Department of Computer Science

http://www.cs.unibo.it

Applicazioni Web: basics

Angelo Di Iorio

(dal materiale su Servlet/Tomcat di Silvio Peroni, speroni@cs.unibo.it)

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

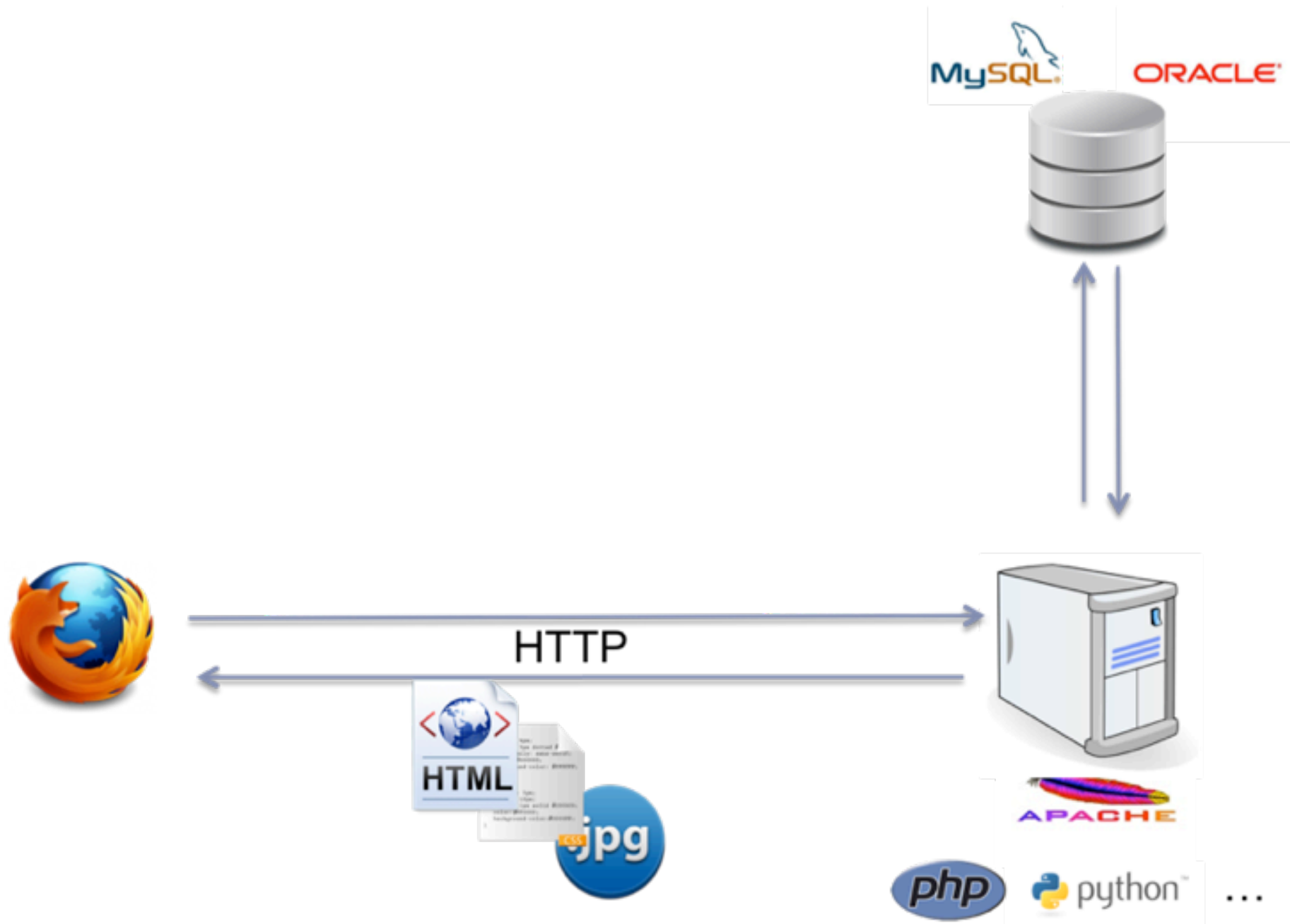
Architettura client-server

- L'organizzazione del WWW è basata su servizi di tipo ***client-server***:
  - I **server** mettono a disposizione le informazioni, cioè i documenti ipertestuali e multimediali, che sono residenti sul server stesso o possono essere creati dinamicamente
  - I **client** (browser) accedono ai documenti multimediali memorizzati sui server, attraverso un indirizzo che individua univocamente i documenti multimediali sulla rete.
- Il protocollo di comunicazione è **HTTP**: HyperText Transfer Protocol

Metodi HTTP

- **GET:** richiede una risorsa al server
- **HEAD:** richiede le informazioni su una risorsa al server (simile al GET ma il server ritorna solo gli header)
- **POST**: trasmette informazioni al server senza la creazione di una nuova risorsa
- **PUT**: trasmette informazioni al server per creare o sostituire una risorsa
- **DELETE**: cancella una risorsa

Contenuti statici e dinamici



Scripting server-side

- I linguaggi di scripting server-side permettono di scrivere codice che viene **interpretato** dal server
  - moduli del server (es. mod_php, mod_python, etc.)
  - CGI, Common Gateway Interface
- Il server riceve la richiesta HTTP e la "passa" all'interprete che processa il file sorgente:
  - legge eventuali parametri di input
  - li elabora e recupera i dati
  - formatta la risposta (codifica nel formato richiesto)

PHP

- PHP (acronimo ricorsivo per "PHP: Hypertext Preprocessor“, o “Personal Home Page”) è un linguaggio di scripting server-side general-purpose e open-source
- Il codice PHP è delimitato da speciali *start* e *end* tag che consentono di passare dal “modo HTML” al “modo PHP”
  - **`<?php ... ?>`** è il più comune ma ce ne sono altri
- PHP fornisce tutti i costrutti di base dei linguaggi procedurali (e object-oriented, vedi prossime slide):
  - Tipi primitivi: *boolean, integer, float, string, array, ...*
  - Variabili (indicate con il simbolo **`$`**) e funzioni
  - Strutture di controllo: *if, else, while, for, switch, ...*

HelloWorld PHP

```
<html>
<head>
  <title>Test PHP</title>
</head>
<body>
   <?php
   $msg = "Hello World!";
   echo "<p>".$msg."</p>";
   ?>
</body>
</html>
```

PHP: processare un form

Lascia un messaggio:

Invia

```
<html>
<head>
    <title>Processing
</head>
<body>
    <p>Lascia un messa
    <form method="post" action="./message.php">
    <p><input type="text" name="testo" /></p>
    <p>
        <input type="submit" value="Invia" />
    </p>
    </form>
</body>
</html>
```

```
<html>
<head>
    <title>Processing form in PHP</title>
</head>
<body>
    <p>Grazie, messaggio ricevuto:</p>
    <p style="font-weight:bold">
        <?php echo $_POST['testo']; ?>
    </p>
    <p><a href='index.php'>Torna Indietro.</a></p>
</body>
</html>
```

Grazie, messaggio ricevuto:

hello world

Torna Indietro.

