

Anno Accademico 2007-2008

Laboratorio di Tecnologie Web

Sviluppo di applicazioni web  
JSP

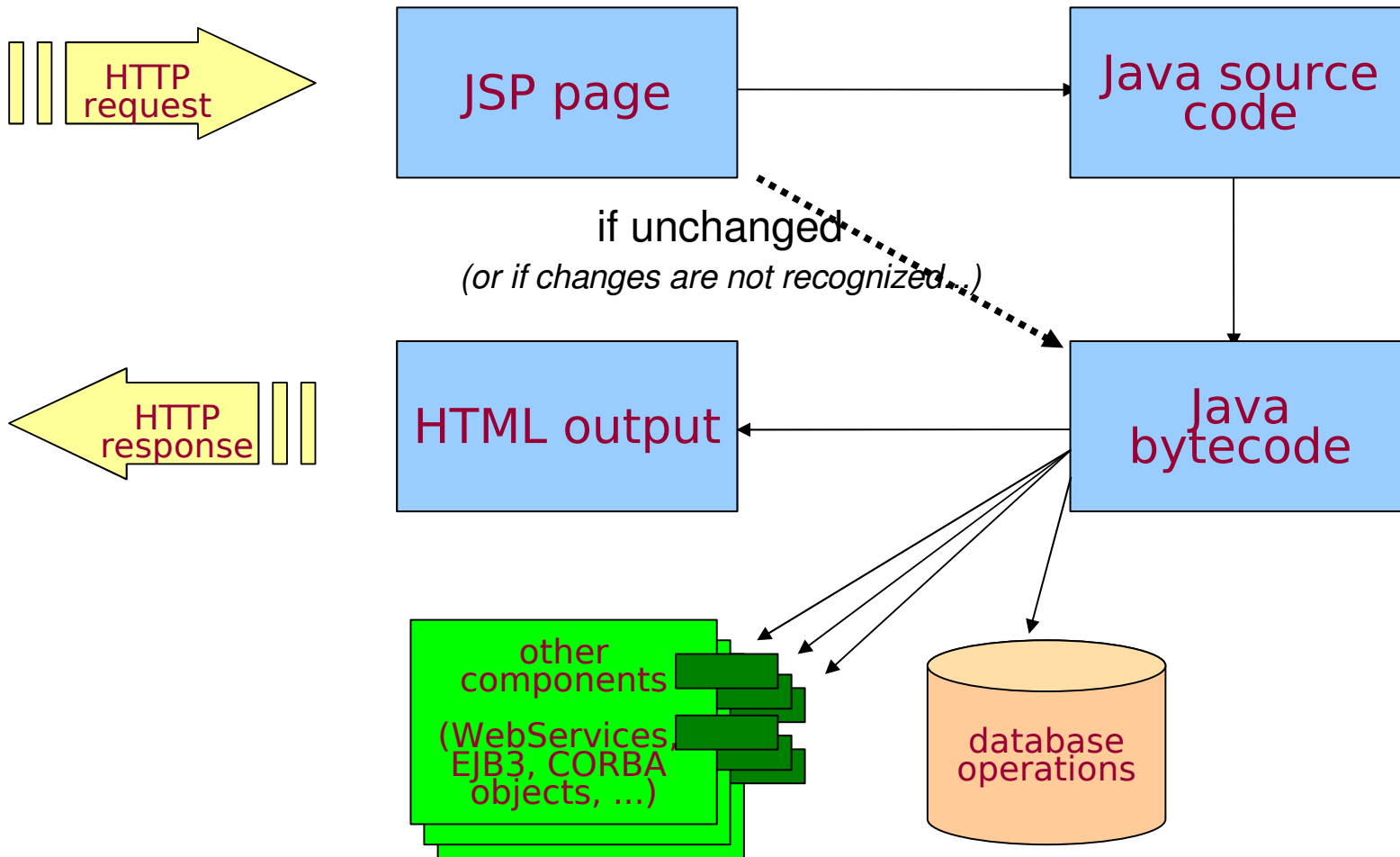
<http://www-lia.deis.unibo.it/Courses/TecnologieWeb0708/>

# Java Server Pages: fundamentals

---

- > HTML pages that embed Java code
- > Tomcat's **JspServlet** handles **\*.jsp** pages:
  - translation to full-fledged Java classes (**\*.java** source files) that extend **HttpServlet** class (through the **HttpJspBase** one, in Tomcat)
  - bytecode compilation (**\*.class** files)
  - request dispatching
- > Server executes Java code to produce
  - dynamic HTML pages
  - side effects (e.g., database modifications...)
- > Though they seem to be part of a scripting language, **JSP pages are not interpreted at run-time**, but compiled to traditional Java code

# Java Server Pages: how does it all work



# Java Server Pages: features and facilities

---

## > **Constructions** (to write JSP code):

- **Declarations:** `<%! %>` let define variables and methods for the Java class implementing the JSP page (you can think of it as equivalent to member stuff)
- **Directives:** `<%@ %>` let define page properties, code imports, and more...
- **Expression:** `<%= %>` evaluate Java expressions (notice: not instructions → no semicolon ';' in the end and result directly sent to the page output writer)
- **Scriptlet:** `<% %>` Java code (it gets evaluated as long as the page is rendered → but it is compiled before! if compiler fails the page is not shown at all)

