
  - **doGet()**: processa le richieste di tipo GET effettuate dalla pagina HTML
  - **doPost()**: processa le richieste di tipo POST effettuate dalla pagina HTML
- Quando una servlet accetta una richiesta da un client questi due metodi ricevono due oggetti istanza delle classi **HttpServletRequest** e **HttpServletResponse**

# HttpServletRequest

---

- L'oggetto **HttpServletRequest** estende **ServletRequest**, viene passato da `service()` e contiene la richiesta del client, in particolare permette di ottenere i parametri inviati dal client, il riferimento alla sessione utente e il flusso dei dati inviati dal client oltre che riconoscere l'utente autenticato.
- I metodi più utilizzati con questo oggetto sono:
  - **getRequestURI()**: restituisce l'URI richiesto;
  - **getMethod()**: fornisce il metodo HTTP utilizzato per re-inoltrare la richiesta;
  - **getParameter(String)**: consente di accedere per nome ai parametri contenuti nella query string del client; ↳ *Nome dato nel nome delle form*
  - **getInputStream()** e **getReader()**: consentono di creare gli stream di input e la possibilità quindi di ricevere i dati della richiesta HTTP.

# HTTPServletResponse

---

- L'oggetto **HttpServletResponse** estende **ServletResponse**, viene passato da **service()** e contiene la risposta per il client. In particolare permette di inviare dati al client in formato HTML oppure come flusso binario, di inviare codici di errore e codici di controllo nell'intestazione della risposta HTTP per controllare il comportamento del browser.
- I metodi per creare gli stream di output e la possibilità quindi di inviare i dati della risposta sono:
  - **getOutputStream()**: per l'invio di dati in forma binaria; *Per inviare risposte*
  - **getWriter()**: per l'invio attraverso il canale System.out. *Per scrivere la risposta come HTML per esempio*
- **HttpServletResponse** fornisce anche dei metodi per accedere e fissare gli header della risposta, come il **setContentType()**, definito prima di poter inviare il body al client.

# Servlet: Ciclo di vita

---

- il **ciclo di vita di una servlet può essere riassunto in tre fasi:**

***init() => service() => destroy()***

- Il **container svolge le seguenti operazioni:**
  - **carica una Servlet la prima volta che viene richiesta;**
  - **richiama il metodo init() della Servlet (che provvede all'inizializzazione della servlet);**
  - **gestisce le varie richieste richiamando il metodo service() delle Servlet in una nuova thread;**
  - **alla chiusura richiama il metodo destroy()** *Non lascia un thread attivo*

# Deployment di una applicazione Web

- Per poter far funzionare una servlet è necessario effettuare dei passaggi previsti dal deployment:
  - Definire il run time environment di una Web Application (WAP)
  - Posizionare tutti i file della applicazione nelle opportuna directory
  - Mappare le URL sui servlet con file di configurazione
  - Definire delle impostazioni di default di un'applicazione (per esempio la pagina di benvenuto e la pagina di errore)
  - Configurare i vincoli di sicurezza dell'applicazione

PASSI CHE FAREMO

Maggior parte di queste cose fanno da un file XML

Ogni WebApplication deve vivere nella cartella WEBAPPS

# Deployment di una applicazione Web

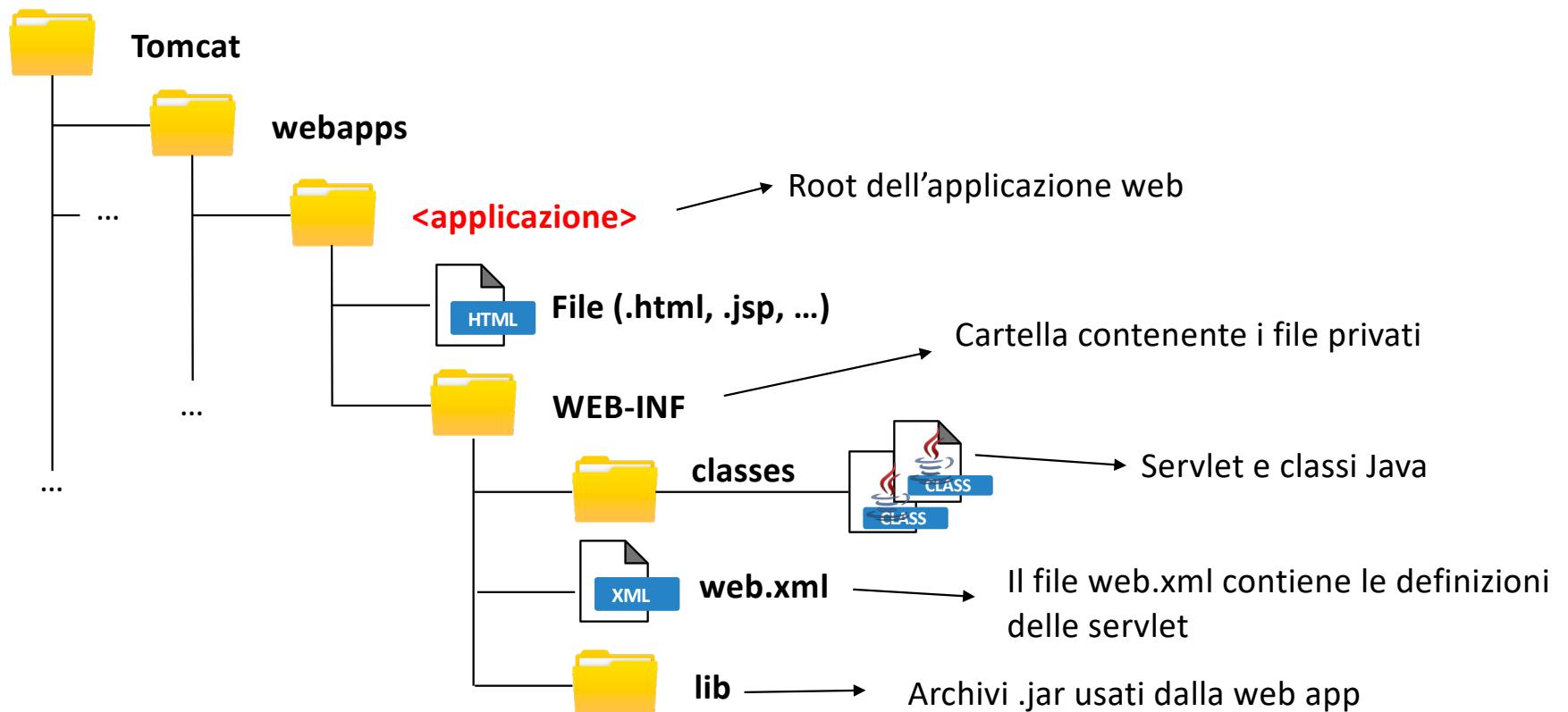
- Tomcat come Web server richiede per ogni WAP una particolare strutturazione delle directory collocate a partire dalla sua cartella **WEBAPPS**:
  - Nella root directory della applicazione vengono copiati i file:
    - \*.html, \*.jsp, ...
  - viene creata una subdirectory **WEB-INF** con due sotto cartelle **classes** e **lib**:
    - nella **WEB-INF** si memorizza il file **web.xml** (**Deployment Descriptor**), che rappresenta il descrittore dell'applicazione. E' un file XML che descrive le servlets e gli altri componenti dell'applicazione;
    - in **WEB-INF/classes** i file **\*.class** delle servlet (bytecode): Questa directory contiene le classi java organizzati in packages *scegliamo noi*
    - in **WEB-INF/lib** eventuali file di libreria **\*.jar**: Contiene i files JAR che contengono classi JAVA necessarie all'applicazione.

classi servlet compilate  
library uscite (\*.jar)

\* Descrizione di sviluppo della WAP.

