Universita’ degli Studi di Bologna
Facolta’ di Ingegneria

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:

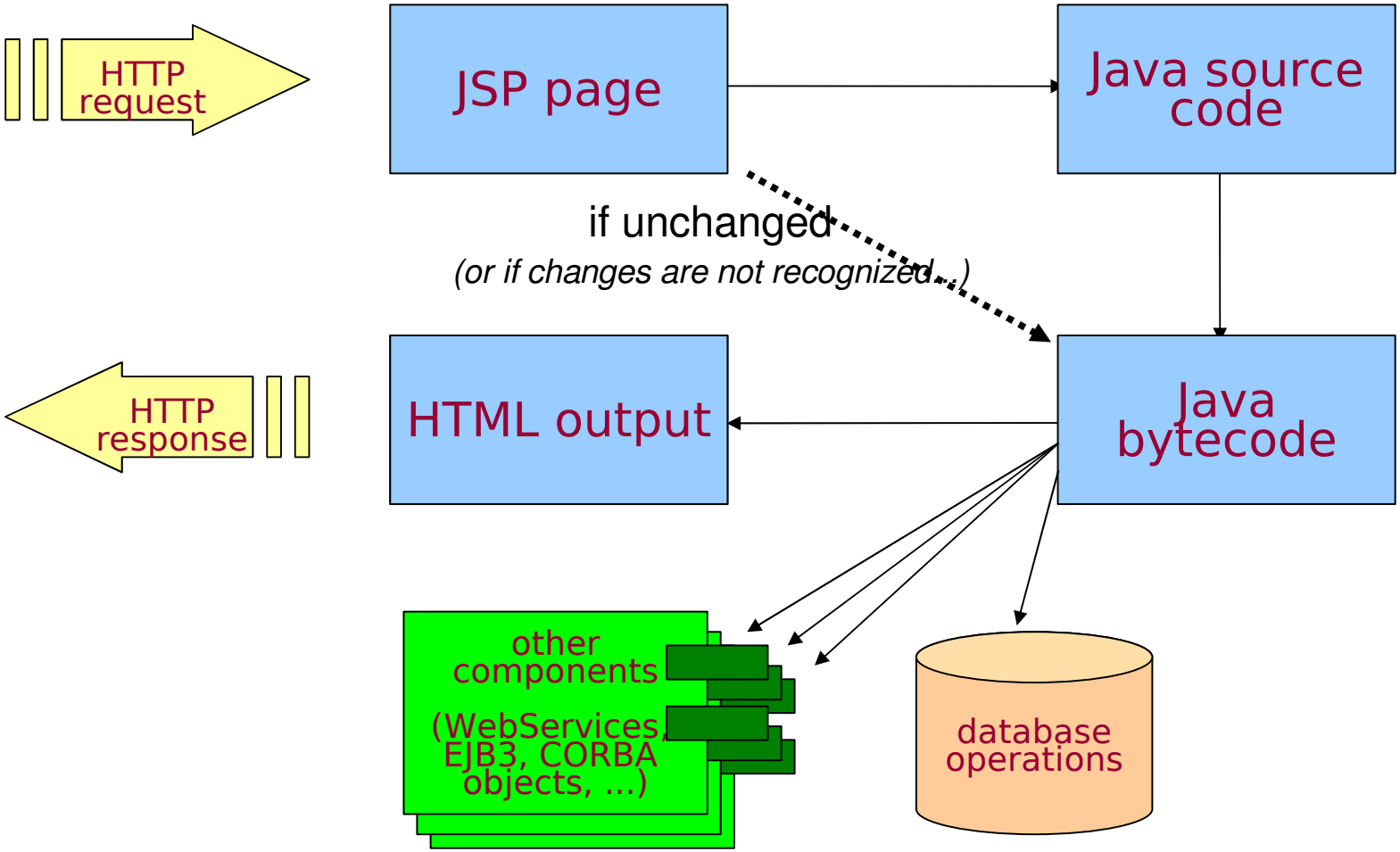
- translation to full-fledged Java classes (****.java*** source files) that extend **`HttpServlet`** class (through the **`HttpJspBase`** one, in Tomcat)
- bytecode compilation (****.class*** files)
- request dispatching

> 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)