## > **Embedded objects** (resources available across the page code)

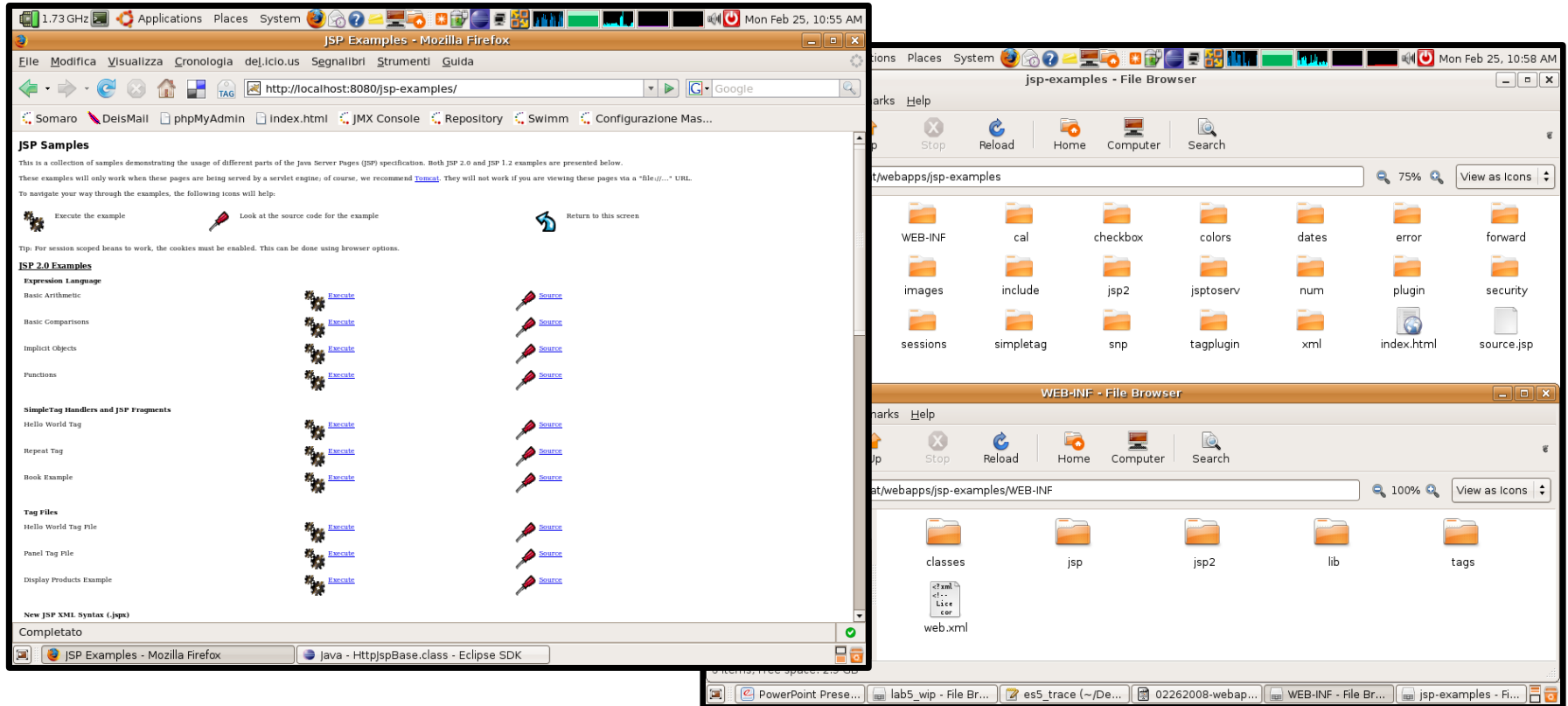
- `page` (and its properties), `out` (where to write HTML output), ...
- `request` (and its attributes/parameters), `response` (and its/properties), ...
- `session`, ...

## > **Tag libraries**

- reusable HTML fragments
- reusable libraries of tags (e.g., JSTL)
- custom Java components implementing custom tags logic

# JSP examples in Tomcat

> Similarly to Servlet examples, Tomcat also provides JSP example pages



> Anyway, having so many things altogether makes investigating uneasy

- not such a good starting point (too many descriptors and configurations...)
- rather, a place to find demonstrations of specific features you might need

# Another place to find your inspiration

## > Sun's J2EE tutorial

- provides several examples of Servlet, JSP (and JSF...) web applications
- also illustrates advanced features (e.g., Security) and subjects of matter that go beyond the scope of this course (i.e., WebServices, EJB3, ...)

yes, that's English..

## > Pay attention...

- code samples sometimes leverage Sun's Application Server specific features and configuration options (see the *sun-web.xml* files that come along with them)

# The template JSP project

## > Download it...

- ...from the course web site and import it in Eclipse (as usual)
- ...customize *environment.properties* (copy it from previous working projects)
- ...launch tomcat and deploy the web app on it

## > What's in there

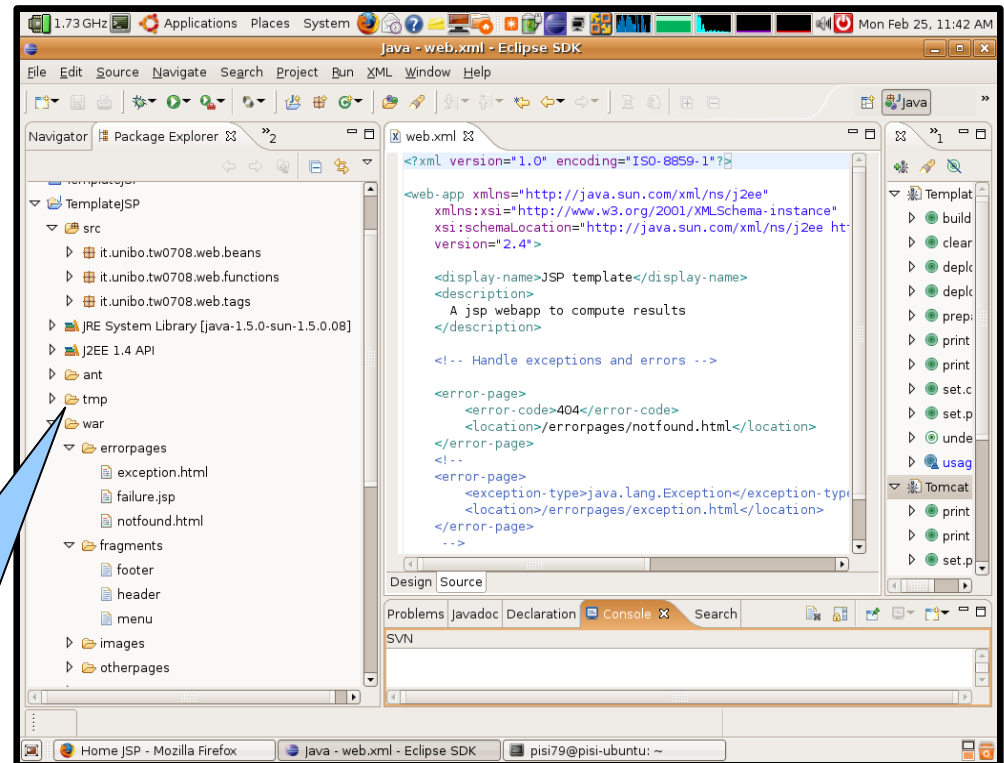
- half a dozen (maybe more...) sample pages to demonstrate features
- sample configuration files
- sample custom tag libraries and Java components

Build file has a few corrections, with respect to previous projects!

- **deploy targets** now include also files with no extension (\*\*/\*\* instead of \*\*/\*.\*)

- **master-classpath path** now includes all jars from the web server location (\*.jar instead of servlet\*.jar)

Thus, make sure you reuse this project for your web apps, not the previous ones!