I nuovi server accettano applicazioni nel formato **Web Application Archive (WAR)**.

# Deployment di una applicazione Web



# Deployment di una applicazione Web

- Una volta replicata la struttura come richiesto da Tomcat, per mandare in esecuzione l'applicazione ci sono due modalità:
  1. digitare nel browser il nome della pagina HTML di inizio della applicazione che al suo interno richiama la servlet:

*digitare index per direttamente*

```
http://localhost:8080/<applicazione>/index.html
```

2. digitare nel browser il nome della servlet connessa alla applicazione:

```
http://localhost:8080/<applicazione>/servlet/<nome_servlet.class>
```

*Nome della servlet connessa all'applicazione*

# Il Context XML descriptor o Deployment descriptor

- Il file **web.xml (deployment descriptor)** serve per **configurare l'applicazione Web** così che **Tomcat possa gestire le richieste provenienti dai client**.
- Contiene l'elenco dei servlet e per ogni servlet permette di definire una serie di **parametri come coppie nome-valore** che costituiscono la descrizione del Context associato a una Web application.
- Nel file web.xml vengono **specificati diversi parametri** come:
  - il **nome della classe che definisce la servlet** *classe principale da richiamare*
  - i **percorsi che causano l'invocazione della servlet** da parte del Container
  - una serie di parametri di configurazione (**coppie nome-valore**)
  - **Contiene anche la mappatura fra URL e Servlet che compongono l'applicazione**  
*↳ dove dove quelle servlet in quale URL si trova*

# Il Context XML descriptor o Deployment descriptor

The diagram illustrates the structure of the web.xml configuration file. It starts with the root element <web-app>. A curly brace on the left, labeled 'Punte genitiva', covers the entire <web-app> element and its child elements: <display-name>, <description>, <servlet>, and <servlet-mapping>. The <display-name> and <description> elements are highlighted in yellow. Handwritten notes explain their functions: '<display-name> specifica un nome che viene utilizzato dall'amministratore del server' and '<description> specifica una descrizione della applicazione che viene visualizzata nel pannello del manager'. The <servlet> and <servlet-mapping> elements are also highlighted in yellow. Handwritten notes explain their functions: '<servlet-name> definisce un nome per la servlet all'interno di questo file', '<description> fornisce una descrizione di questa servlet', '<servlet-class> specifica il nome completo della classe servlet compilata', and '<servlet-mapping> specifica l'associazione tra il nome dato alla servlet nell'interno del file web.xml (<servlet-name>) e l'<url-pattern> con cui la servlet potrà essere invocata'. A curly brace on the right, labeled 'Mapping delle servlet', covers the <servlet-mapping> element and its child elements: <servlet-name> and <url-pattern>.

```
<web-app>
    <!-- definizione della Servlet -->
    <servlet>
        <servlet-name>ServletHelloWWW</servlet-name>
        <description> Richiama l'esecuzione della classe ServletHelloWWW </description>
        <servlet-class>ServletHelloWWW</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ServletHelloWWW</servlet-name>
        <url-pattern>/servlets/servlet/ServletHelloWWW</url-pattern>
    </servlet-mapping>
</web-app>
```

*Se me ho detto prima,  
sono stato molto più veloce*

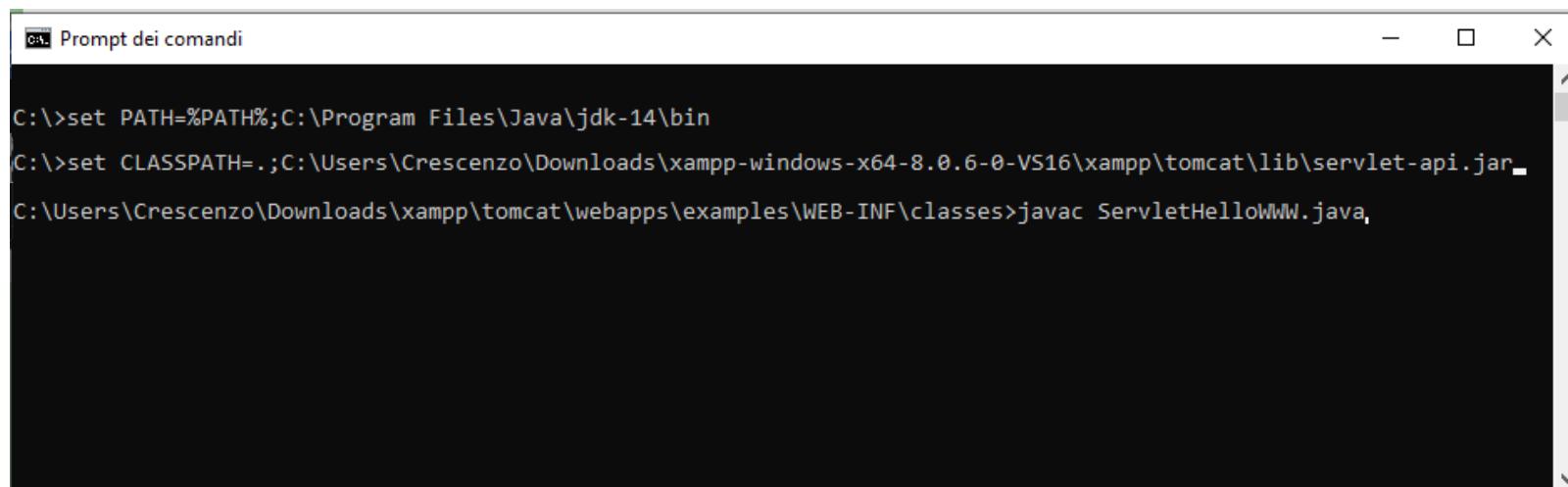
# Servlet: Esempio

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletHelloWWW extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML><HEAD>");
        out.println("<TITLE>Hello World </TITLE></HEAD>");
        out.println("<BODY bgcolor=\"#FF0000\"> <H1> Hello World!!!!</H1></BODY></HTML>");
    }
}
```

metodo per stampare a "video" a Ray. È sempre una risposta qualsiasi a Ray varrà interpretata come risposta deve essere interpretata dal browser

NOTA: Se modifichi servlet, allora debvi ricompilare il codice e riavviare server.

## Servlet: Compilazione