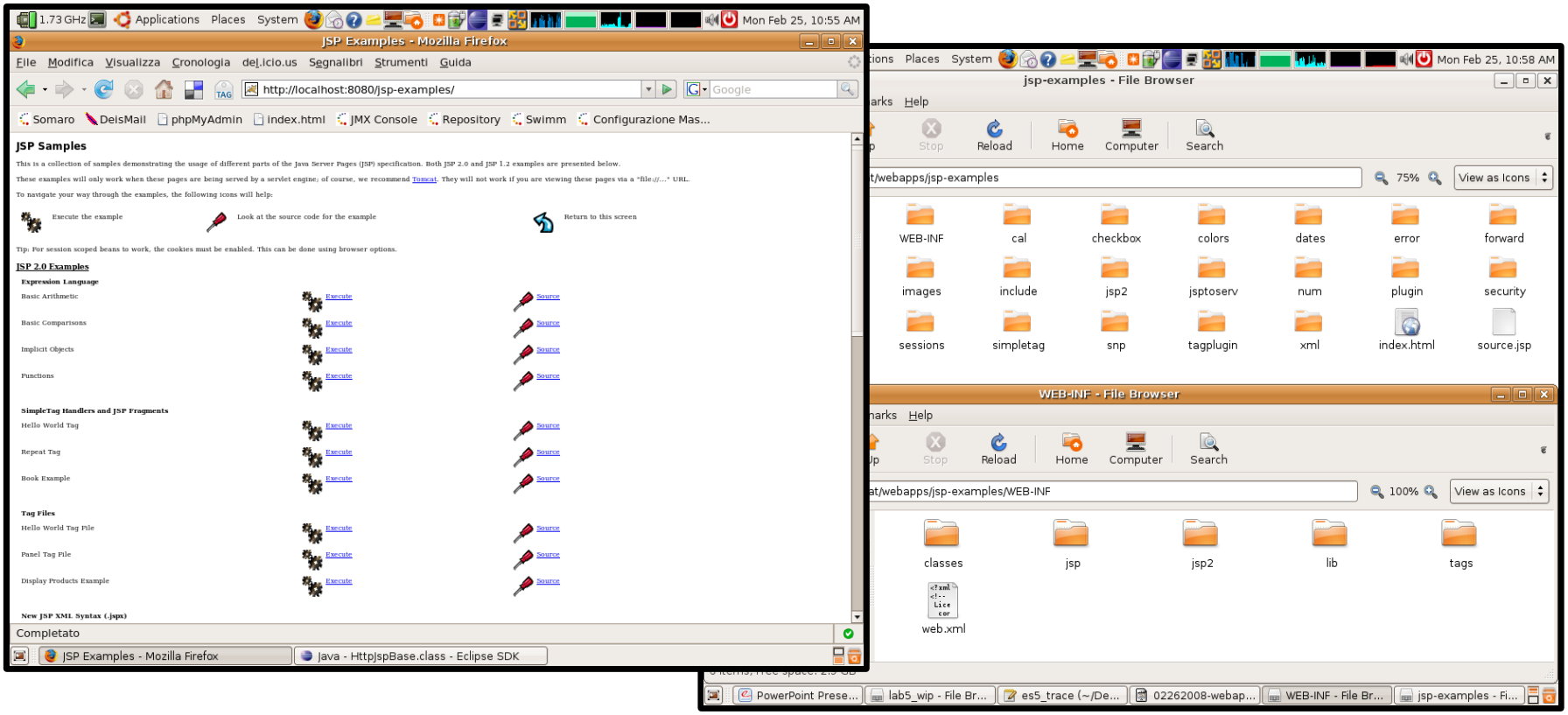
- page (and its properties), out (where to write HTML output), ...
- request (and its attributes/parameters), response (and its/properties), ...
- session, ...

> **Tag libraries**

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

JSP examples in Tomcat

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



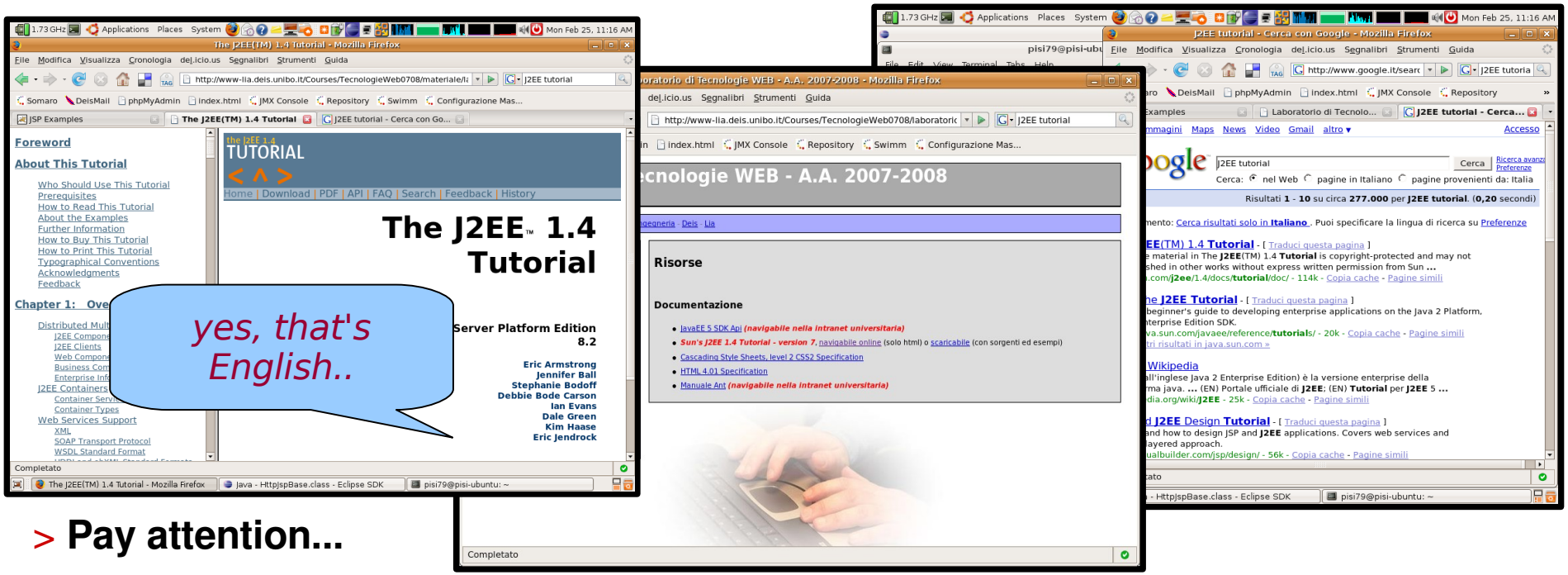
> Anyway, having so many things altogether makes investigating uneasy

- 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 Servet, 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, ...)



> **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)