# index.jsp

## > directives

- inclusion of HTML fragments (menu and footer)
- exception handling via custom pages

## > declarations (*member* methods and variables)

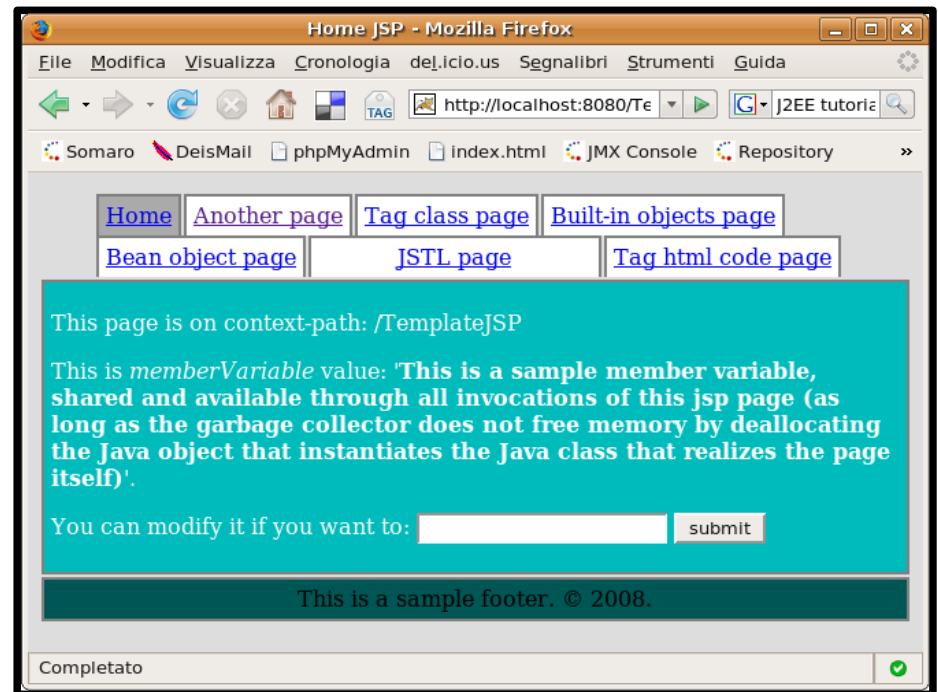
- set the value, browse other pages and come back
- browse from a different browser



- your value is still there (since the Java object performing the page service is still in memory, on the server!)

## > scriptlets

- generating content by invoking page methods and variables





# failure.jsp

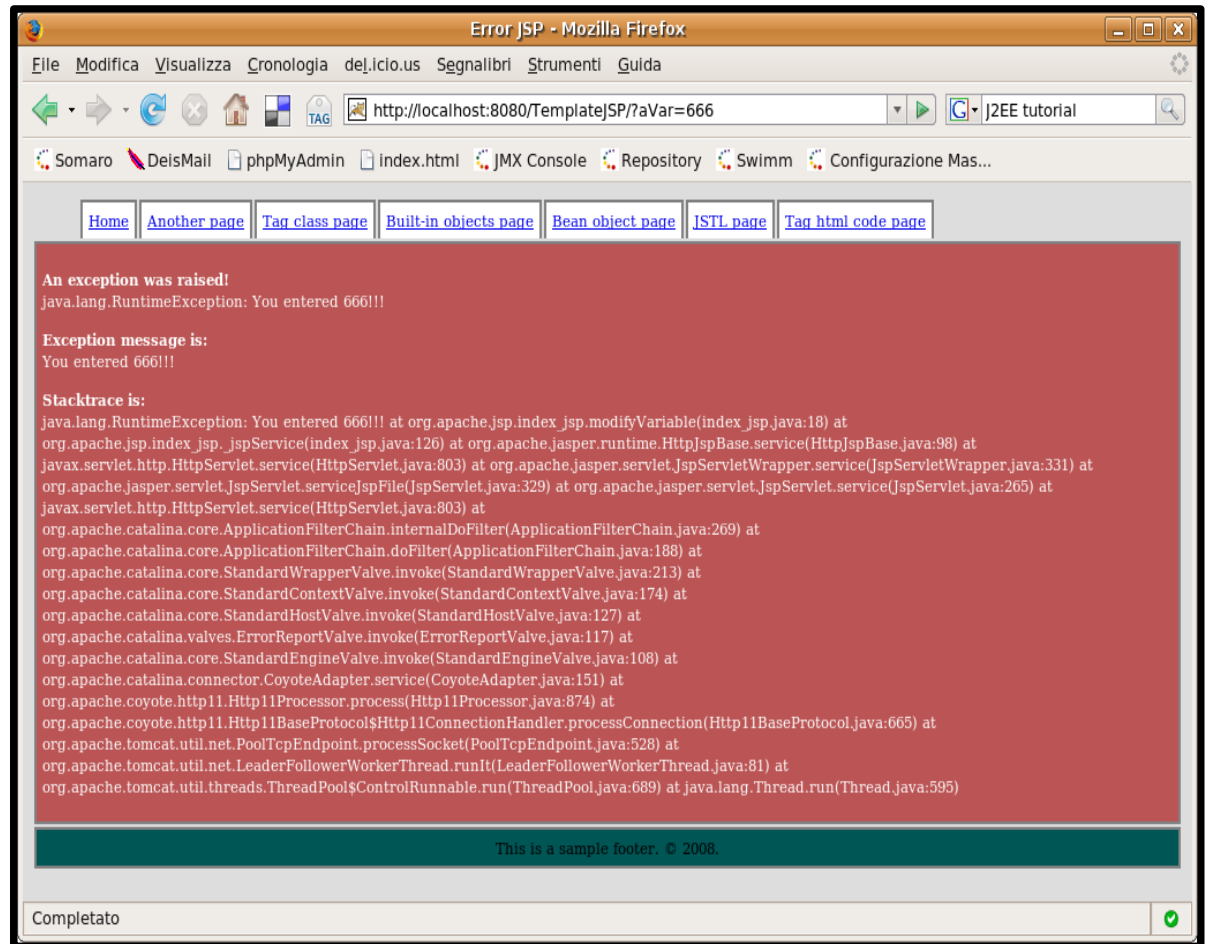
> force throwing an exception by typing '666' into *index.jsp* form



- Tomcat catches the exception and forwards request to the *failure.jsp* page

> **directives**

- this is an error page (setting *@page isErrorPage* to “*true*” gains access to the 'exception' built-in object)