```
C:\>set PATH=%PATH%;C:\Program Files\Java\jdk-14\bin  
C:\>set CLASSPATH=.;C:\Users\Crescenzo\Downloads\xampp-windows-x64-8.0.6-0-VS16\xampp\tomcat\lib\servlet-api.jar  
C:\Users\Crescenzo\Downloads\xampp\tomcat\webapps\examples\WEB-INF\classes>javac ServletHelloWWW.java,
```

Importante: sbloccare l'invoker nel file, nel caso in cui le servlet siano in una directory differente

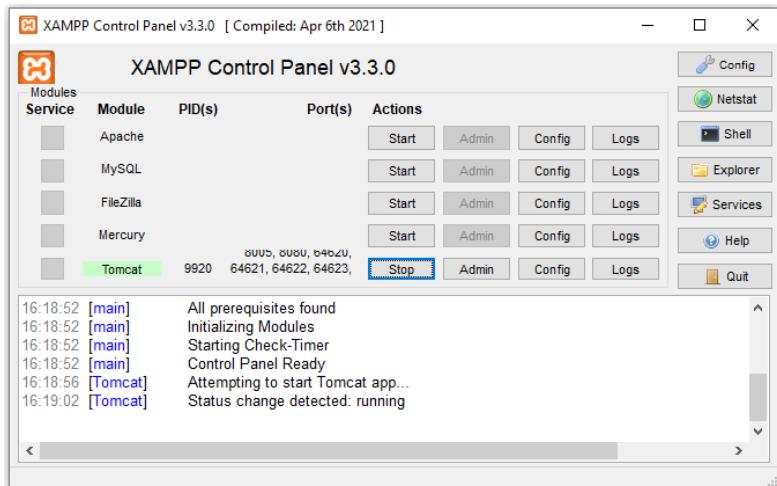
**C:\Programmi\Apache Software Foundation\Tomcat 5.5\conf\web.xml**

Copiare il .class nella dir

**C:\Programmi\Apache Software Foundation\Tomcat 5.5\webapps\examples\WEB-INF\classes**

Oppure copiare il file java e generarlo direttamente lì dentro il .class

# Servlet: Compilazione



The screenshot shows the Apache Tomcat 8.5.65 homepage. At the top, it says 'Apache Tomcat/8.5.65' and 'localhost:8080'. The main content area features a green banner with the text 'If you're seeing this, you've successfully installed Tomcat. Congratulations!' and a cartoon cat icon. Below this, there's a 'Developer Quick Start' section with links to Tomcat Setup, First Web Application, Realms & AAA, JDBC DataSources, Examples, and Servlet Specifications. To the right, there are three boxes: 'Managing Tomcat' (with links to Documentation, Configuration, and Wiki), 'Documentation' (with links to 8.5 Documentation, 8.5 Configuration, and 8.5 Wiki), and 'Getting Help' (with links to FAQ and Mailing Lists). At the bottom, there are links for Other Downloads, Other Documentation, Get Involved, Miscellaneous, and Apache Software Foundation.

# Servlet: Compilazione

The screenshot shows the Tomcat Web Application Manager interface at [localhost:8080/manager/html](http://localhost:8080/manager/html). The page displays a list of deployed applications and provides options for deploying new ones.

**Applications**

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	<a href="#">Start</a> <a href="#">Stop</a> <a href="#">Reload</a> <a href="#">Undeploy</a> <a href="#">Expire sessions</a> with idle ≥ 30 minutes
/EsempioServlet	None specified	Primo esempio di utilizzo di una Servlet	true	0	<a href="#">Start</a> <a href="#">Stop</a> <a href="#">Reload</a> <a href="#">Undeploy</a> <a href="#">Expire sessions</a> with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	<a href="#">Start</a> <a href="#">Stop</a> <a href="#">Reload</a> <a href="#">Undeploy</a> <a href="#">Expire sessions</a> with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	<a href="#">Start</a> <a href="#">Stop</a> <a href="#">Reload</a> <a href="#">Undeploy</a> <a href="#">Expire sessions</a> with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	<a href="#">Start</a> <a href="#">Stop</a> <a href="#">Reload</a> <a href="#">Undeploy</a> <a href="#">Expire sessions</a> with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	<a href="#">Start</a> <a href="#">Stop</a> <a href="#">Reload</a> <a href="#">Undeploy</a> <a href="#">Expire sessions</a> with idle ≥ 30 minutes

**Deploy**

Deploy directory or WAR file located on server

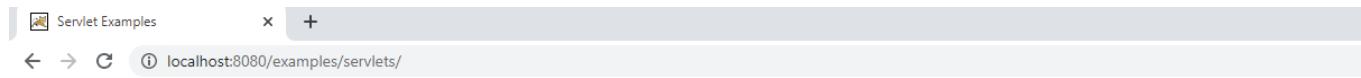
Context Path (required):   
XML Configuration file path:   
WAR or Directory path:

**WAR file to deploy**

Select WAR file to upload  Nessun file selezionato

# Servlet: Compilazione

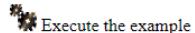
---



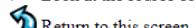
## Servlet Examples with Code

This is a collection of examples which demonstrate some of the more frequently used parts of the Servlet API. Familiarity with the Java(tm) Programming Language is assumed.

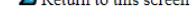
These examples will only work when viewed via an http URL. They will not work if you are viewing these pages via a "file:///..." URL. Please refer to the *README* file provided with this Tomcat distribution for more information.  
Wherever you see a form, enter some data and see how the servlet reacts. When playing with the Cookie and Session Examples, jump back to the Headers Example to see exactly what your browser is doing.  
To navigate your way through the examples, the following icons will help:



Execute the example



Look at the source code for the example



Return to this screen

Tip: To see the cookie interactions with your browser, try turning on the "notify when setting a cookie" option in your browser preferences. This will let you see when a session is created and destroyed.

Prova



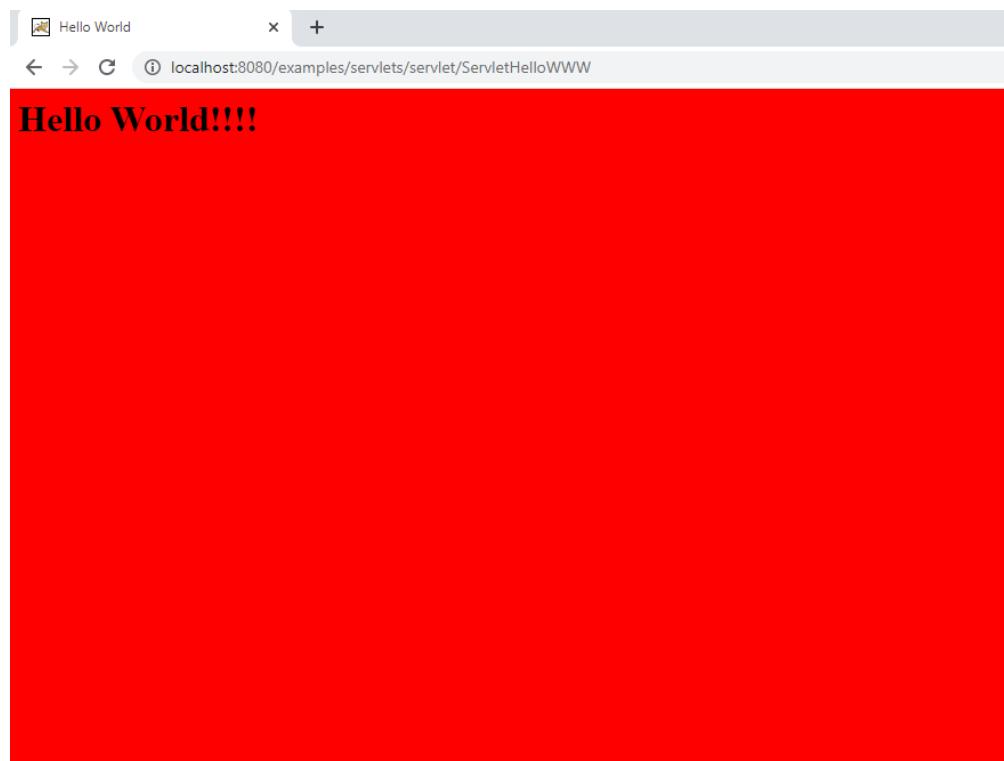
Hello World



...

# Servlet: Compilazione

*Servlet ha risposto quando la chiamava*



# Servlet con passaggio dei parametri

---

- Per passare dei parametri alla Servlet, oltre ai parametri di configurazione letti dal file `web.xml` mediante l'oggetto `ServletConfig` le servlet devono acquisire i parametri dal client e comportarsi di conseguenza.
- Le form HTML del client hanno sostanzialmente due modalità di invio dei dati alla servlet:
  - **GET:** agganciati alla URL come query string, si utilizza il metodo `doGet()`
  - **POST:** inseriti nel body del pacchetto HTTP, si utilizza il metodo `doPost()`

# Servlet con passaggio dei parametri

- `doGet()` e `doPost()` ricevono 2 argomenti:
  - l'oggetto `HttpServletRequest` per la gestione dei dati ricevuti dal client
  - l'oggetto `HttpServletResponse` per la gestione dei dati da inviare al client
- Con i metodi HTTP viene utilizzato il metodo `getParameter(<parametro>)` che restituisce una stringa col contenuto del parametro indicato.
- Sono anche disponibili i metodi `getParameterNames()` e `getParameterValues()` che rispettivamente forniscono i nomi dei parametri e un array che ne contiene i valori.

<input type="checkbox"/> <code>String getParameter(String)</code>	ritorna il valore del parametro dal suo nome;
<input type="checkbox"/> <code>Enumeration getParameterNames()</code>	ritorna una enumerazione dei nomi dei parametri;
<input type="checkbox"/> <code>String[] getParameterValues()</code>	ritorna una array di valori del parametro.
<input type="checkbox"/> <code>String getQueryString()</code> ritorna la query string completa (con il metodo GET)	

# Esempio

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
  <HEAD>
    <TITLE>Esempio di Servlet con GET </TITLE>
  </HEAD>
  <BODY>
    <FORM ACTION="servlet/EsempioForm" METHOD="GET">
      <p>
      <p>
        Inserisci il tuo nome e clicca sul pulsante per inviare i dati alla servlet <font color="#FF0000">Form1</font>
      </p>
      <p>
        <input name="nome" type="text" value=<@Sistemi@>> </P>
        <P>
          <input name="cognome" type="text" id="cognome" value="WEB"> <BR> <BR>
          <INPUT TYPE="submit" VALUE="Submit">
          <INPUT TYPE="reset" VALUE="Reimposta">
        </P>
        <p><strong>Nota</strong>: viene restituita una pagina <font color="#FF0000">HTML</font> generata dalla servlet
        <font color="#FF0000">EsempioForm.java</font>
      </p>
    </FORM>
  </BODY>
</HTML>
```