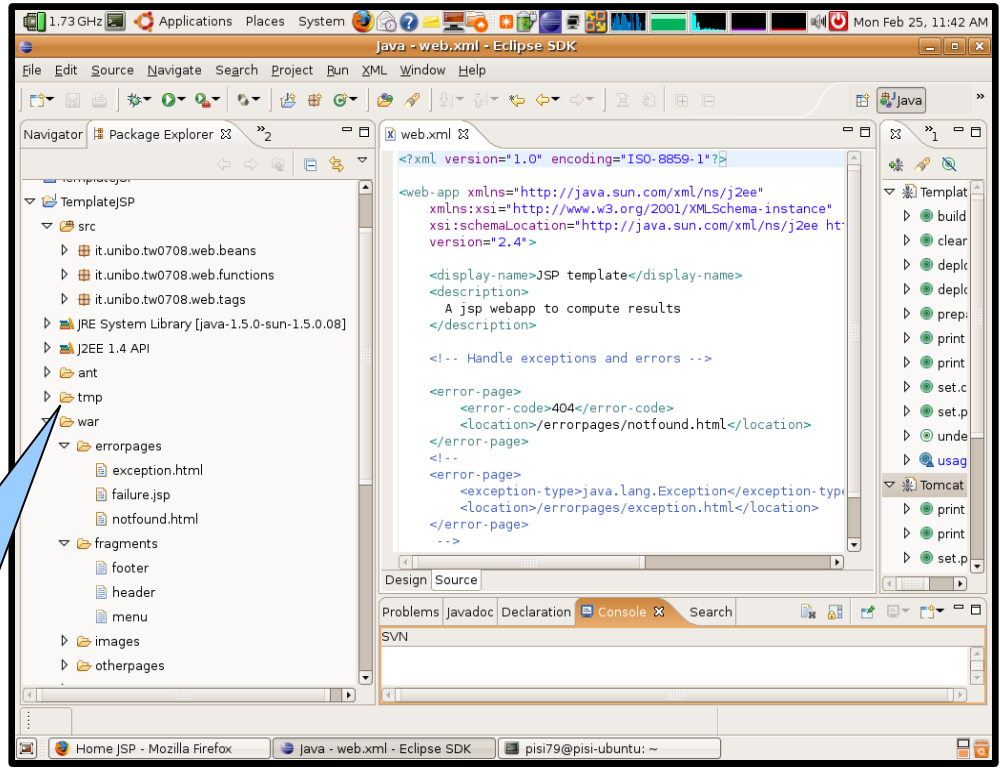
The template JSP project

> **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

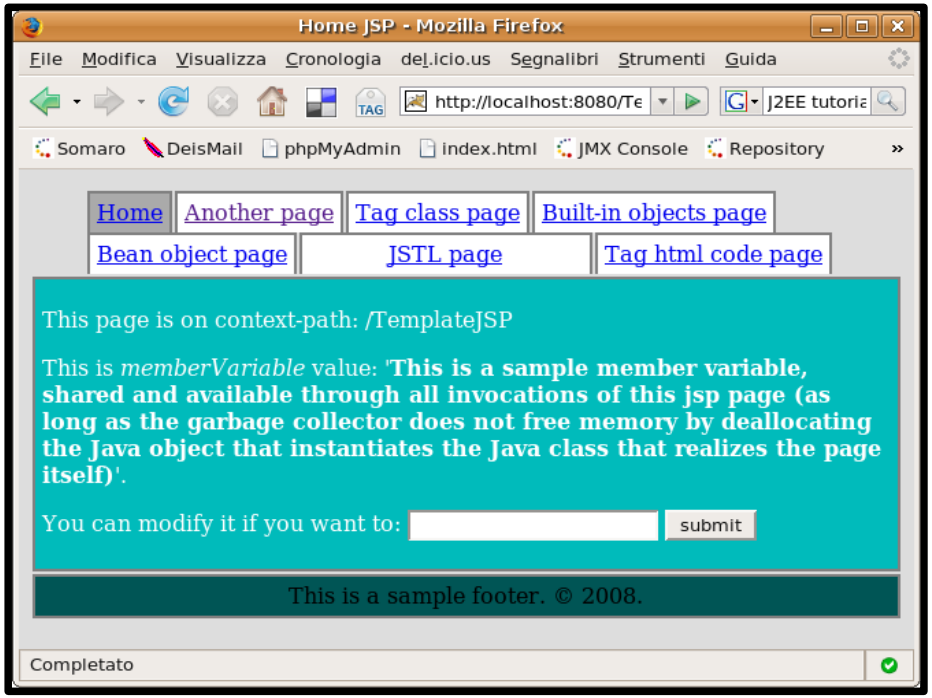
> **directives**
  - inclusion of HTML fragments (menu and footer)
  - exception handling via custom pages

> **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

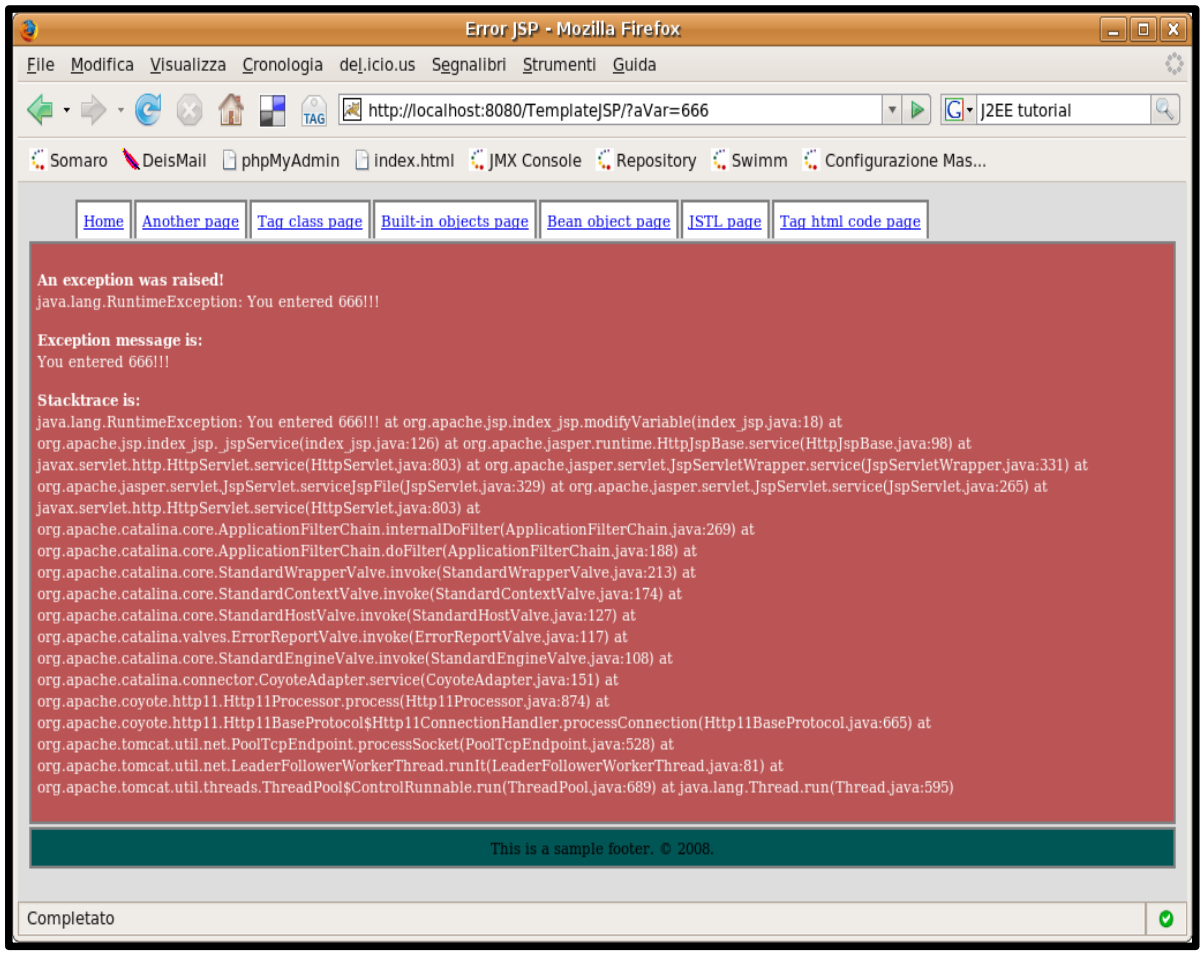
> force throwing an exception by typing '*666*' into *index.jsp* form