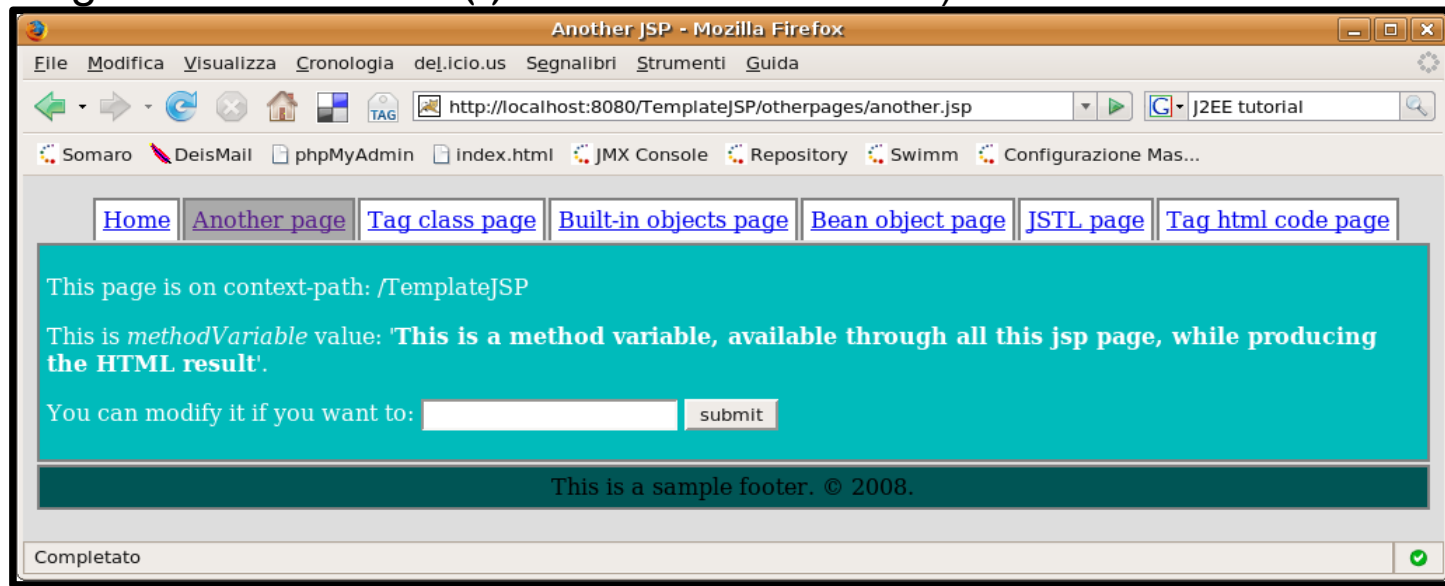
# another.jsp

## > Just using method variables

- JSP scriptlet (i.e., stuff within `<% %>`) is compiled to `HttpServlet.service()` method
- this is why you cannot define functions outside `<%! %>`: it would be like defining methods within one another (and you cannot do that in Java)



- variable value remains valid just as long as the page produces its output (i.e., as long as the `service()` method is executed)