→ URL associata a un servlet

Inserisci il tuo nome e clicca sul pulsante per inviare i dati alla servlet **Form1**

Sistemi

WEB

**Nota:** viene restituita una pagina **HTML** generata dalla servlet **Form1.java**

# Esempio

```
// legge due parametri da una form HTML - metodo POST
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class EsempioForm extends HttpServlet {
    private String titolo, saluto;
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
        // i parametri nome e cognome vengono passati dal browser
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // get writer
        String nome = request.getParameter("nome");
        String cognome = request.getParameter("cognome");
        // crea una pagina HTML e la invia al client
        out.println("<html>"); out.println("<head>");
        out.println("<title> Esempio di Passaggio parametri da una form HTML - POST </title>");
        out.println("</head>");
        // contenuto della pagina html
        out.println("<body>");
        out.println("Ciao " + nome + " " + cognome + "!</p>");
        out.println("<b>La data di oggi e': " + new java.util.Date() + "</b></p>");
        out.println("</body>");
        out.println("</html>");
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
        doGet(request, response);
    }
}
```

Ciao Sistemi WEB  
La data di oggi è: .....

# Servlet con permanenza dei dati

---

- HTTP è “stateless”, cioè l’interazione client-server viene “dimenticata” e si passa a considerare la richiesta successiva.
- Possibili soluzioni:
  - Inserire informazioni inerenti allo stato direttamente negli URL, accodandoli come stringa di query (complessa e insicura).
  - Aggiungere alla pagina HTML dei campi nascosti (hidden) nella form, ma anche questa (insicura);
  - Utilizzo di cookie
  - Utilizzo delle sessioni.

# Cookie

---

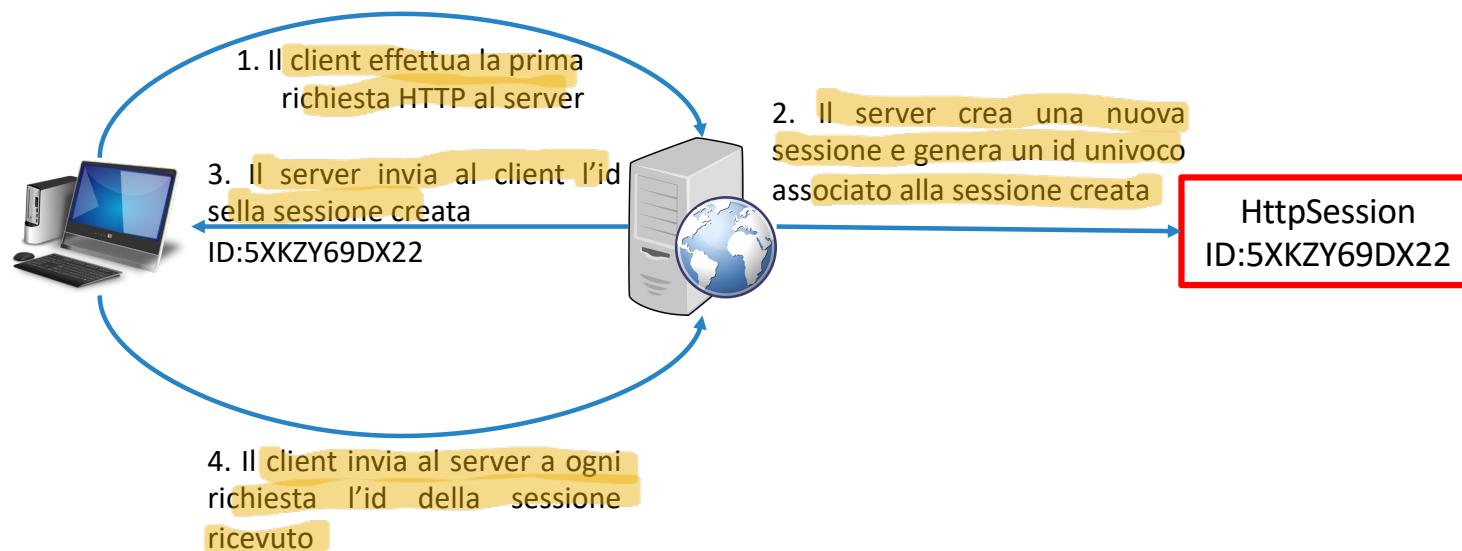
- Server e client si scambiano l'informazione presente nei cookie inglobandola negli header del protocollo http: il browser allega i cookie scaricati da un dato host a ogni richiesta effettuata da questo host.
- In Java l'API delle servlet fornisce la classe `javax.servlet.http.Cookie` che ci consente di utilizzare e manipolare facilmente i cookie.
- Creazione cookie: il metodo `addCookie()` dell'oggetto `HttpServletResponse` crea un nuovo cookie.
- Lettura cookie: il metodo `getCookies()` dell'oggetto `HttpServletResponse` legge dall'header HTTP i cookie (non è possibile accedere al singolo cookie, il metodo `getCookies()` ritorna tutti i cookie)
- Un cookie è un'istanza della classe `Cookie` che ha come costruttore:

**Cookie (String nome, String valore)**

```
Cookie mioCookie = new Cookie("Test","Prova");
response.addCookie(mioCookie);
```

# Sessioni

- La prima differenza sostanziale tra **cookies** e **sessioni** è che con le sessioni i dati sono memorizzati sul **server** e non sul **client**.
- Sul **client** viene memorizzato solo un **ID di sessione** (sempre attraverso un cookie).



# Sessioni

---

- Se il browser non supporta il cookie, il numero di sessione deve essere registrato nelle query string

```
http://localhost/servlet/miaServlet?sessionID=69163392288624
```

- Le API di Java mettono a disposizione la classe `javax.servlet.http.HttpSession` per definire le sessioni: il metodo `getSession()` permette di creare e restituire il contenuto della sessione.

```
HttpSession sessione = request.getSession();
```

# Sessioni

- Per gestire le sessioni sono disponibili i seguenti metodi:
  - per avere informazioni sulla sessione (ID della sessione, tempo trascorso, tempo di inattività)

```
String getID()  
long getCreationTime()  
long getLastAccessedTime()  
void invalidate()
```

ritorna l'ID della sessione  
ritorna il timestamp della creazione della sessione  
ritorna il timestamp dell'ultimo accesso alla sessione  
interrompe la validità della sessione

- per memorizzare dati di sessione nella forma di coppia chiave = valore, dove per evitare conflitti si segue la convenzione di dare un nome agli oggetti secondo lo schema:

<nome-applicazione>.<nome-oggetto>

```
void setAttribute(String chiave, Object valore)  
Object getAttribute(String chiave)  
void removeValue(String chiave)
```

aggiunge un oggetto  
recupera un oggetto dalla sessione  
elimina un oggetto dalla sessione

**public void setMaxInactiveInterval (int timeout)** per impostare il timeout si utilizza

Ho avuto in Java richiamati con un po' per mettere a connettere con un tipo di database.

# JDBC: Java Database Connectivity

---

- Sin dalle prime versioni di Java, esistono delle API che permettono alle applicazioni scritte in questo linguaggio di avere un livello di astrazione uniforme verso i database: le JDBC.
- JDBC si compone di una serie di classi per accedere alle BD: sfruttando un apposito driver JDBC costituito da una classe Java è possibile connettersi a un particolare database e oggi tutti i principali DBMS dispongono di uno specifico driver JDBC.
- Insieme a JDBC vi è un particolare driver, chiamato ponte JDBC-ODBC, che permette l'utilizzo di qualsiasi fonte di dati per la quale è disponibile un driver ODBC (Open DataBase Connectivity) (è possibile interagire con tutti i DBMS presenti sul mercato)

# JDBC: Java Database Connectivity

- Per prima cosa è necessario caricare il driver idoneo per l'utilizzo del particolare database che si intende sfruttare: ad esempio per la connessione ad un DBMS MySQL, che dispone sia di un driver JDBC che di un driver ODBC.
- Si può utilizzare il driver JDBC chiamato Connector/J, scaricabile dalla pagina <http://dev.mysql.com/downloads/connector/j/>.
    - Aggiungere il percorso dell'archivio JAR nella variabile di sistema CLASSPATH.
  - Come secondo passo si apre una connessione verso un database necessario all'applicazione, sfruttando il driver caricato al passo precedente.

```
dbc:mysql://[hostname][:port]/[dbname]?[param1=val1][&param2=val2]...
```

↑ Nome del database, parametri che posso passare.

# JDBC: Java Database Connectivity

---

3. Se la connessione va a buon fine, viene creato un oggetto `java.sql.Statement`, necessario per interagire con il DBMS attraverso delle query espresse in linguaggio SQL che memorizzano il risultato in un oggetto `java.sql.ResultSet`
4. I risultati ottenuti possono essere manipolati sfruttando i metodi dell'oggetto `resultSet`, ad esempio con un ciclo `while` si scorre il contenuto del `resultSet`.

# Esempio

Fatto su Java, non necessariamente per Servlet.

```
import java.sql.*;
public class EsempioDB {
    public static void main(String[] args) {
        // carico il driver per la connessione al DB mysql
        String DRIVER = "com.mysql.jdbc.Driver";
        try // carico il driver {
            Class.forName(DRIVER);
            // nome ed indirizzo del database
            String URL_mioDB = "jdbc:mysql://localhost:3306/proveJava";
            // definizione delle query
            String query = "SELECT Cognome, Nome, Dipartimento, Citta FROM impiegati";
            // stabilisco la connessione
            System.out.println("Connessione con: " + URL_mioDB);
            Connection connessione = DriverManager.getConnection(URL_mioDB, "root", "");
            Statement statement = connessione.createStatement();
            // interrogo il DBMS mediante una query SQL
            ResultSet resultSet = statement.executeQuery(query);
            // Scorro e mostro i risultati.
            while (resultSet.next()) {
                String cognome = resultSet.getString(1);
                String nome = resultSet.getString(2);
                String dipartimento = resultSet.getString(3);
                String citta = resultSet.getString(4);
                System.out.println("Lette informazioni..");
                System.out.println("Cognome: " + cognome);
                System.out.println("Nome: " + nome);
                System.out.println("Dipartimento: " + dipartimento);
                System.out.println("Citta': " + citta);
                System.out.println();
            }
        } catch (Exception e) { // il driver non può essere caricato, oppure Errore nella connessione
            System.out.println("Si sono verificati dei problemi");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

↑ Passo URL dell'connessione

↳ parla MySQL per eseguire query.

# Esempio con Servlet

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

public class ServletDB extends HttpServlet {
    private final String url = "jdbc:mysql://";
    private final String serverName = "localhost";
    private final String portNumber = ":3306/";
    private final String databaseName= "provejava";
    private final String userName = "root";
    private final String password = "";
    private final String URL_mioDB = url + serverName + portNumber + databaseName;
    private final String DRIVER = "com.mysql.jdbc.Driver";
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException { response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        try {
            Class.forName(DRIVER);
            String cognome = request.getParameter('cognome')
            String query = "SELECT Cognome, Nome, Dipartimento, Citta FROM impiegati WHERE cognome = ' + cognome;
            Connection connessione = DriverManager.getConnection(URL_mioDB,userName,password);
            Statement statement = connessione.createStatement();
            ResultSet resultSet = statement.executeQuery(query);
            out.println("<pre>");
            out.println("<b>cognome"+ "#;" +"nome"+ "#;" +"dipartimento"+ "#;" +"citta</b>");
            while (resultSet.next()) {
                String cognome = resultSet.getString(1);
                String nome = resultSet.getString(2);
                String dipartimento = resultSet.getString(3);
                String citta = resultSet.getString(4);
                System.out.println(cognome+ "#;" + nome+ "#;" + dipartimento+ "#;" + citta);
            }
            out.println("</pre>");
        }
        catch (Exception) {
            out.println("Errori nel caricamento del driver o della connessione");
            System.exit(1);
        }
    }
}
```