↓

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

> **directives**

- this is an error page (settiing *@page isErrorPage* to "*true*" gains access to the '`exception`' built-in object)



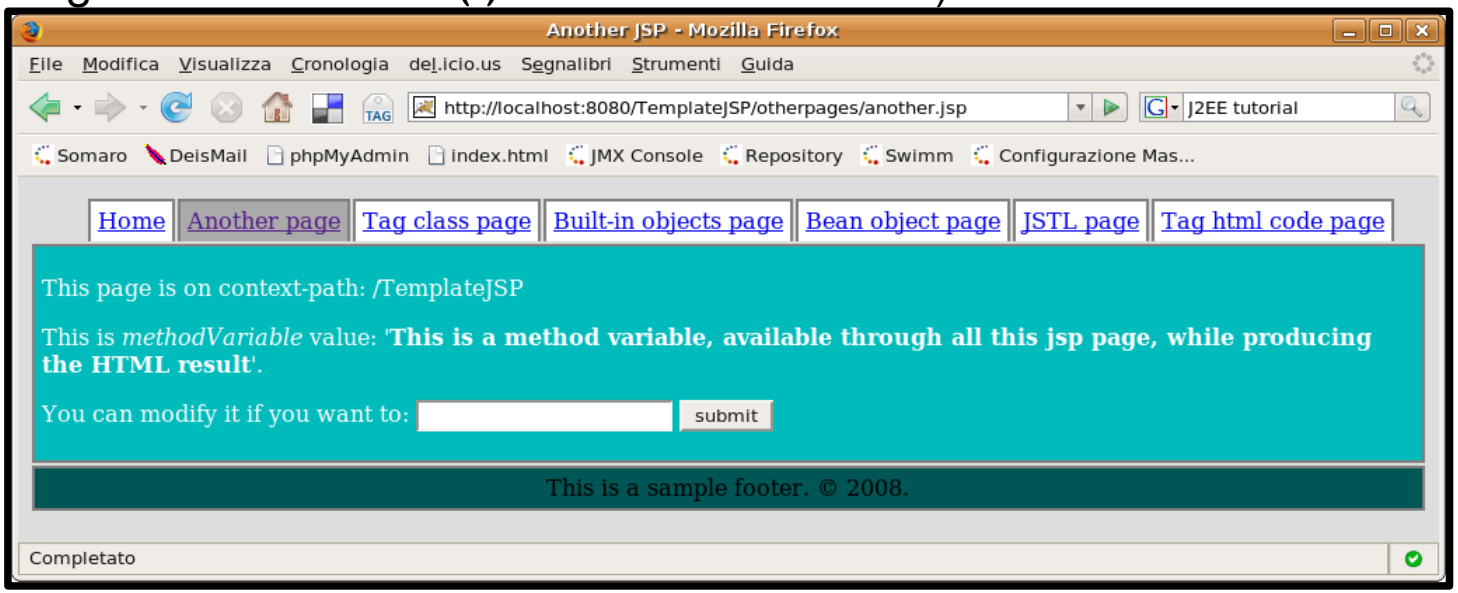
another.jsp

> 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)