PHP: variabili speciali

- PHP definisce un certo numero di array associativi speciali disponibili all'interno degli script server-side tra le quali:
  - **$_SERVER**: contiene variabili impostate dal web server e relative all'ambiente di esecuzione dello script
  - **$_GET**: contiene variabili ricevute dallo script via HTTP GET
  - **$_POST**: contiene variabili ricevute dallo script via HTTP POST
  - **$_SESSION**: contiene variabili che sono correntemente registrate nella sessione di esecuzione dello script

PHP: molto altro

- PHP ha molte altre caratteristiche utili per costruire applicazioni sofisticate:
  - Innanzitutto supporta **programmazione object-oriented**: *classi, oggetti, interfacce, ereditarietà, etc.*
  - Integrazione con i database e con i meccanismi di storage
  - Numerose librerie built-in o disponibili in rete
- Noi non guardiamo questa parte, vi rimando alla lezione del corso di tecnologie web (prevista a breve)

Un punto importante

- Il codice può essere "immerso" nelle pagine HTML o, molto meglio, organizzato in componenti disaccoppiate, modulari, testabili e basate su pattern
- **Non mescolare HTML e PHP**!

```
<?php
if ($expression) {
?>
<strong>Questa è vera.</strong>
<?php
            } else {
?>
<strong>Questa è falsa.</strong>
<?php
}
?>
```

- **Lo stesso discorso vale per tutti gli altri linguaggi di scripting server-side**

JSP: JavaServer Pages

- JSP è un linguaggio di scripting server-side che permette di creare pagine web dinamiche usando tecnologie Java
- Per molti aspetti simile a PHP, permette di "embeddare" il codice Java nelle pagine HTML
- JSP usa delimitatori speciali per racchiudere le funzioni di scripting
  - `<% ... %>`: frammenti Java eseguiti quando la pagina JSP viene processata
  - `<%= ... %>`: espressioni il cui valore viene inserito nella pagina costruita dinamicamente
  - `<%@ ..%>`: direttive di pre-processing utili, ad esempio, per includere altri frammenti JSP,

JSP: processare un form

Lascia un messaggio:

Invia

```
<html>
<head>
    <title>JSP</title>
    </head>
<body>
    <p>Lascia un messaggio:</p>
    <form method="post" action="./message
        <p><input type="text" name="testo
        <p>
            <input type="submit" value="Invia" />
        </p>
    </form>
</body>
</html>
```

```
<html>
<head>
    <title>JSP</title>
</head>
<body>
    <p>Grazie, messaggio ricevuto:</p>
    <p style="font-weight:bold">
        <%= request.getParameter("testo") %>
    </p>
    <p><a href='./oldjsp.html'>Torna Indietro.</a></p>
</body>
</html>
```

Grazie, messaggio ricevuto:

hello world

Torna Indietro.

JSP e Servlet

- Le pagine JSP sono **tradotte** a run-time in **Servlet** che, a loro volta, sono **compilate** ed eseguite in un **Web Container**

- Una Servlet è una classe Java conforme a un protocollo, chiamato *Java Servlet API*, attraverso il quale una classe Java può rispondere a richieste HTTP
  - Il contenuto generato è solitamente HTML, ma è possibile produrre altri formati come XML o JSON

- Il Web Container (es. Tomcat, JBOSS, Jetty, ...) è un componente di un server web che gestisce il ciclo di vita delle servlet e fornisce/delimita l'ambiente di esecuzione

Ciclo di vita di una servlet

- La servlet viene caricata dal *Web container* durante il suo start-up
- Il Web container chiama (e lo fa solo una volta) il metodo *init()* per inizializzare la servlet. In questo modo abilita la servlet a rispondere alle varie richieste dirette verso di lei
- Ogni richiesta HTTP è servita usando un thread separato. Il Web container chiama il metodo *service()* della servlet che riceve la richiesta. È quest'ultimo metodo che redireziona la richiesta in modo opportuno, chiamando il metodo appropriato per soddisfarla – *doGet()*, *doPost()*, *doPut()*, *doDelete()*
  - Le servlet implementano i metodi di HTTP
  - Se una richiesta fa riferimento a un metodo non implementato nella servlet, viene chiamato lo stesso metodo della super classe della servlet (tipicamente ritorna un errore)
- Infine, il Web container chiama (e lo fa solo una volta) il metodo *destroy()* quando vuole rendere la servlet non più disponibile

Servlet: struttura di base

```java
package it.essepuntato.test.servlet;

import java.io.IOException;

/**
 * Servlet implementation class UnaServlet
 */
public class UnaServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public UnaServlet() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        // TODO Auto-generated method stub
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        // TODO Auto-generated method stub
    }
```

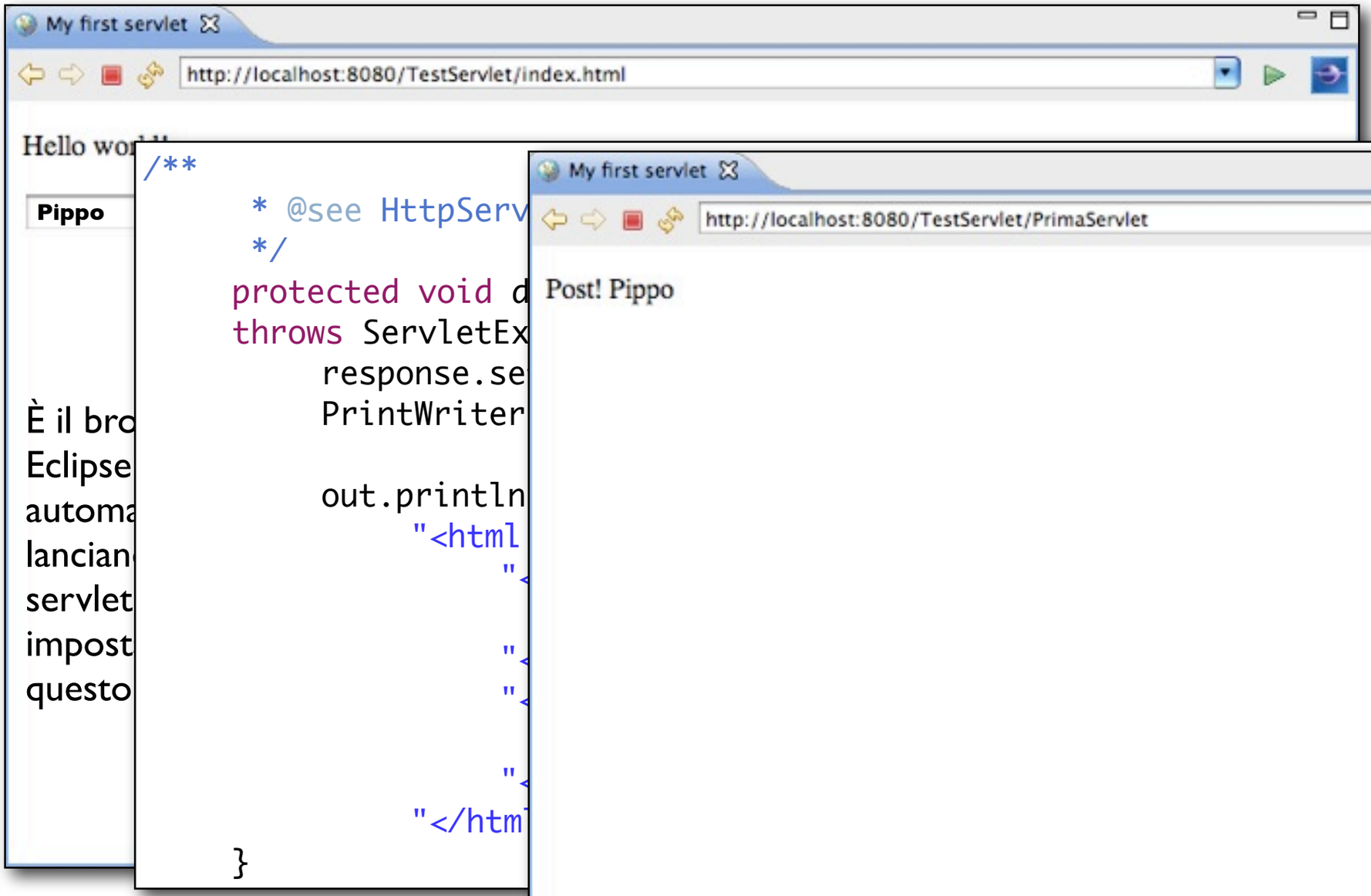
HelloWorld Servlet

contiene tutte le informazioni relative alla richiesta (header http, metodo usato, parametri, ecc.)

contiene tutto ciò che contiene la risposta (output HTML, invio di errori, tipo di contento ritornato, ecc.)