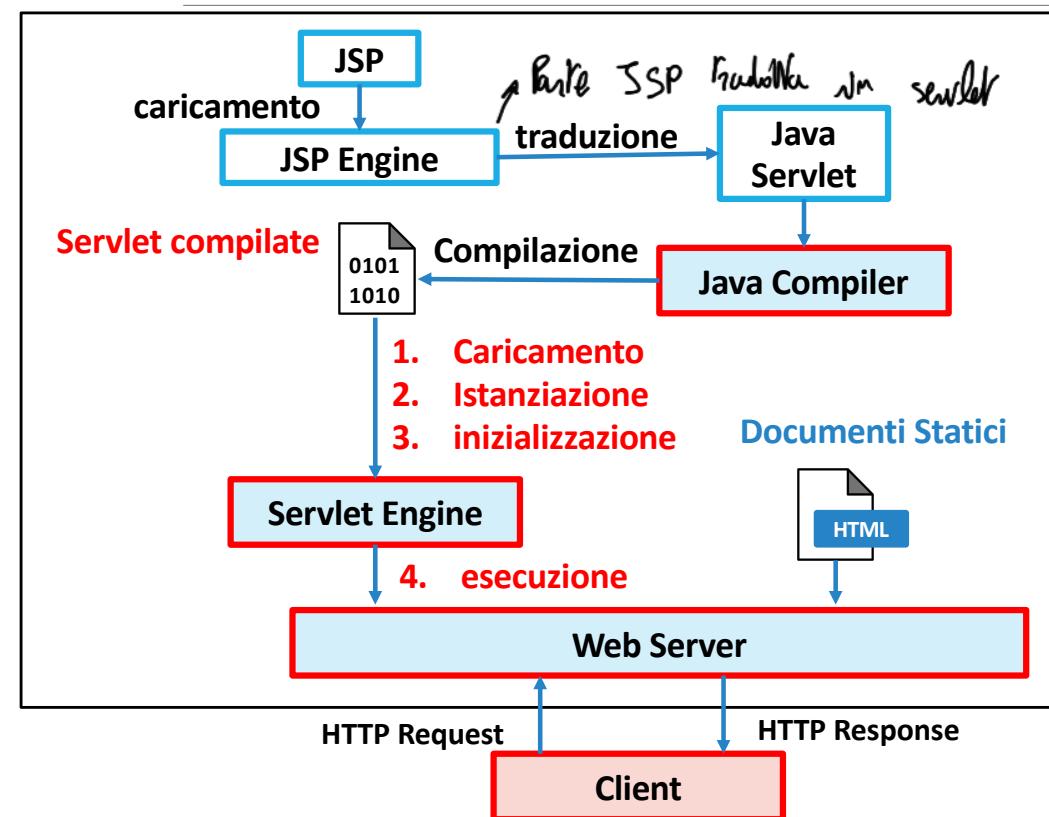
# JSP (Java Server Pages)

---

- Una Java Server Page è basata su Java, permette lo sviluppo di applicazioni Web con contenuti dinamici senza dover essere legati a un ambiente in particolare: una JSP è un'astrazione di alto livello per rappresentare delle servlet.
- Anche con JSP, come con le servlet, è possibile lavorare separatamente sulla presentazione grafica della pagina e includere in essa la logica di gestione creata, scritta appunto in JSP.
- Una pagina JSP è rappresentata da codice HTML al cui interno sono incapsulate delle direttive JSP (racchiuse in un particolare tag `<%...%>`) che rappresentano codice Java.
- Mentre nelle servlet è necessario compilare il codice Java e collocare il file .class nella directory classes prima di iniziare il caricamento delle pagine della wap, con le JSP la compilazione avviene in automatico, ovvero quando la pagina viene caricata.

Con le JSP e PHP posso anche in modalità script, progetto parte grafico insieme a logica di controllo.

# JSP (Java Server Pages)



- Quando un client richiede una pagina JSP il server effettuerà i seguenti passi:
  - compila il codice JSP per ottenere il codice Java rappresentante una servlet;
  - compilato il codice Java, individuato per generare il file .class rappresentante una servlet, invia alla classe la richiesta HTTP per ottenerne la risposta desiderata;
  - un programma preprocessore elabora la risorsa rimpiazzando i tag con codice HTML;
  - Spedisce la pagina risultante al browser del client.

# JSP: Tag

---

- Oltre all'HTML, in un pagina JSP le parti variabili sono contenute all'interno di tag speciali che possono avere due tipi diversi di sintassi:
  - Scripting-oriented tag;
  - XML-Oriented tag.

# JSP: Scripting-oriented tag

---

□ Le **scripting-oriented tag** sono definite da **delimitatori** entro cui è presente lo **scripting (self-contained)** e sono di quattro tipi:

- **<%= ... %> espressione** (calcolo valore)
- **<%! ... %> dichiarazione** (oggetti, variabili)
- **<% ... %> scriptlet** (entire e uscire dalla logica Java, es. Se voglio fare un if, parte di svolgendo)
- **<%@ ... %> direttiva** (import per esempio)

# JSP: XML-oriented tag

---

- Le **XML-oriented tag** utilizzano come sintassi la notazione tipica delle pagine **XML** e permettono di definire i medesimi tipi di comandi dei tag **scripting-oriented**.
  - `<jsp:expression>espressione</jsp:expression>`
  - `<jsp:declaration>dichiarazione</jsp:declaration>`
  - `<jsp:scriptlet>codice-java</jsp:scriptlet>`
  - `<jsp:directive.>tipo_direttiva attribute_direttiva />`

Nel **WEB** è più diffusa la notazione con i tag **scripting-oriented**

# JSP: Espressioni

---

- ❑ Nelle pagine JSP per valutare espressioni si usano delimitatori `<%= e %>` e il risultato viene convertito in stringa e inserito nella pagina al posto del tag.
- ❑ `<%= espressione %>`
- ❑ Viene valutata l'espressione e stampato il risultato nella pagina html generata.

```
<html>
  <head>
    <title>Primo codice JSP </title>
  </head>
  </body>
    Ciao la data di oggi è: <%= new java.util.Date().toString() %> <!-- espressione -->
  </body>
</html>
```