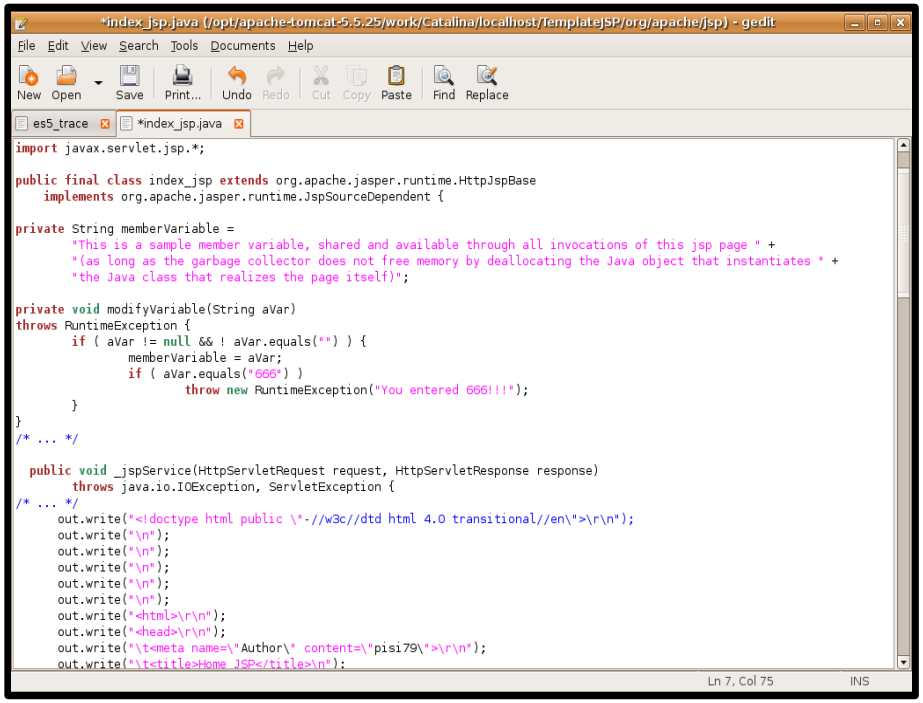


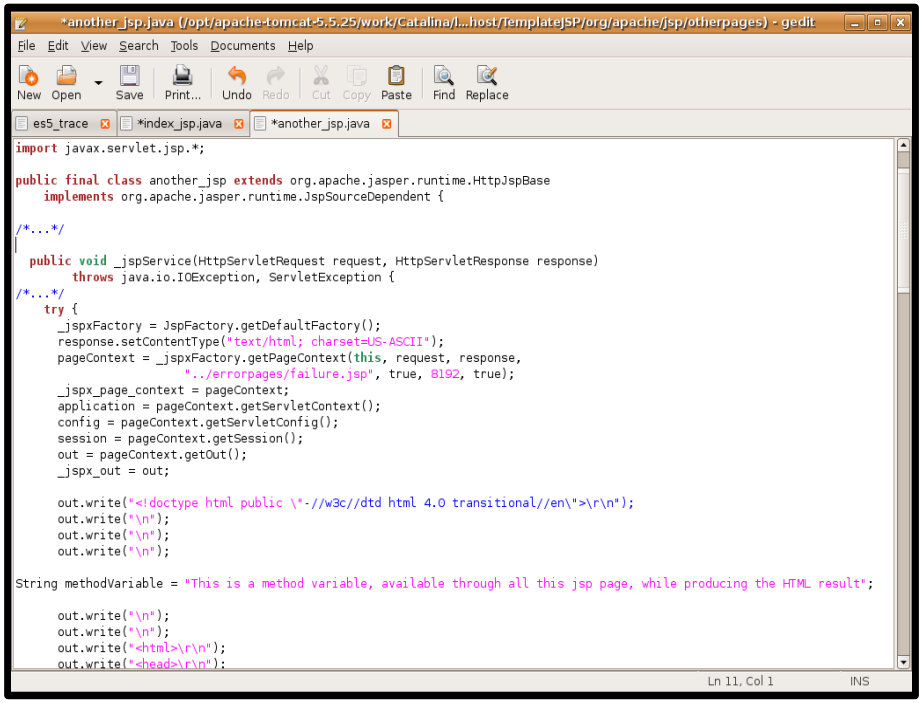
index.jsp *vs.* another.jsp

> 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!