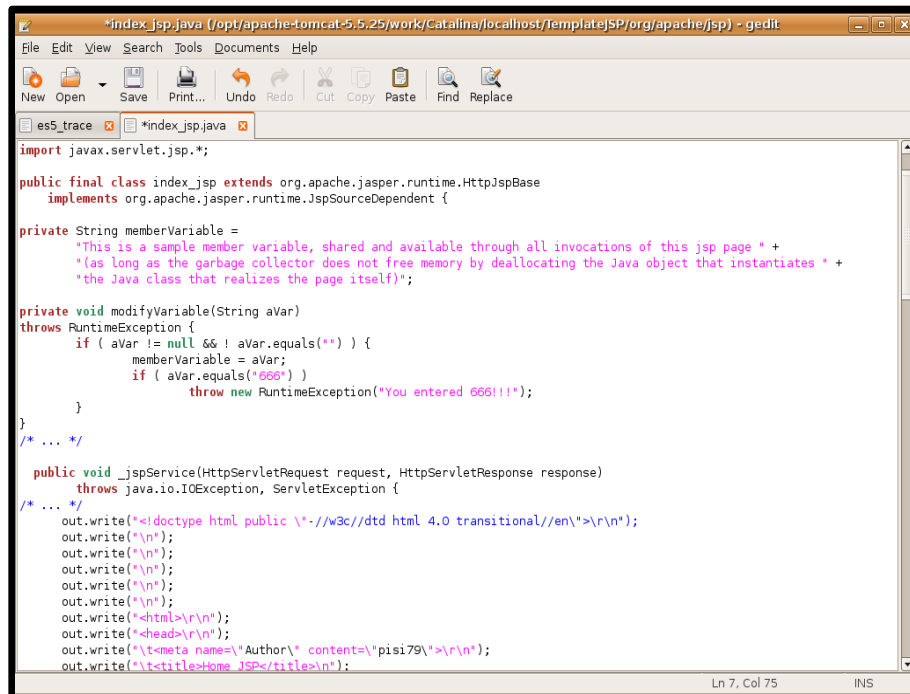


# index.jsp vs. another.jsp

> You can find the corresponding Java code (generated by Tomcat) by opening

*\$TOMCAT\_HOME/work/Catalina/localhost/TemplateJSP/org/apache/jsp*

> Compare sources of the two classes and find where `<%@ %>`, `<%! %>`, `<% %>`, and `<% %>` stuff is!



```
import javax.servlet.jsp.*;

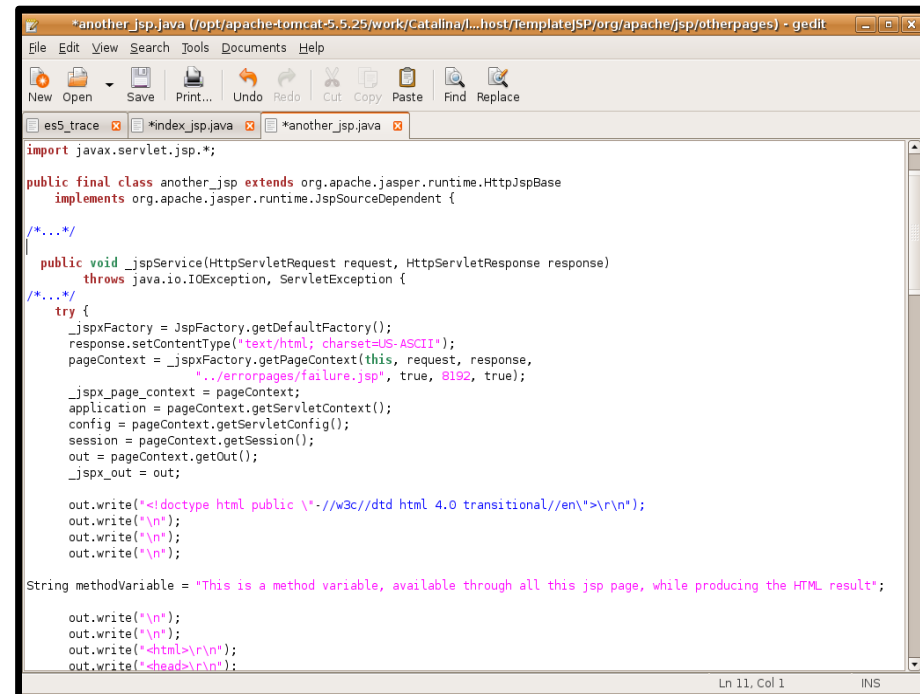
public final class index_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

    private String memberVariable =
        "This is a sample member variable, shared and available through all invocations of this jsp page " +
        "(as long as the garbage collector does not free memory by deallocating the Java object that instantiates " +
        "the Java class that realizes the page itself)";

    private void modifyVariable(String aVar)
    throws RuntimeException {
        if ( aVar != null && ! aVar.equals("") ) {
            memberVariable = aVar;
            if ( aVar.equals("666") )
                throw new RuntimeException("You entered 666!!!");
        }
    }
    /* ... */

    public void _jspService(HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, ServletException {

    /* ... */
        out.write("<doctype html public "-//w3c//dtd html 4.0 transitional//en">\r\n");
        out.write("\n");
        out.write("\n");
        out.write("\n");
        out.write("\n");
        out.write("\n");
        out.write("<html>\r\n");
        out.write("<head>\r\n");
        out.write("<meta name='Author' content='psi79'>\r\n");
        out.write("<title>Home JSP</title>\r\n");
    }
}
```



```
import javax.servlet.jsp.*;

public final class another_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

    /*...*/

    public void _jspService(HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, ServletException {

    /*...*/
        try {
            _jspxFactory = JspFactory.getDefaultFactory();
            response.setContentType("text/html; charset=US-ASCII");
            pageContext = _jspxFactory.getPageContext(this, request, response,
                "../errorpages/failure.jsp", true, 8192, true);

            _jspx_page_context = pageContext;
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();
            _jspx_out = out;

            out.write("<doctype html public "-//w3c//dtd html 4.0 transitional//en">\r\n");
            out.write("\n");
            out.write("\n");
            out.write("\n");

            String methodVariable = "This is a method variable, available through all this jsp page, while producing the HTML result";

            out.write("\n");
            out.write("\n");
            out.write("<html>\r\n");
            out.write("<head>\r\n");
        }
    }
}
```

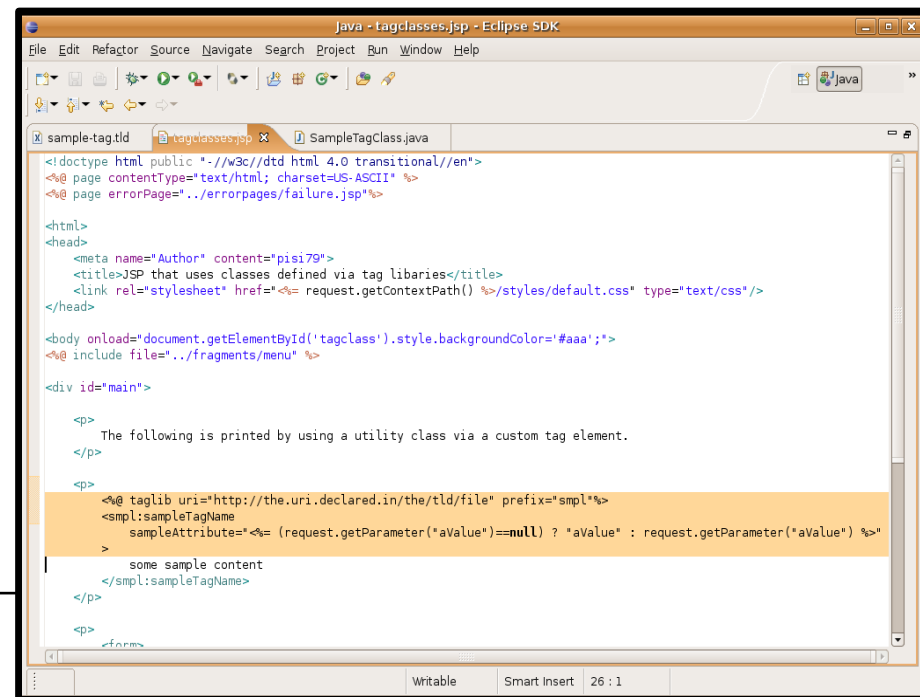
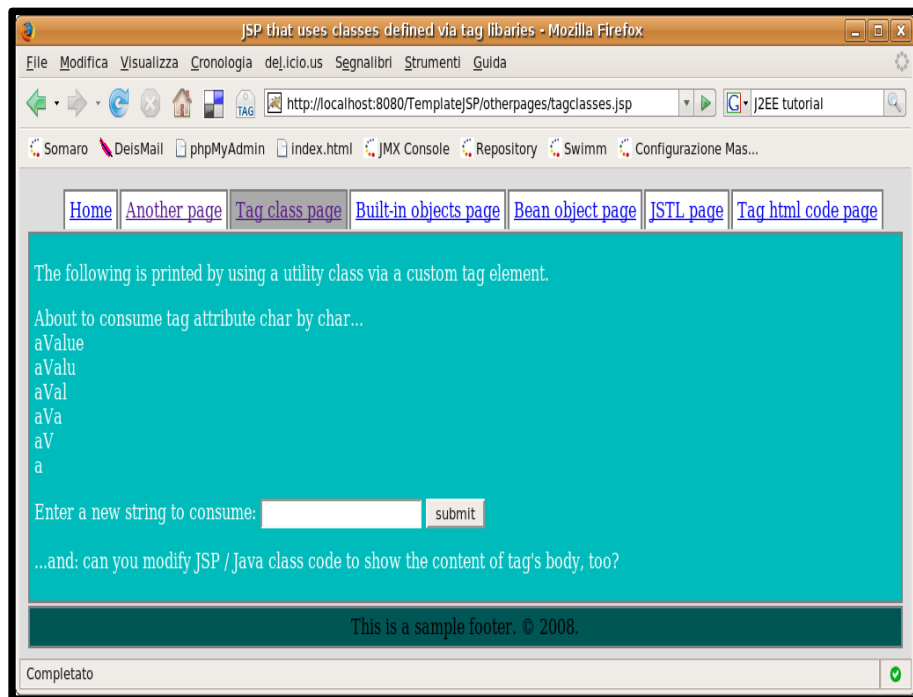
# tagclasses.jsp

## > Usage of JSP custom tags in the HTML code

- tags are defined in a *taglibrary*, outside the page
- prefix *simpl* denotes those tags in the page
- a URI identifies the taglibrary (*http://the.uri.declared.in/the/tld/file*)

## > *sampleTagName*

- prints out HTML code that repeats tag attribute several times on different lines, suppressing the last character each time
- in the page code, the attribute value is read from the HTTP request



# tagclasses.jsp

> *WEB-INF/tags/sample-tag.tld* file...