# JSP: Dichiarazioni

Per dichiarare variabili e metodi si utilizzano i delimitatori `<%!` e `%>` che vengono poi referenziati in qualsiasi punto del codice JSP, richiamandoli all'interno di una espressione sempre utilizzando i delimitatori `<%=` e `%>`.

```
<HTML>
  <HEAD>
    <TITLE> Pagina JSP </TITLE>
  </HEAD>
  <BODY><!-- dichiarazioni -->
    <%! String utente = "Sistemi Web e Basi di Dati";
      double[] prezzi = {9.5, 3.7, 22.5};
      double getTot() {
        double totale = 0.0;
        for (int x = 0; x < prezzi.length; x++)
          totale += prezzi[x];
        return totale;
      }
    %>
    <!-- espressioni -->
    <P>
      Gentile utente <%= utente %>, nella giornata di oggi <%= new java.util.Date().toString() %>
    </P>
    <P>
      ha effettuato acquisti per un totale di €: <%= getTot() %>.
    </P>
  </BODY>
</HTML>
```

La pagina JSP è trasformata in una Servlet e i metodi così dichiarati diventano metodi della servlet: il container crea un'istanza e a ogni richiesta all'istanza genera un thread.

# JSP: Scriptlet

---

- In una pagina JSP è possibile inserire frammenti di codice Java (e non solo) chiamati scriptlet, delimitati dal tag <% e %>. Comprendono istruzioni di controllo per la gestione del flusso di esecuzione del codice (es: selezione e l'iterazione).
- Quando la pagina viene tradotta all'atto della sua esecuzione tutti gli scriptlet divengono un blocco di codice Java.

```
<HTML>
    <HEAD>
        <TITLE> Pagina JSP </TITLE>
    </HEAD>
    <BODY> <!-- dichiarazioni -->
        <%! String utente = "Sistemi Web e Basi di Dati";
           double totale = 1230.50; %>
        <!-- espressioni -->
        <P><FONT COLOR="#0000FF">
            <h3>Gentile utente <%=utente%></H3></FONT> oggi <%= new java.util.Date().toString() %>
        </P>
        <P>ha effettuato acquisti per un totale di €: <%= totale %>:</P>
        <!-- scriptlet -->
        <% if (totale > 1000){ %>
            <H3>può pagare a rate</H3>
        <% } else { %>
            <H3>il pagamento deve avvenire in una sola soluzione.</H3>
        <% } %>
    </BODY>
</HTML>
```

# JSP: Oggetti impliciti

---

□ Il blocco di codice specificato dalla **scriptlet** viene inserito all'interno del metodo **service()** della **servlet** corrispondentemente generata alla richiesta di esecuzione della pagina **JSP**. Sono presenti quindi i seguenti oggetti:

- **page**: L'oggetto **page** rappresenta l'istanza corrente della **servlet** può essere utilizzato per accedere a tutti i metodi definiti nelle **servlet**.
- **config**: Contiene la **configurazione della servlet** (parametri di inizializzazione) e generalmente è lasciato inutilizzato.
- **request**: È l'oggetto di classe **HttpServletRequest** che rappresenta la **richiesta HTTP** che ha portato all'attivazione della pagina JSP/**servlet**: fornisce i metodi di accesso alla request HTTP corrente.
- **response**: È l'oggetto di classe **HttpServletResponse** che rappresenta la **risposta HTTP** da inviare al client.
- **out**: È un oggetto di classe **JSPWriter.out** e rappresenta il flusso di output su cui viene prodotta la **pagina Web**: spesso il suo uso è implicito, ma all'occorrenza si può fare riferimento a questa variabile invocandone i metodi nella successiva tabella.
- **session**: È l'oggetto di classe **HttpSession** che rappresenta la sessione HTTP all'interno della quale è stata invocata la pagina JSP: fornisce le informazioni sul contesto di esecuzione della JSP, cioè della sessione corrente di esecuzione per l'utente.

# JSP: Oggetti impliciti

---

□ Il blocco di codice specificato dalla **scriptlet** viene inserito all'interno del metodo **service()** della **servlet** corrispondentemente generata alla richiesta di esecuzione della pagina **JSP**. Sono presenti quindi i seguenti oggetti:

- **application**: È un oggetto che fornisce informazioni sul contesto di esecuzione della JSP e permette di accedere e di memorizzare gli oggetti per renderli accessibili da qualsiasi utente e modificabili da ogni pagina.
- **pageContext**: È un oggetto di classe **PageContext**, che rappresenta l'insieme degli oggetti impliciti associati all'intera pagina: l'oggetto può essere trasferito da una pagina JSP a un'altra ma viene poco utilizzato per lo scripting.
- **exception**: È un oggetto connesso alla gestione degli errori che viene utilizzato nelle Error Page, quelle che sono appositamente dichiarate con l'attributo **errorPage** impostato a true, e rappresenta l'eccezione che non viene gestita da nessun blocco catch.

# JSP: Direttive

---

- Sono comandi JSP valutati a tempo di compilazione e le più utilizzate sono le seguenti:
  - **include**: permettono di includere un altro documento;
  - **taglib**: carica una libreria di custom tag implementate dallo sviluppatore;
  - **page**: permette di importare package, dichiarare pagine d'errore, definire il modello di esecuzione della JSP relativamente alla concorrenza ecc.

- Include

```
<%@ include file="nomeFile.xxx"%>
```

- Permette di includere sia file statici (HTML) che altre JSP.

# JSP: Direttive

---

- **Page:** Permette di specificare alcune proprietà della pagina JSP che possono essere usate sia al momento della compilazione per costruire il corpo di alcuni metodi della

```
<%@ page attributo="valore" %>
```

- **taglib:** Nelle JSP è possibile definire dei tag addizionali che possono essere usati nella pagina nell'HTML definendone il loro comportamento in apposite librerie, le tag library, che vengono importate con la direttiva taglib.

```
<%@ taglib uri="libreria" prefix="prefisso" %>
```

- Il prefisso indicato è quello che verrà utilizzato all'interno della pagina per specificare i tag presenti nella libreria (sarà il prefisso di ogni tag presente nella libreria).

# JSP: Commenti

---

□ Questi marcatori permettono di inserire diversi tipi di commenti all'interno delle JSP.

□ <!-- commenti di contenuto -->

□ Sono i tipici commenti di HTML.

□ <%-- commenti JSP --%>

□ Sono i commenti visibili solo nel sorgente della JSP. (la servlet equivalente non conterrà nulla di questi commenti).

□ <% /\* commenti di scripting \*/ %>

□ Sono i commenti all'interno della parte di codice della JSP e sono visibili anche nella servlet equivalente.

# JSP: Azioni

---

- Questi marcatori permettono di supportare diversi comportamenti della pagina JSP.
- Vengono processati ad ogni invocazione della pagina JSP.
- Permettono di trasferire il controllo da una JSP all'altra, di interagire con JDB, ecc.
- Un'azione è introdotta con un tag del tipo:  
`<jsp:tipoAzione.../>`
- Ad esempio, se si vuole includere all'interno di una JSP dinamicamente un'altra JSP, è sufficiente inserire nel punto dove si vuole l'incursione l'azione:

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

# JSP: Java Bean

---

- All'interno delle pagine JSP è possibile utilizzare particolari classi Java con un'interfaccia molto semplice: sono i Java Bean ("chicchi di Java").
- Un Java Bean è un componente nella tecnologia Java, cioè una classe, che può essere utilizzato in modo standard in più applicazioni.