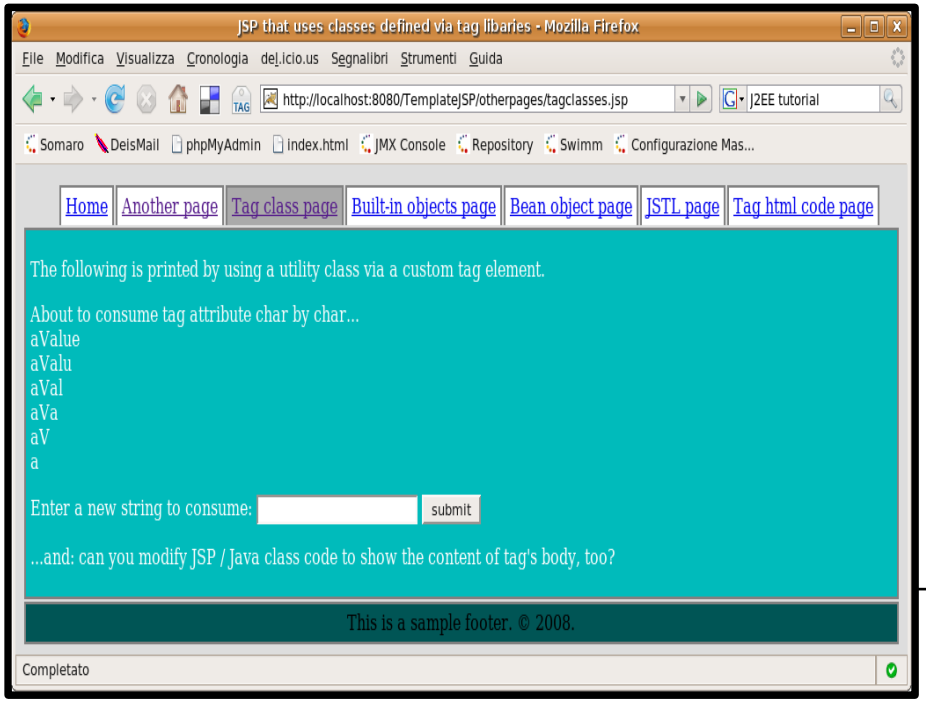


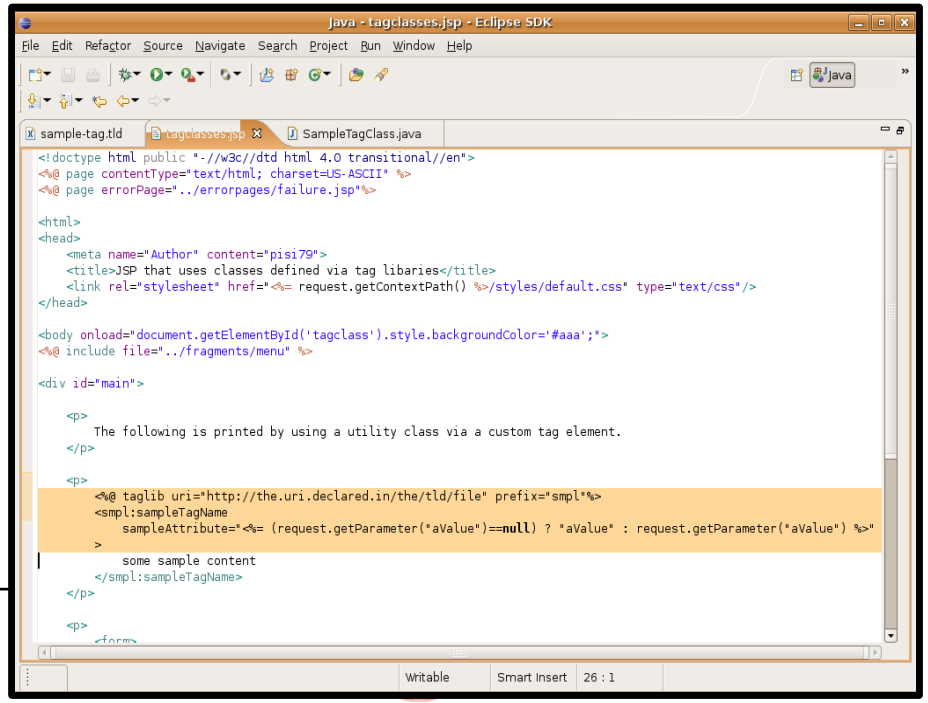


tagclasses.jsp

> Usage of JSP custom tags in the HTML code
  - tags are defined in a *taglibrary*, outside the page
  - prefix *smpl* 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