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("application/xhtml+xml"); //text/html per HTML5 (no XML)
        PrintWriter out = response.getWriter();

        out.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
                "<html xmlns=\"http://www.w3.org/1999/xhtml\">" +
                    "<head>" +
                        "<title>My first servlet</title>" +
                    "</head>" +
                    "<body>" +
                        "<p>Hello world!</p>" +
                        "<form method=\"post\" action=\"./index.html\">" +
                            "<input type=\"text\" name=\"testo\" />" +
                            "<input type=\"submit\" value=\"Invia\" />" +
                        "</form>" +
                    "</body>" +
                "</html>");
    }
```

Servlet: processare un form



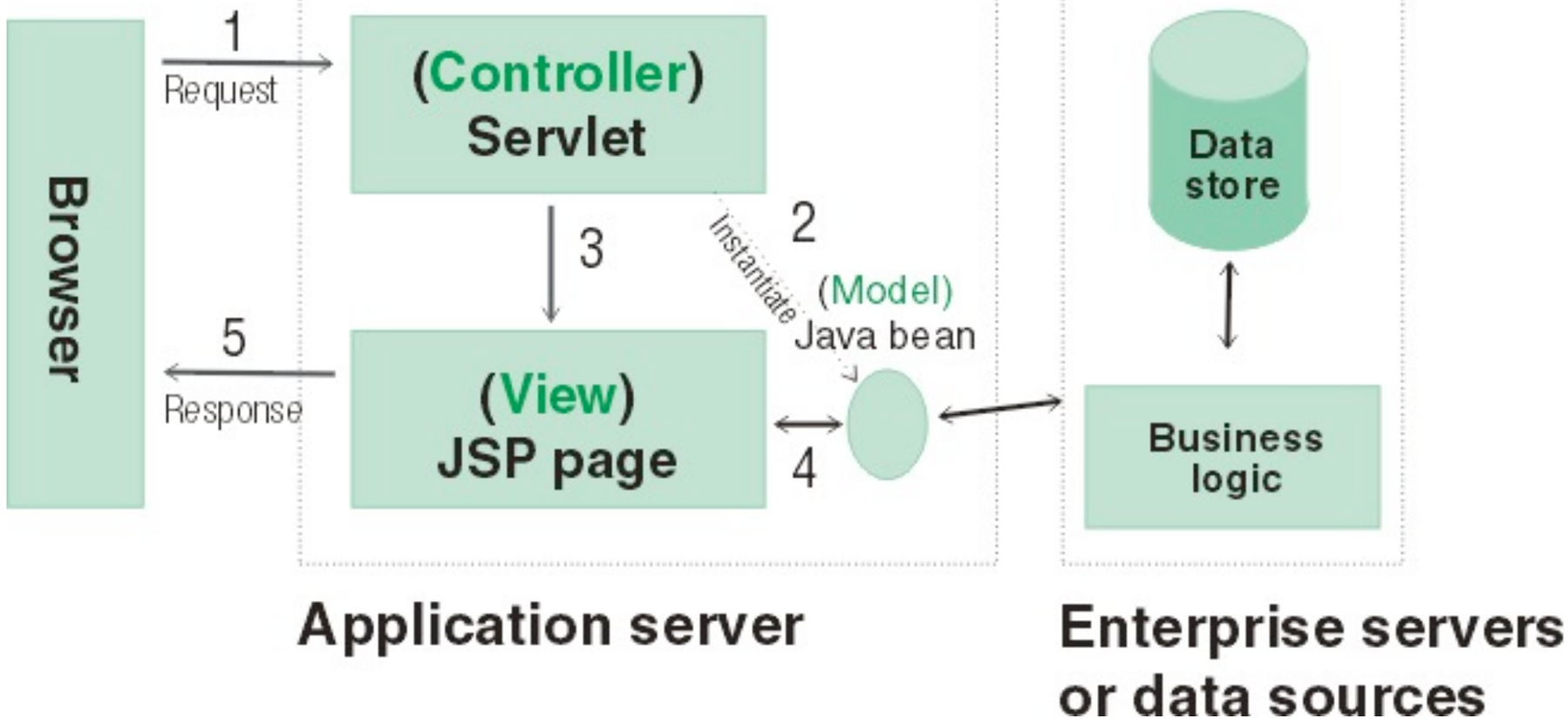
Business logic e presentazione

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
     response.setContentType("application/xhtml+xml");

   // Business logic:
   // Elaborazione parametri
    String testo = request.getParameter("testo");

   // Processing e salvataggio dati  ....

   //  Output
     PrintWriter out = response.getWriter();
     out.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
          "<html xmlns=\"http://www.w3.org/1999/xhtml\">" +
               "<head>" +
                    "<title>My first servlet</title>" +
               "</head>" +
               "<body>" +
                    "<p>Post! " + testo + "</p>" +
               "</body>" +
          "</html>");
}
```

Model View Controller

- MVC è un pattern architetturale che permette di separare la logica di business dalla presentazione
- L'applicazione si struttura in tre componenti principali:
  - **Model**: si occupa di gestire i dati, memorizzarli (ancora meglio se in un layer separato) e modificarli
  - **View**: si occupa della visualizzazione dei dati e dell'interazione con l'utente
  - **Controller**: si occupa della logica dell'applicazione, della lettura dei parametri e dello smistamento delle richieste e dell'interazione tra Model e View

JSP Model 2 Architecture



MVC con servlet e JSP

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
          response.setContentType("application/xhtml+xml");

        // Elaborazione parametri
          request.setAttribute("messaggio", request.getParameter("testo"));

        // Business logic (Model)

        // View
        RequestDispatcher view = request.getRequestDispatcher("./WEB-INF/       templates/
        messagge.jsp");
          view.forward(request, response);
    }
```

```
<html>
    ...
    <body>
        <p>Second servlet: post and JSP!</p>
        <p><b>${messaggio}</b></p>
        <p><a href='./SecondServlet'>Torna indietro.</a></p>
    </body>
</html>
```

Un esempio utile

- Dopo aver recuperato nella parte Model una lista di oggetti Message (indipendentemente da dove e come sono memorizzati!) possiamo mostrarli in una tabella, nella corrispondente View:

```
<table>
<tr>
<% while(iteratorMessages.hasNext())
  {
  msg = (Message) iteratorMessages.next();%>
  <tr>
     <td>
          <a href="ShowMessages?id=<%= msg.getId()%>">
          <%= msg.getId() %>
          </a>
     </td>
     <td>
          <%=msg.getContent()%>
  </td>
<%}%>
</table>
```

Note su MVC

- La versione che abbiamo visto è un MVC molto rudimentale
- Si potrebbe prevedere un *FrontController (*o *Dispatcher*): unico "punto di ingresso" al sistema che smista le richieste a *Controller* specializzati, e che a loro volta hanno un *View* associato
- Questa struttura è fornita ad esempio dai principali framework per applicazioni Web in Java (Struts, Spring) o PHP (Zend, Symphony, ...), Python, Ruby
- Noi non guardiamo più in dettaglio MVC e Java Beans (e altri dettagli sintattici di JSP) ma è importante **separare la logica di business dalla presentazione**
- Lo stesso discorso vale per un'applicazione in PHP o altri linguaggi

Department of CS http://www.cs.unibo.it

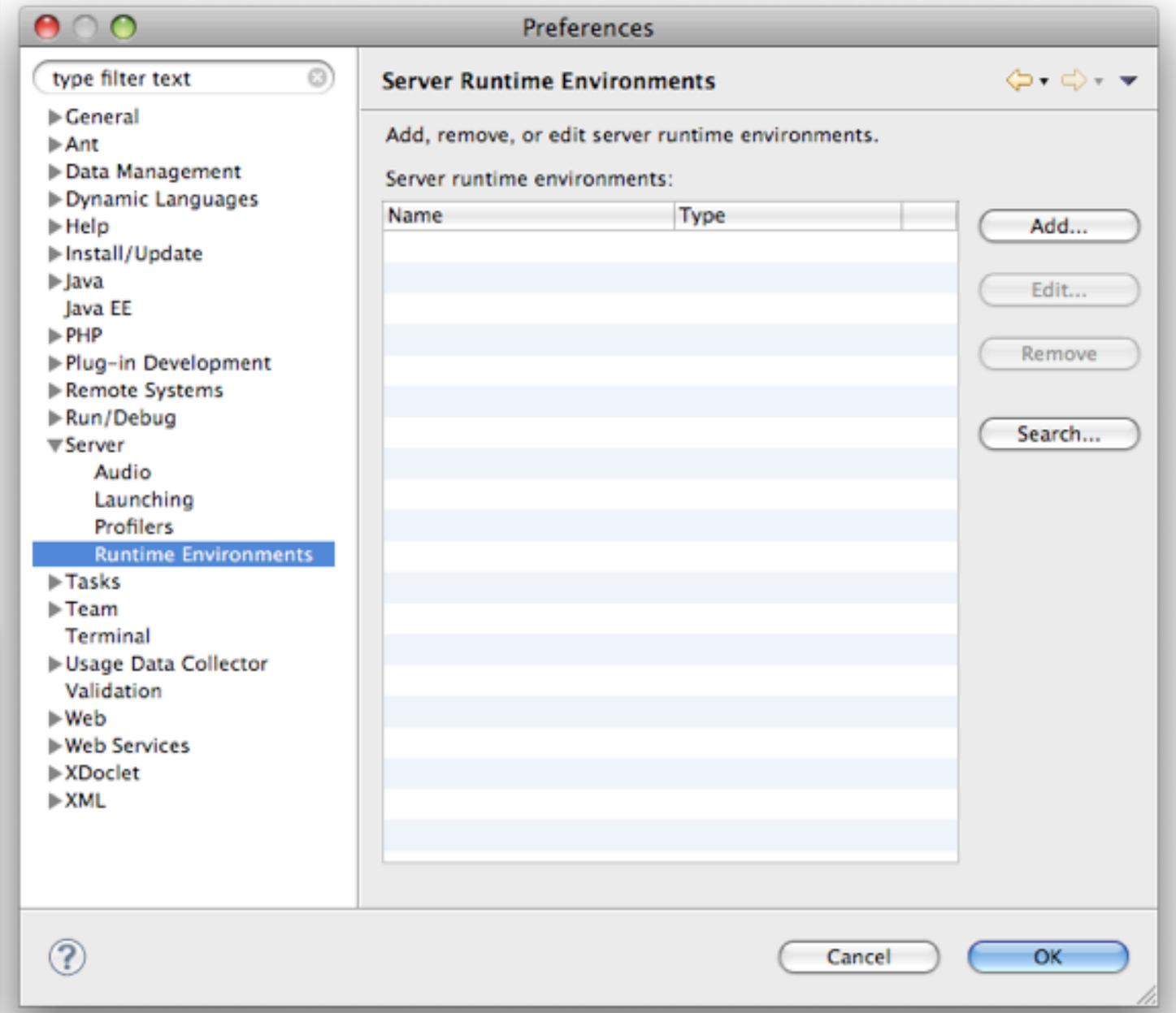
Tomcat e Servlet in Eclipse

Un Web container: Tomcat

- *Apache Tomcat* – `http://tomcat.apache.org` – è un *Web container* sviluppato dalla Apache Software Foundation
- Scaricare il pacchetto binario per il sistema operativo scelto, decomprimere il pacchetto e, da shell, entrare dentro la directory *bin*
  - avvio: *startup*
  - stop: *shutdown*

Tomcat e Eclipse

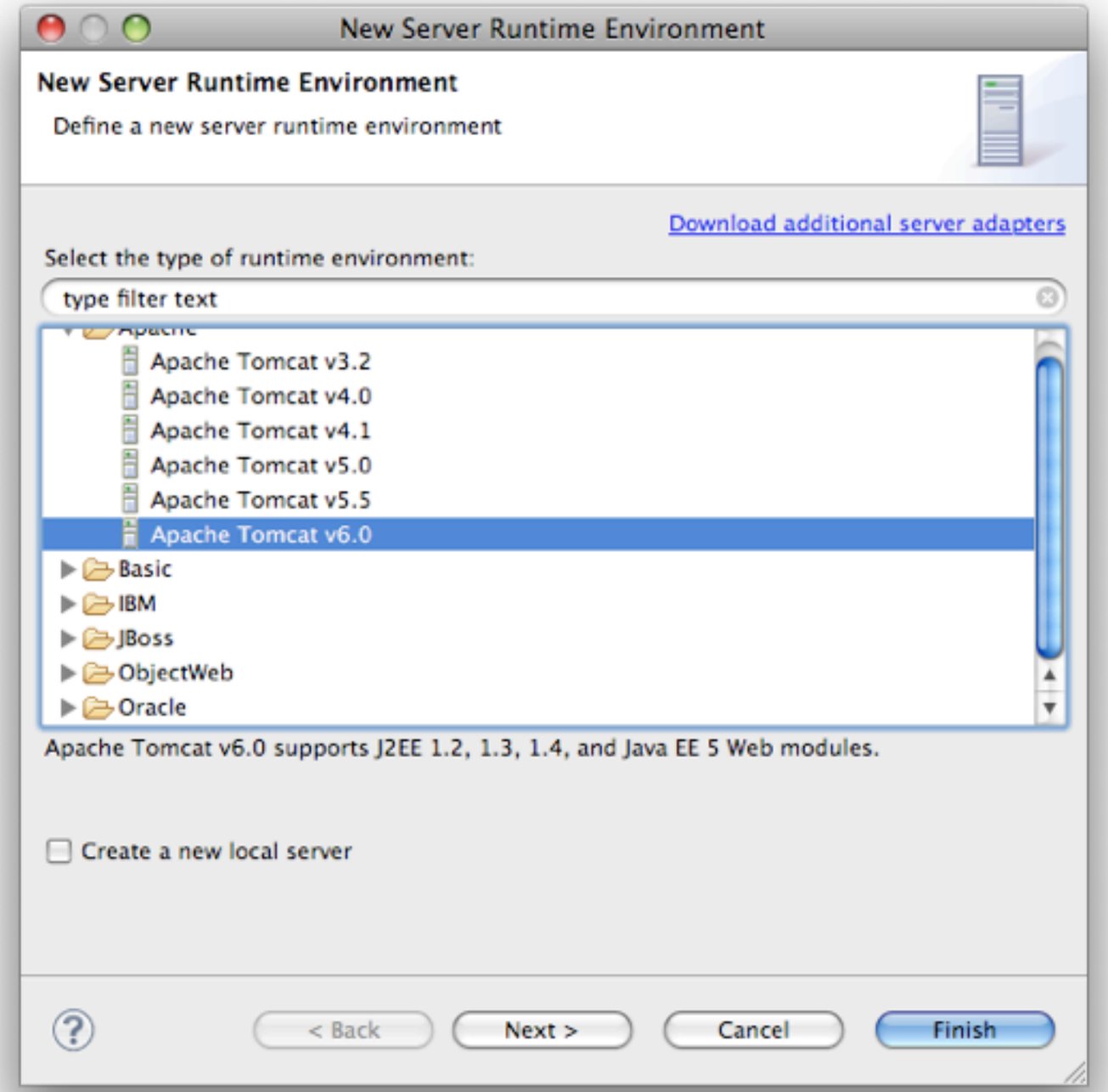
Configurare un nuovo server