# JSP: Java Bean

---

- Un Bean è una classe particolarmente semplice che risponde ai seguenti requisiti:
  - È una classe public
  - Ha un costruttore vuoto
  - Non ha variabili pubbliche
- ogni variabile è accessibile tramite metodi del tipo `getAttributo()` e `setAttributo()`
  - `isAttributo()` nel caso di variabili booleane
- se manca il `get` si parla di variabile `writeonly`
- se manca il `set` si parla di variabile `readonly`

# JSP: Java Bean - Esempio

---

```
public class StringBean {  
    private String message = "Nessun messaggio specificato";  
    public String getMessage() {  
        return(message);  
    }  
    public void setMessage(String message) {  
        this.message = message;  
    }  
}
```

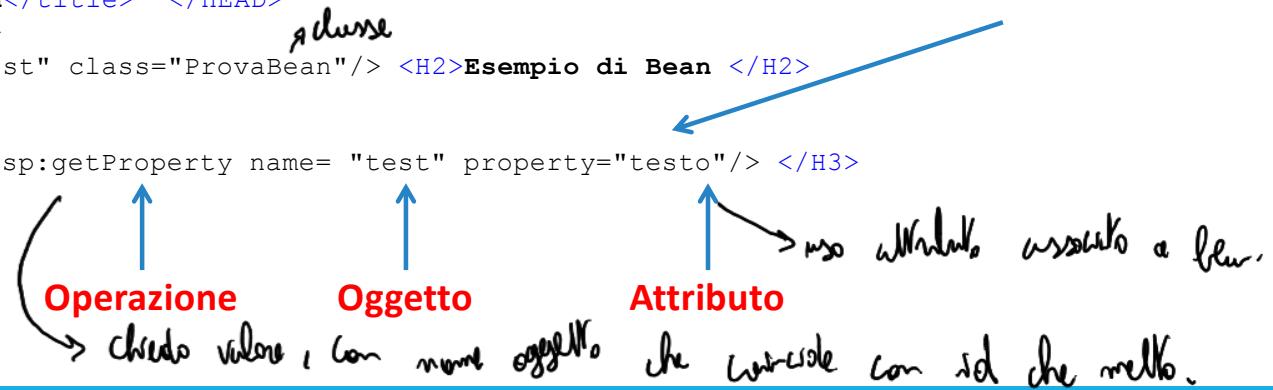
# JSP: Java Bean

→ dobbiamo usare la parte XML oriented

- Tra le varie azioni esiste la <jsp:useBean ...> che consente di usare un Bean all'interno di una pagina
- Tale tecnica è particolarmente utile in quanto consente
  - il riutilizzo di classe java
  - di mantenere la logica di business fuori dalle pagine

```
<HTML>
  <HEAD> <title>JSP e Bean</title>  </HEAD>
  <BODY BGCOLOR="#ffffcc">
    <jsp:useBean id="test" class="ProvaBean"/> <H2>Esempio di Bean </H2>
    <HR>
    <CENTER>
      <H3>Saluto: <jsp:getProperty name= "test" property="testo"/> </H3>
    </CENTER> <HR>
  </BODY>
</HTML>
```

In pratica esegue il metodo getTesto()



Fatto get→keyNo.

# JSP: Java Bean

- Tra le varie azioni esiste la <jsp:useBean ...> che consente di usare un Bean all'interno di una pagina
- Tale tecnica è particolarmente utile in quanto consente
  - il riutilizzo di classe java
  - di mantenere la logica di business fuori dalle pagine

```
<HTML>
  <HEAD> <title>JSP e Bean</title>  </HEAD>
  <BODY BGCOLOR="#ffffcc">
    <jsp:useBean id= "test" class="ProvaBean"/> <H2>Esempio di Bean </H2>
    <HR>
    <CENTER>
      <H3>Saluto: <jsp:setProperty name="test" property="testo" value="ciao"/> </H3>
    </CENTER> <HR>
  </BODY>
</HTML>
```

In pratica esegue il metodo setTesto()

↑  
Operazione      ↑  
Oggetto      ↑  
Attributo

Qua setto al valore  
ma me lo stampa anche



# JSP: Java Bean

- `<jsp:useBean id="book1" class="coreservlets.Book" />`
- Si può pensare che tale riga sia equivalente a `<% coreservlets.Book book1 = new coreservlets.Book(); %>` anzi che la seconda sia più potente essendo possibile, ad esempio, usare costruttori diversi da quello vuoto.
- In realtà la `jsp:useBeans` permette di usare l'attributo `scope` che consente lo condivisione di Beans
- Quindi
  - `<jsp:useBean id="book1" class="coreservlets.Book" />` instanzia un Bean solo se non esiste uno con lo stesso nome e con lo stesso scope. In caso contrario ci si collega a quello esistente.

## SINTASSI COMPLETA

```
<jsp:useBean id="saluto" class="ProvaBean" scope="page|request|session|application"/>
```

# JSP: Java Bean – Condivisione (Scope)

---

- ❑ l'attributo **scope** è opzionale e permette di definire l'ambito di accessibilità e il tempo di vita dell'oggetto.
- ❑ **page**
  - ❑ Valore di default. Il bean è visibile solo all'interno della pagina.
- ❑ **session**
  - ❑ Il bean, una volta creato, vive fino a che la sessione viene invalidata (esplicitamente o implicitamente)
- ❑ **request**
  - ❑ Il bean vive per il tempo della richiesta (che potrebbe coinvolgere più pagine)
- ❑ **application**
  - ❑ Il bean è condiviso da tutta l'applicazione.

NOTA: come creare sessioni. Possono appartenere a server.

Verifica interazione con le sessioni.

# JSP: MySQL Esempio

```
<HTML>
<BODY>
    <H2>JSP e database MySQL</H2>
    <%@ page import="java.sql.*" %>
    <TABLE BORDER="1">
        <% // carico il driver per la connessione al DB MySQL
        String DRIVER = "com.mysql.jdbc.Driver";
        String URL_mioDB = "jdbc:mysql://localhost:3306/proveJava";
        String query = "SELECT Cognome, Nome, Indirizzo, Citta FROM Amici";
        try{
            Class.forName(DRIVER);
            Connection connessione = DriverManager.getConnection(URL_mioDB, "root", "");
            Statement statement = connessione.createStatement();
            ResultSet resultSet = statement.executeQuery(query);
            out.println("<PRE>");
            out.println("<B>cognome"+ "#;" +"nome"+ "#;" +"indirizzo"+ "#;" +"citta</B><BR>");
            while (resultSet.next()) {
                String cognome = resultSet.getString(1);
                String nome = resultSet.getString(2);
                String indirizzo = resultSet.getString(3);
                String citta = resultSet.getString(4);
                out.println( cognome+ "#;" + nome+ "#;" + indirizzo+ "#;" + citta);
            }
            out.println("</PRE>");
            connessione.close();
            out.close();
        }
        catch (Exception e) { System.err.println("Errore: " + e); }
    %>
    </TABLE>
</BODY>
</HTML>
```