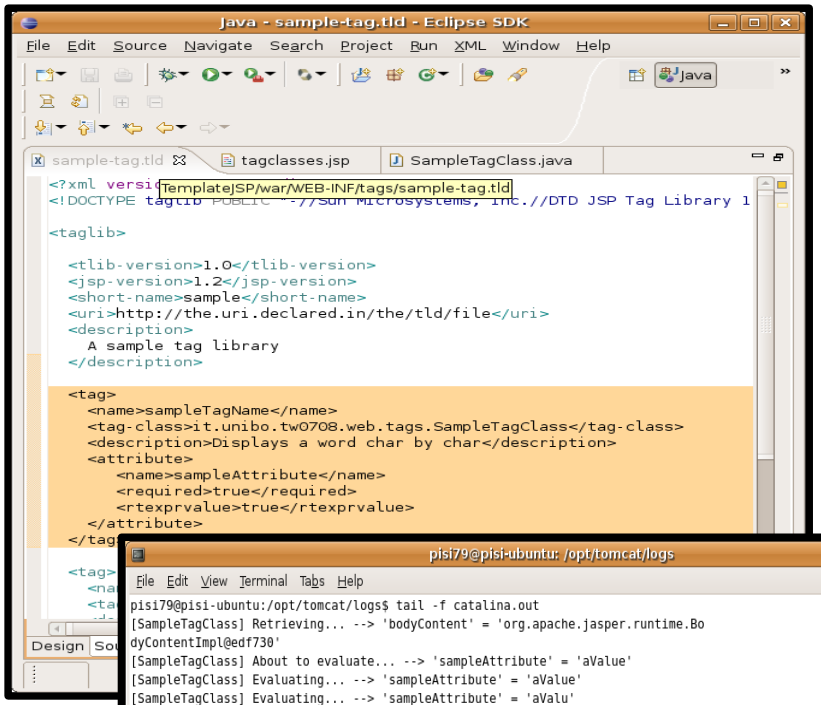


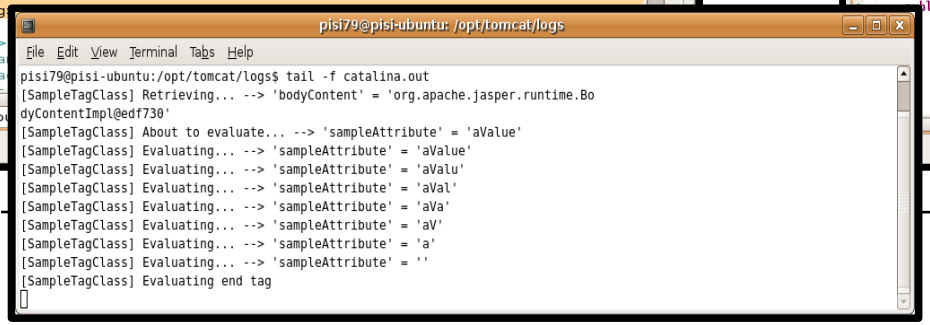


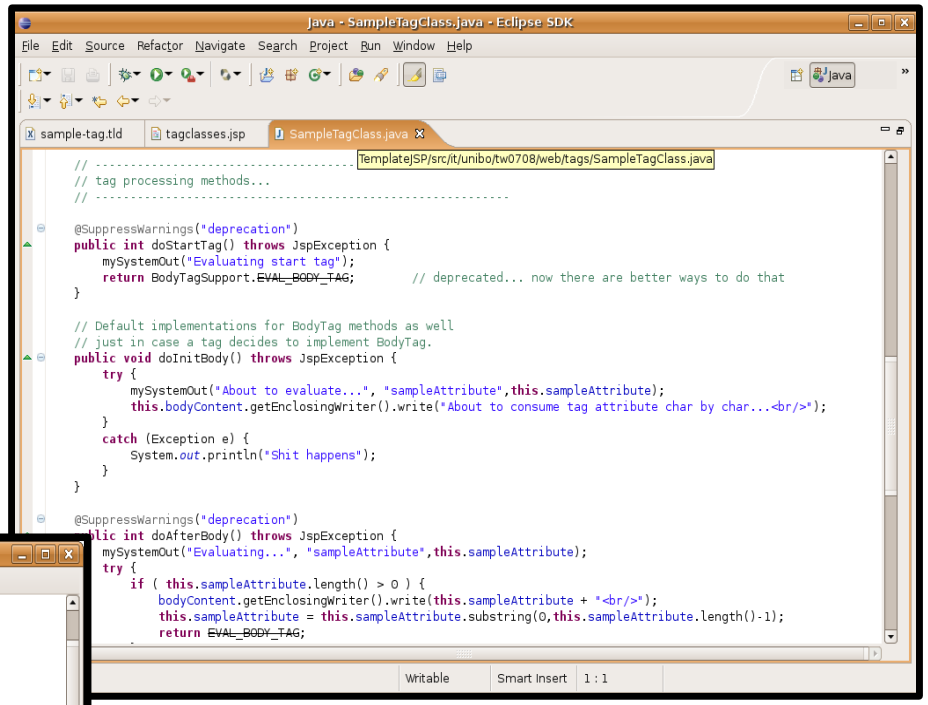
tagclasses.jsp

> *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



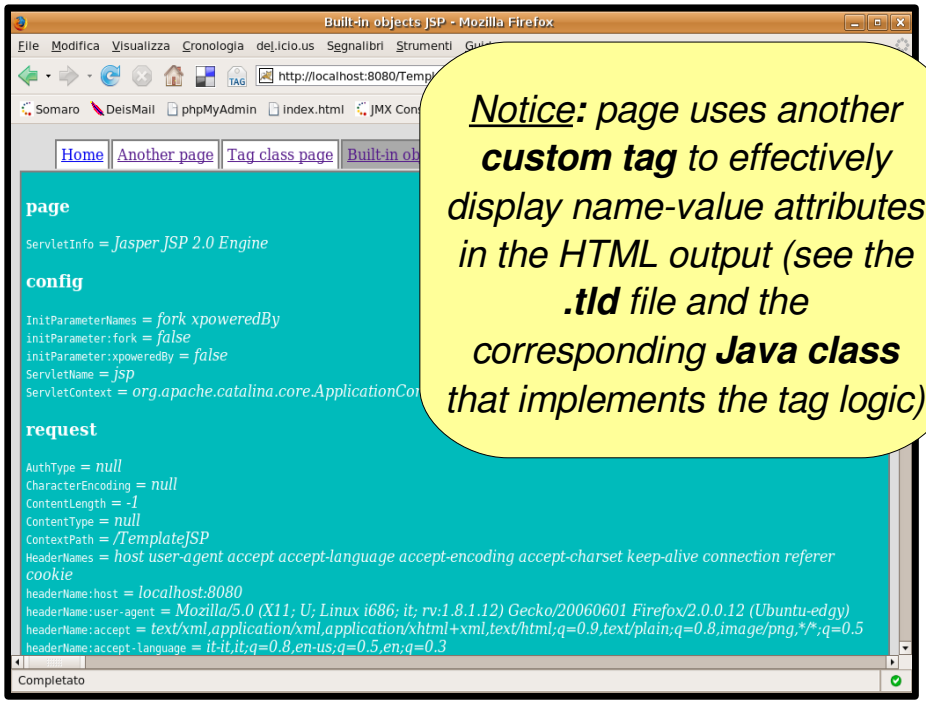




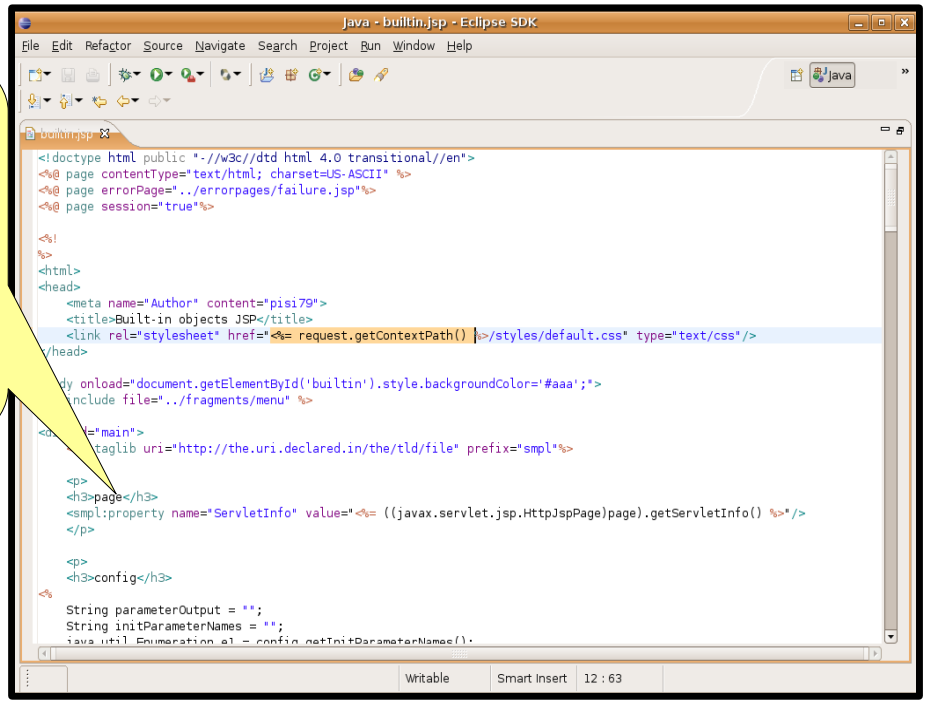
builtin.jsp (1/2)

> 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



Notice: *page uses another* ***custom tag*** *to effectively display name-value attributes in the HTML output (see the* ***.tld*** *file and the corresponding* ***Java class*** *that implements the tag logic)*