- reports the same URI used in the JSP page
- maps tags to Java classes

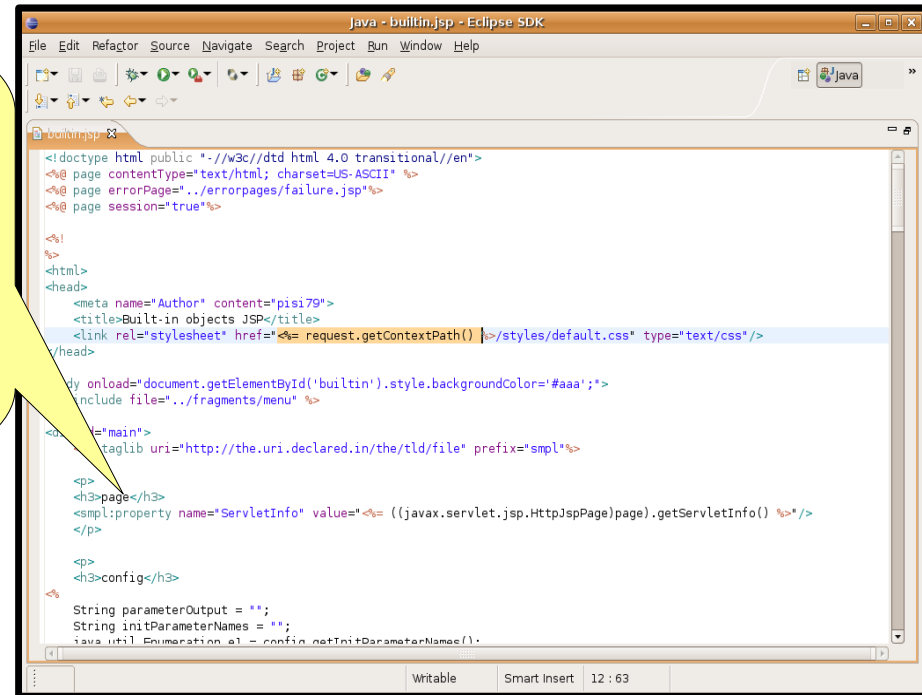
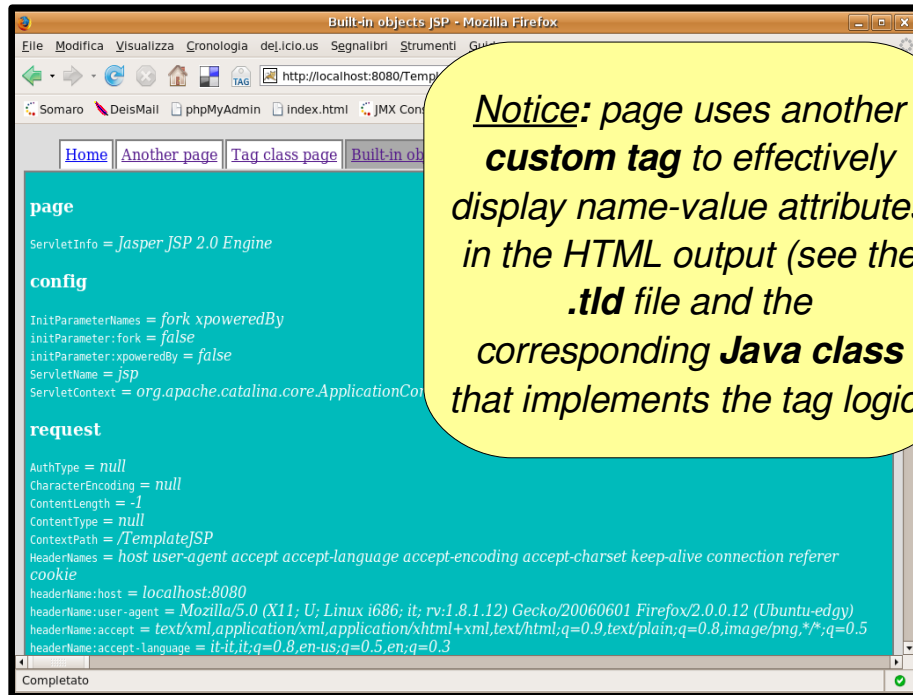
> *it.unibo.tw0708.web.tags.SampleTagClass* class...

- realizes logic and HTML output associated with the *sampleTagName* tag
- uses attributes and body of that tag to perform tasks / produce output



# builtin.jsp (1/2)

- > Just a sample page printing out properties from the available built-in objects
  - see the use of `page.contextPath()` to link styles no matter the current page location
  - try to invoke the page by adding request parameters to the page URL (*...?name1=value1&name2=value2...*)
  - sniff at the code to see what built-in objects offer



## builtin.jsp (2/2)

- > Try to modify page to leave cookies in the **response**...

e.g.,

```
<%  
    response.addCookie(  
        new Cookie(  
            "Cookie_" + System.currentTimeMillis(),  
            "" + new java.util.Random().nextInt()  
        )  
    );  
%>
```

- > ...or objects (for instance... string attributes) in the **session**:

e.g.,

```
<%  
    session.setAttribute(  
        "Attribute_" + System.currentTimeMillis(),  
        "" + new java.util.Random().nextDouble()  
    );  
%>
```

Does it seem  
that running  
**undeploy** /  
**deploy** targets  
is not making  
any difference?

Try using  
Tomcat-related  
ANT tasks to  
**reload** or even  
**stop / start**  
your web  
application!!