Andare sulle preferenze di Eclipse e cliccare sull'etichetta "Runtime Environments" di "Server"

Tomcat e Eclipse

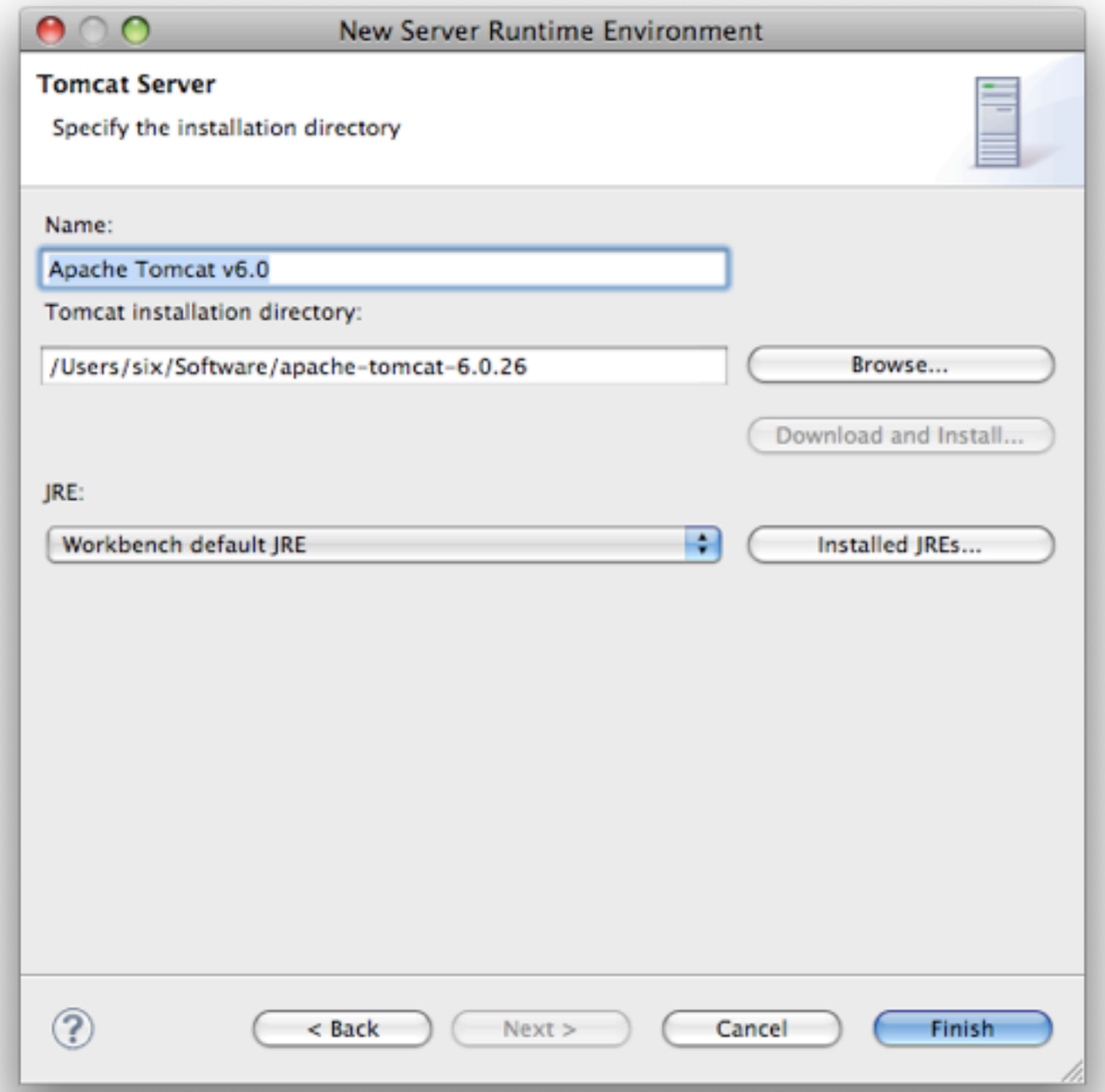
Scelta del server



Selezionare "Apache Tomcat v6.0", che avete precedentemente installato nel vostro sistema

Tomcat e Eclipse

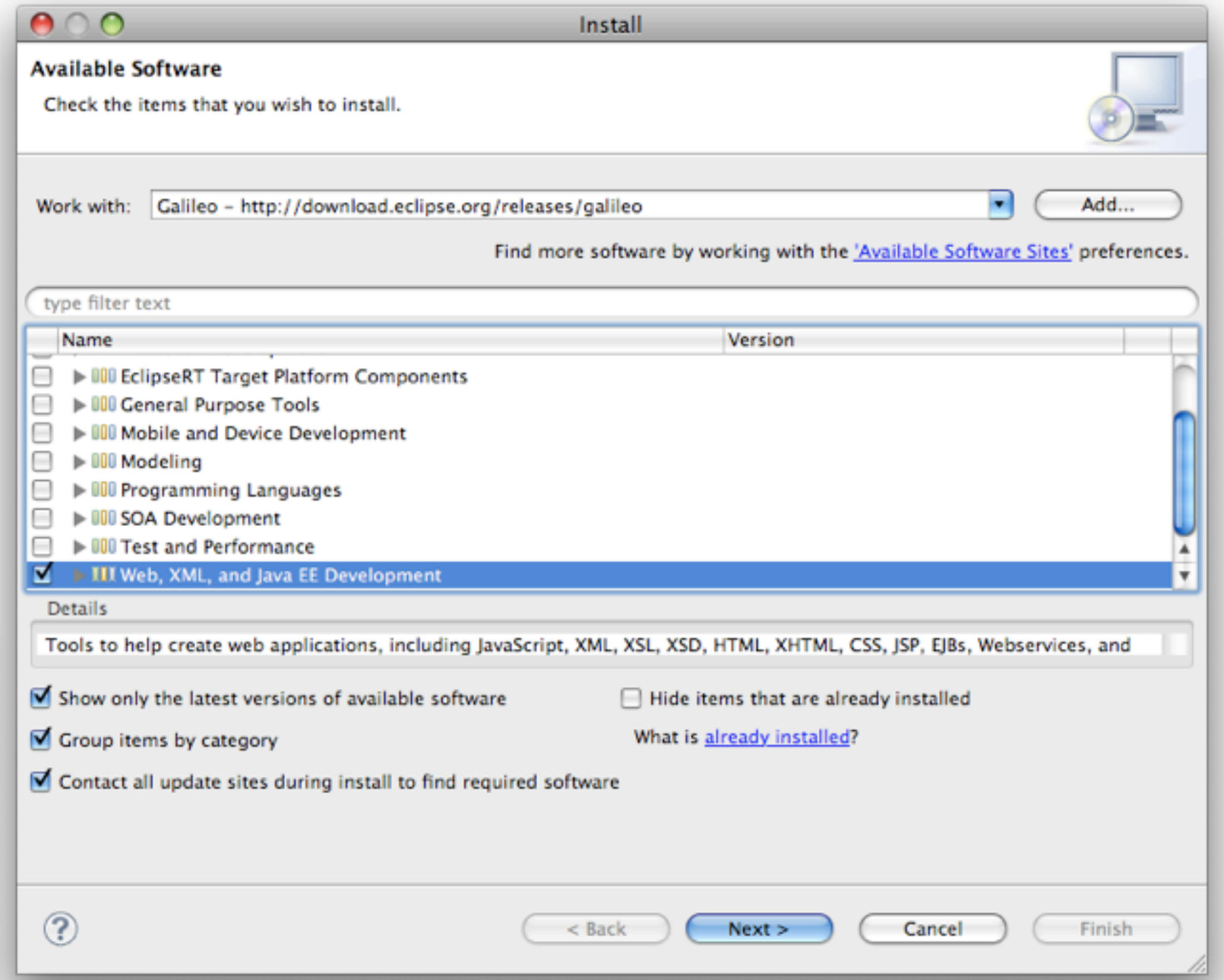
Specificare il percorso



Specificate un'etichetta e il percorso di dove avete installato Tomcat sul vostro sistema

Servlet e Eclipse

Web, XML, and Java EE Development

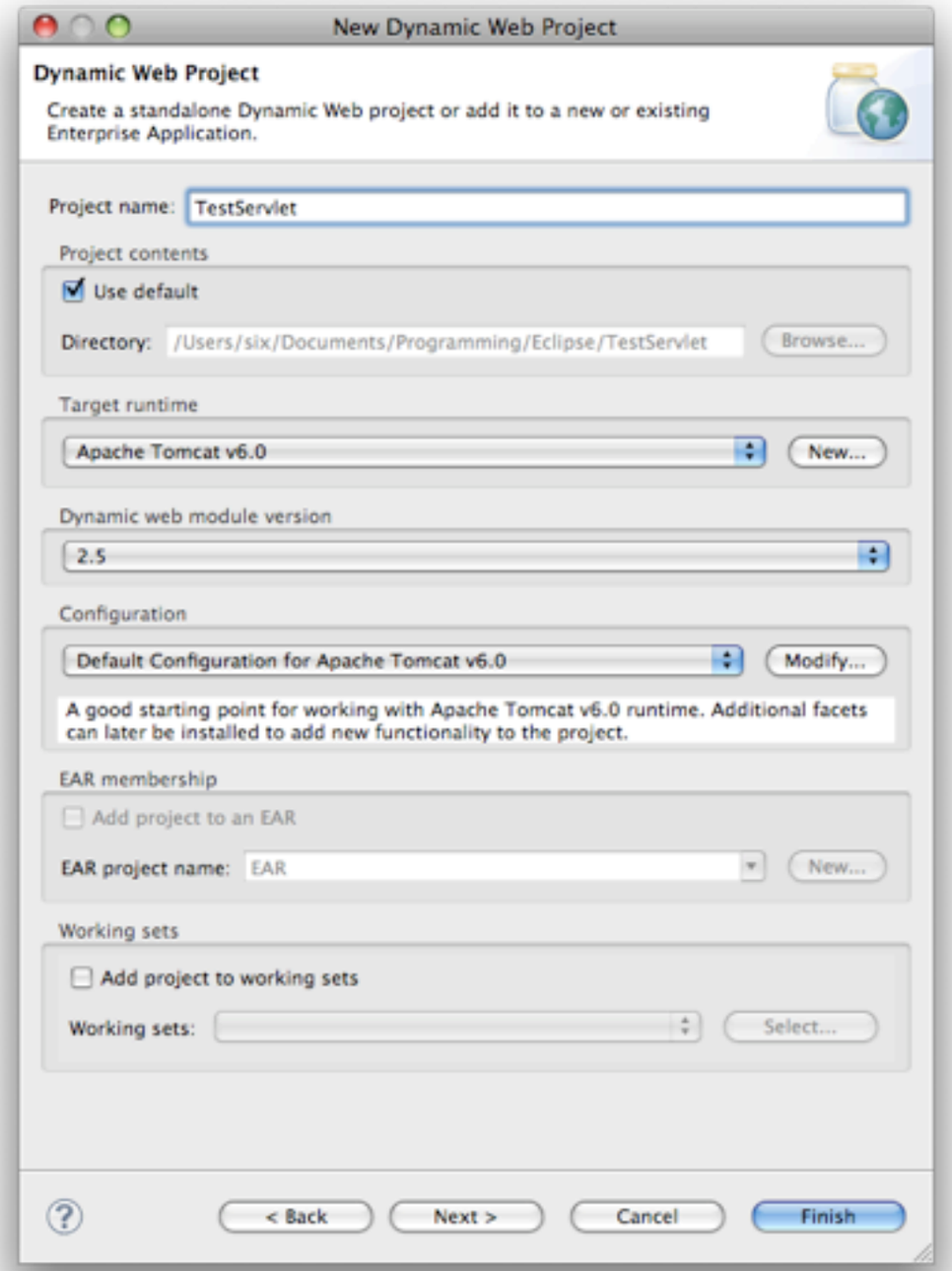


Installare tutto quel che riguarda il modulo "Web, XML, and Java EE Development"

http://www.cs.unibo.it
Department of CS

Servlet e Eclipse

Dynamic Web Project

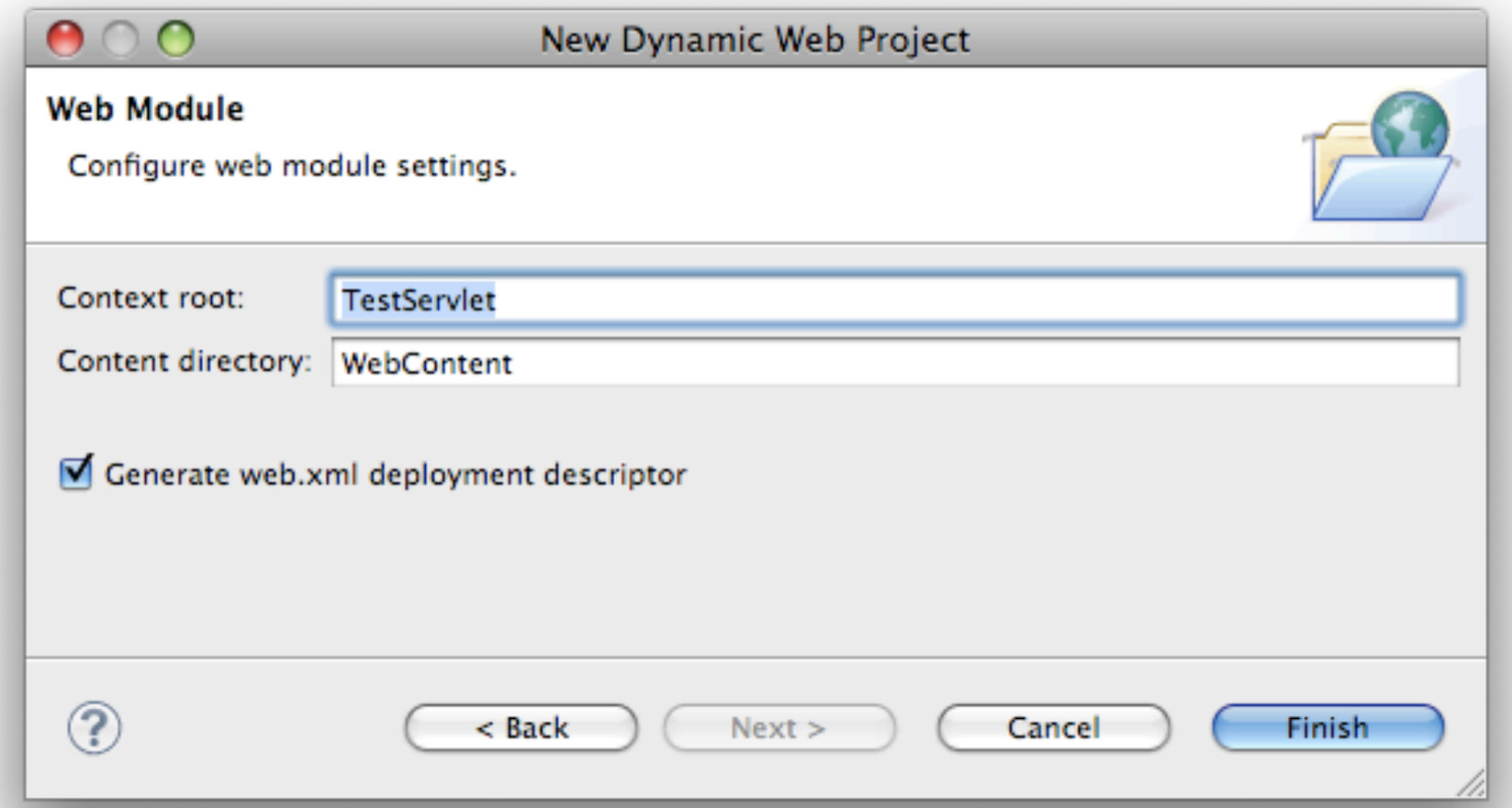


Creare un nuovo "Dynamic Web Project", specificando Tomcat 6.0 come server di riferimento (usare la configurazione di default del framework)

Servlet e Eclipse

Root e contenuto

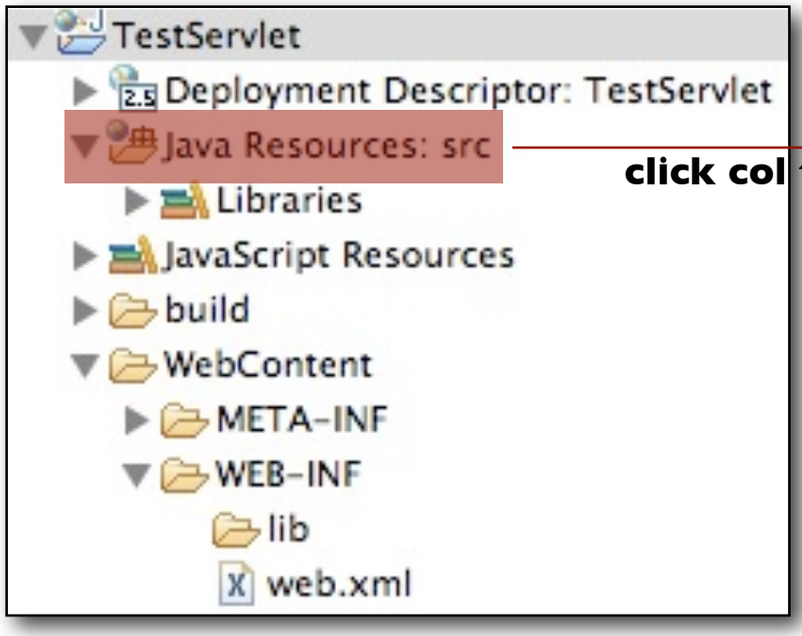
La *Context root* è il nome su cui il vostro progetto viene mappato all'interno del Web container – `http://[Web container]/[Context root]`



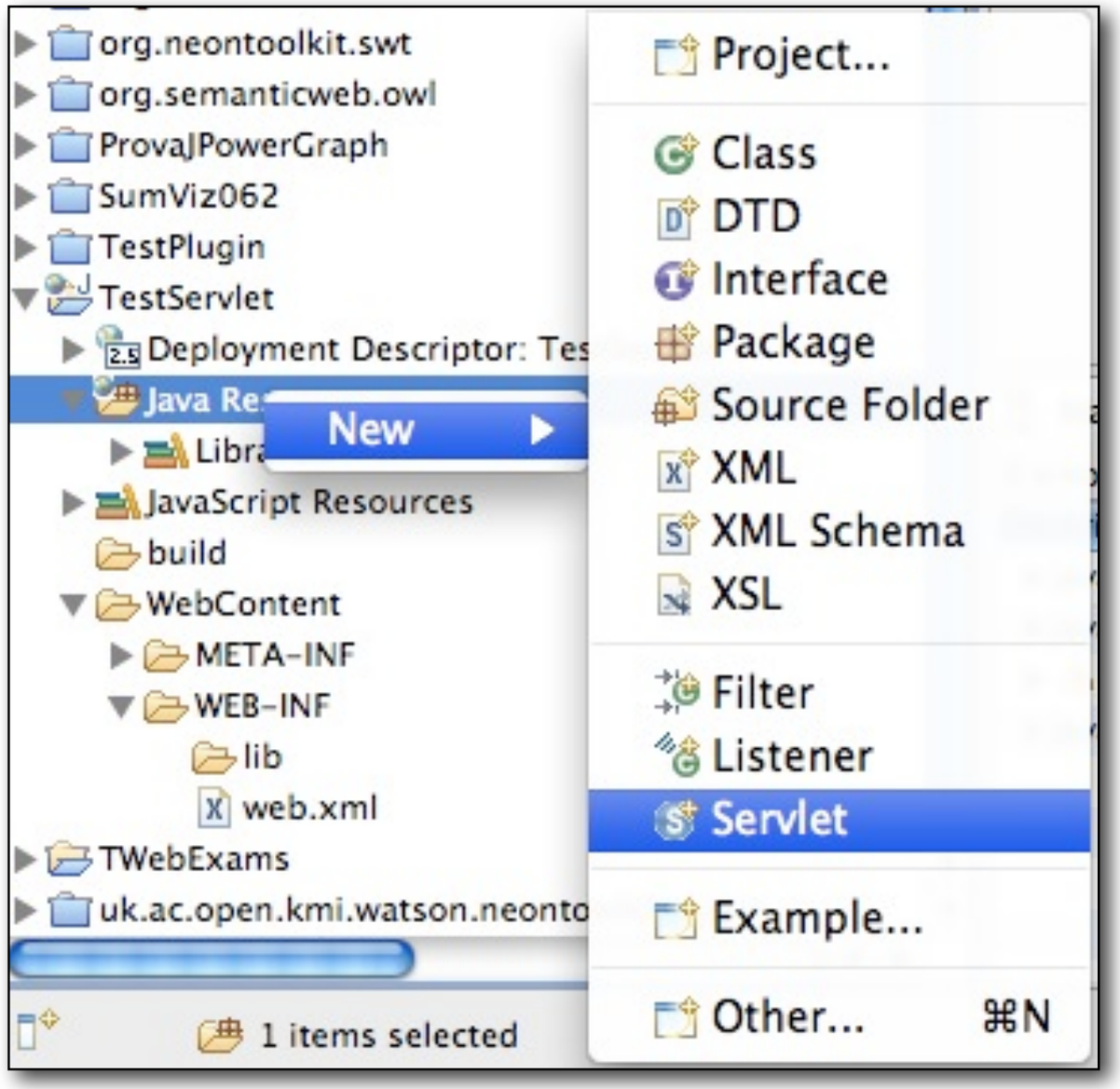
Content directory specifica quale debba essere il nome della cartella che contiene tutti i file relativi alla vostra applicazione Web (es: documenti HTML statici, file CSS, XSLT, ecc.)

Servlet e Eclipse

Creare una nuova servlet

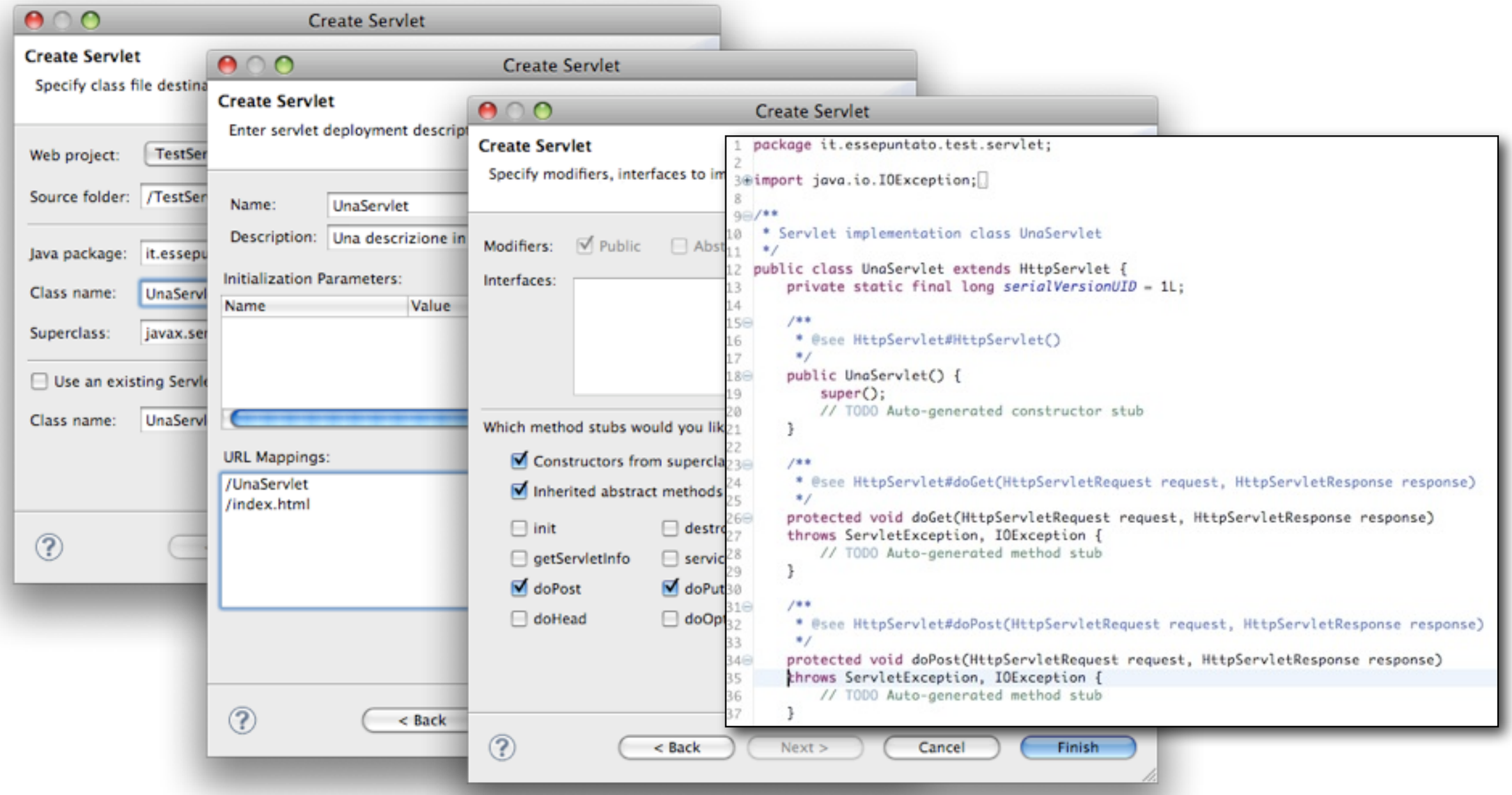


click col tasto destro



Servlet e Eclipse

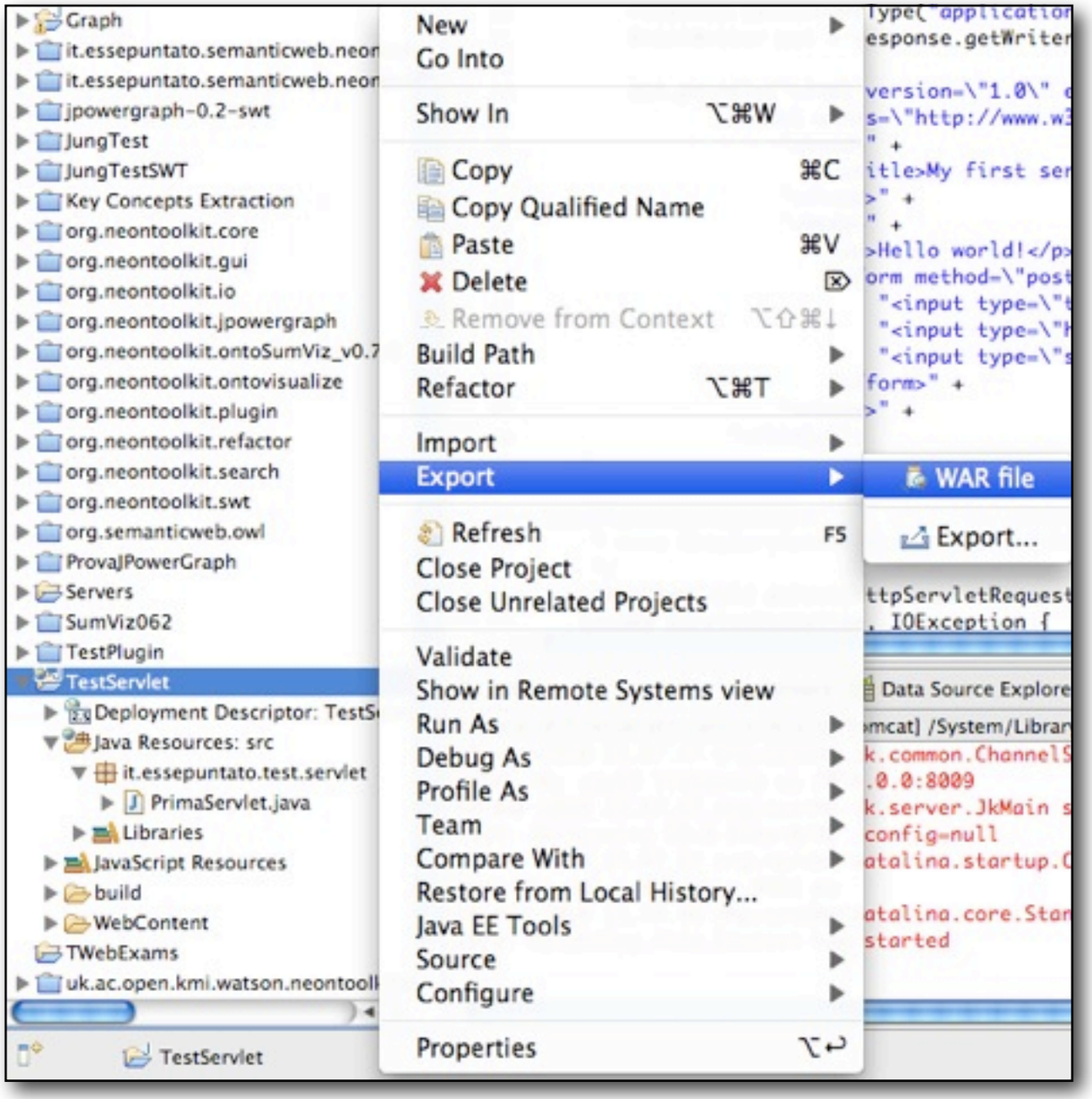
Classe, mapping, metodi



Creazione del War

- Una volta completato il progetto è necessario creare un *web application archive* (WAR), un pacchetto contenente tutto il progetto realizzato

- Va copiato nella cartella *webapps* di Tomcat in modo che possa essere caricato dal framework



Bibliografia

- *Servlet and JSP development with Eclipse WTP*, Lars Vogel – http://www.vogella.de/articles/EclipseWTP/article.html
- *Developing Web applications with Tomcat and Eclipse*, Nathan A. Good – http://www.ibm.com/developerworks/opensource/library/os-eclipse-tomcat/index.html

Bibliografia

- *Struts framework and the model-view-controller design pattern* IBM WebSphere Information Center, http://publib.boulder.ibm.com/infocenter/iadthelp/v7r0/index.jsp?topic=/com.ibm.etools.struts.doc/topics/cstrdoc001.html