# bean.jsp

- > Page instantiates four objects of class *it.unibo.tw0708.web.beans.ABeanClass*
- every object is associated to one of the four different scopes of availability
  - page initializes object member values
  - page offers links to navigate to alternate pages handling these objects

The image displays three screenshots related to the `bean.jsp` page:

- Browser Output (Mozilla Firefox):** Shows the rendered page with navigation links: [Home](#), [Another page](#), [Tag class page](#), [Built-in objects page](#), [Bean object page](#), [JSTL page](#), and [Tag html code page](#). Below the links, four bean objects are listed with their names, classes, and property values:
  - Bean name: pageScopedBean  
Bean class: class it.unibo.tw0708.web.beans.ABeanClass  
Bean property1: p1Page  
Bean property2: p2Page
  - Bean name: requestScopedBean  
Bean class: class it.unibo.tw0708.web.beans.ABeanClass  
Bean property1: p1Request  
Bean property2: p2Request
  - Bean name: sessionScopedBean  
Bean class: class it.unibo.tw0708.web.beans.ABeanClass  
Bean property1: p1Session  
Bean property2: p2Session
  - Bean name: applicationScopedBean  
Bean class: class it.unibo.tw0708.web.beans.ABeanClass  
Bean property1: p1Application  
Bean property2: p2ApplicationAt the bottom, there are instructions: "Go to a [page](#) that can set these beans' values... or Go to [another page](#) that uses these beans without".
- Source Code (Eclipse SDK):** Shows the JSP code for `bean.jsp`. It includes a title, a link to a stylesheet, and four `<jsp:useBean>` tags, each followed by `<jsp:setProperty>` tags to initialize the bean's properties. The beans are created with scopes `page`, `request`, `session`, and `application`.
- Class Definition (Eclipse SDK):** Shows the source code for the `ABeanClass` class. It is a public class with a constructor, two private fields (`property1` and `property2`), and four public methods: `getProperty1()`, `setProperty1()`, `getProperty2()`, and `setProperty2()`.



# bean1.jsp, bean2.jsp, bean3.jsp

---

## > *bean1.jsp*

- linked by *bean.jsp* page
- exposes forms to modify bean attribute values
- see how session- and application-scoped beans remain valid across pages

## > *bean2.jsp*

- linked by both *bean.jsp* and *bean1.jsp* pages
- just shows bean attribute values, where available
- since it does not take part in session (by setting the *@page session attribute* to *false*) it cannot declare the session-scoped bean at all
- try to uncomment code that uses it → JSP compiler error (why not showing *failure.jsp*?)

## > *bean3.jsp*

- linked by two of the four form actions in *bean1.jsp* page
  - redirects requests to *bean2.jsp* page (though URL in the browser does not show that → see *bean3.jsp* source to find the *<jsp:forward>* action)
  - tells the difference between page-scoped and request-scoped bean
-

## jstl.jsp (1/2)

---

- > An example of using **tags from the JSP Standard Tag Library**...
  - in this case: `<c:forEach var="item" items="{cart.items}">`
- > ...a custom **tag library defining functions**...
  - associated to URI `http://it.unibo.tw0708/tld/function`
  - mapped by `WEB-INF/tags/sample-function.tld`
  - implemented by the public static methods in class `it.unibo.tw0708.web.functions.Discount`
- > ...JSP **expression language** notation ( `{ ... }` )
  - to access bean properties in the HTML code
  - to invoke tag library functions in the HTML code
- > ...few other things we have already seen...
  - a session-scoped 'cart' bean, of class `it.unibo.tw0708.web.beans.Cart` that holds items selected by the user
  - request's context path being used to 'absolutely' reference styles
  - JSP error pages, inclusion of external HTML fragments
- > ...and lot of **HTML-embedded styling** → see how pages can easily become **messy!!!**

## jstl.jsp (2/2)

### > Page structure:

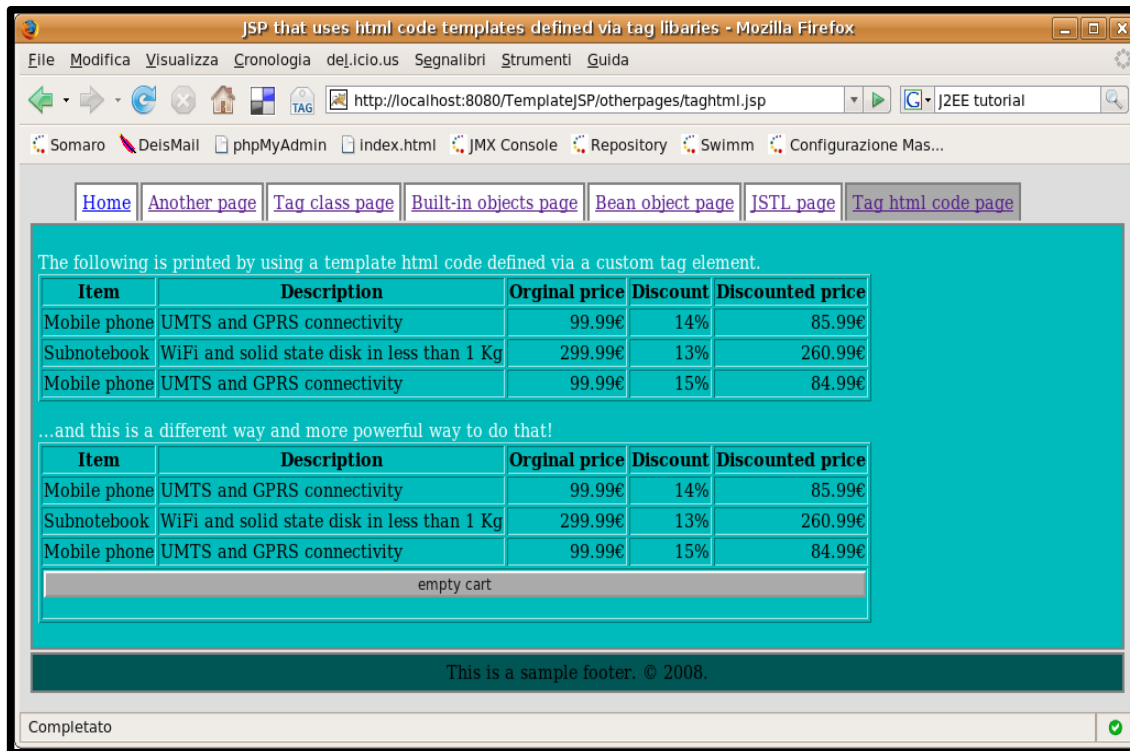
- A **table** holds form input fields
- Each discount value is calculated by invoking a custom **function**
- Form sends selected items along with a new action **request** to the current page
- Request parameters are read to add **items** in the **session-scoped cart**
- The 'empty' parameter, if present, is handled by emptying the cart
- **JSTL foreach tag** is used to iterate over cart items within the table element
- Another form holds the 'empty' submit button and asks for confirmation to the user via a **Javascript popup**

The screenshot shows a Mozilla Firefox browser window displaying a JSP page titled "JSP that uses the Standard Tag Library - Mozilla Firefox". The page URL is "http://localhost:8080/TemplateJSP/otherpages/jstl.jsp". The page content includes a navigation menu with links like "Home", "Another page", "Tag class page", "Built-in objects page", "Bean object page", "JSTL page", and "Tag html code page". Below the menu, there is a section titled "Please, check the article you want to buy!" containing a table with columns "Item", "Description", "Price", and "Discount". The table lists three items: "DVD Player" (199.99€ with 20% discount), "Subnotebook" (299.99€ with 17% discount), and "Mobile phone" (99.99€ with 18% discount). Below the table is an "add to cart" button. Underneath, there is a section titled "Your cart (printed by using JSP Standard Tag Library):" with a table showing "Mobile phone" (99.99€ original price, 16% discount) and an "empty cart" button. A JavaScript confirmation popup is visible in the foreground, titled "La pagina sul server http://localhost:8080 rip" and asking "Are you sure?" with "Annulla" and "OK" buttons. The footer of the page reads "This is a sample footer. © 2008." and the browser status bar shows "Completato".

# taghtml.jsp (1/3)

> This last page...

- ...iterates over items in cart (which is a session-scoped bean: still available)...
- ...in two different ways...
- ...by leveraging **tags that correspond to external and parameterized HTML fragments...**



The screenshot shows a Mozilla Firefox browser window titled "JSP that uses html code templates defined via tag libraries - Mozilla Firefox". The address bar shows "http://localhost:8080/TemplateJSP/otherpages/taghtml.jsp". The page content includes a navigation menu with links like "Home", "Another page", "Tag class page", "Built-in objects page", "Bean object page", "JSTL page", and "Tag html code page". The main content area has a teal background and contains two identical tables. The first table is preceded by the text "The following is printed by using a template html code defined via a custom tag element." and the second table is preceded by "...and this is a different way and more powerful way to do that!". Below the second table is a grey box labeled "empty cart". The footer of the page reads "This is a sample footer. © 2008." and the status bar at the bottom says "Completato".

Item	Description	Original price	Discount	Discounted price
Mobile phone	UMTS and GPRS connectivity	99.99€	14%	85.99€
Subnotebook	WiFi and solid state disk in less than 1 Kg	299.99€	13%	260.99€
Mobile phone	UMTS and GPRS connectivity	99.99€	15%	84.99€

Item	Description	Original price	Discount	Discounted price
Mobile phone	UMTS and GPRS connectivity	99.99€	14%	85.99€
Subnotebook	WiFi and solid state disk in less than 1 Kg	299.99€	13%	260.99€
Mobile phone	UMTS and GPRS connectivity	99.99€	15%	84.99€

- ...whose **definitions** (*\*.tag* files) **can be found at the location** (*WEB-INF/tags/*) **specified by the JSP page itself**
- ...and whose **tag names correspond to the \*.tag file names** at that location

## taghtml.jsp (2/3)

- > The first way exploits *WEB-INF/tags/cartRow.tag* file and the *cartRow* custom tag
  - tag prints out a 5-columns table row
  - tag attributes are mapped to body contents of the table data elements

```
Java - taghtml.jsp - Eclipse SDK
File Edit Refactor Source Navigate Search Project Run Window Help
taghtml.jsp cartRow.tag
<%@ taglib uri="http://it.unibo.it/08/tld/function" prefix="htmltags" %>
<%@ taglib prefix="htmltags" tagdir="/WEB-INF/tags" %>
<table border="1">
  <tr>
    <th>Item</th><th>Description</th><th>Original price</th>
  </tr>
  <c:forEach var="item" items="${cart.items}">
    <htmltags:cartRow
      name="${item.name}"
      description="${item.description}"
      originalprice="${item.price}"
      discount="${item.discount}"
      price="${myfn:discount(item.price,item.discount)}"
    </htmltags:cartRow>
  </c:forEach>
</table>
```

```
Java - cartRow.tag - Eclipse SDK
File Edit Refactor Source Navigate Search Project Run Window Help
taghtml.jsp cartRow.tag
<%@ attribute name="name" %>
<%@ attribute name="description" %>
<%@ attribute name="originalprice" %>
<%@ attribute name="discount" %>
<%@ attribute name="price" %>
<tr>
  <td>${name}</td>
  <td>${description}</td>
  <td style="text-align: right">${originalprice}&#8364;</td>
  <td style="text-align: right">${discount}%</td>
  <td style="text-align: right">${price}&#8364;</td>
</tr>
```

## taghtml.jsp (3/3)

- > The second way lets the custom *cartTable* tag perform also the item iteration!
  - ...custom tag imports JSTL tag library and our function tag library
  - ...attributes are not only of string type, this time:
    - one is the HTML fragment to produce the 'empty' button in the last row
    - the other one is the whole Java cart object!

```
Java - taghtml.jsp - Eclipse SDK
File Edit Refactor Source Navigate Search Project Run Window Help
</p>
<htmltags:cartTable cart="${cart}">
  <jsp:attribute name="lastrow">
    <form action="taghtml.jsp" method="get">
      <input type="submit" value="empty cart" name="...
    </form>
  </jsp:attribute>
  <jsp:body>
    ...and this is a different way and more powerful w
  </jsp:body>
</htmltags:cartTable>
</div>
Writable Smart Insert 1 : 1

Java - cartTable.tag - Eclipse SDK
File Edit Refactor Source Navigate Search Project Run Window Help
cartTable.tag taghtml.jsp
<@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<@ taglib uri="http://it.unibo.tw0708/tld/function" prefix="myfn" %>
<@ attribute name="lastrow" fragment="true" %>
<@ attribute name="cart" type="it.unibo.tw0708.web.beans.Cart" %>
<p>
  <jsp:doBody/>
  <table border="1">
    <tr>
      <th>Item</th><th>Description</th><th>Original price</th><th>Discount</th><th>Discounted price</th>
    </tr>
    <c:forEach var="item" items="${cart.items}">
      <tr>
        <td>${item.name}</td>
        <td>${item.description}</td>
        <td style="text-align: right">${item.price}&#8364;</td>
        <td style="text-align: right">${item.discount}%</td>
        <td style="text-align: right">${myfn:discount(item.price,item.discount)}&#8364;</td>
      </tr>
    </c:forEach>
    <tr>
      <td colspan="5">
        <jsp:invoke fragment="lastrow" />
      </td>
    </tr>
  </table>
</p>
Writable Smart Insert 1 : 1
```