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()
        );
%>
```

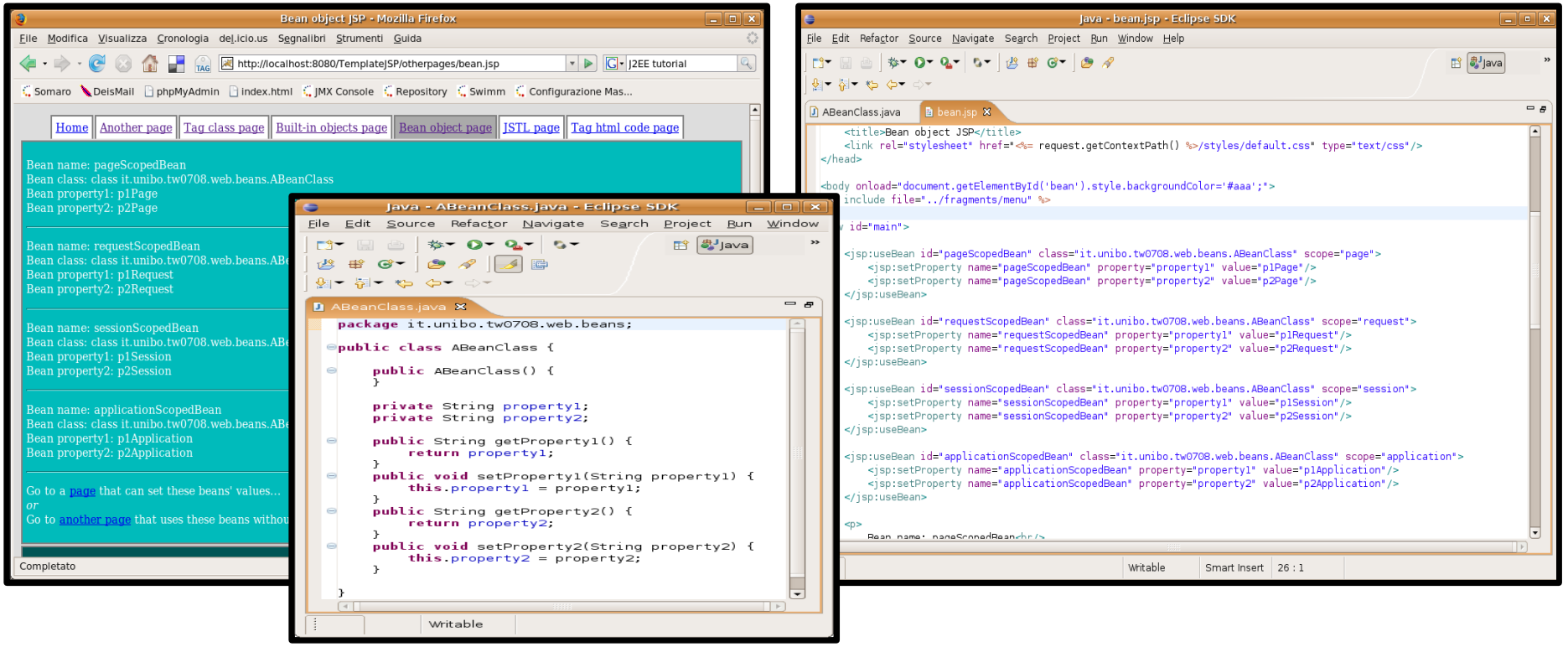
*Does it seem that running **undeploy** / **deploy** targets is not making any difference?*

*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



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

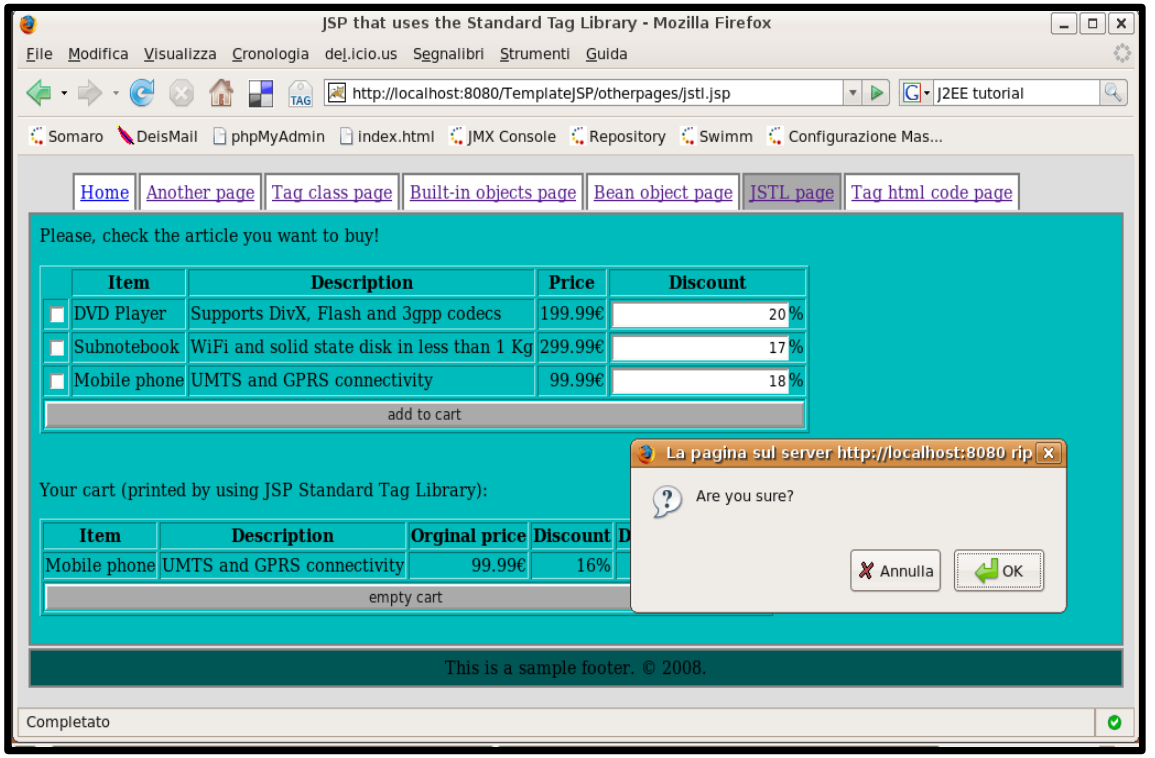
>...few other things we have already seen...
  - a session-scoped 'cart' bean, of class *it.unibo.tw0708.web.beans.Cart* that holds items selected by the user
  - request's context path being used to 'absolutely' reference styles
  - JSP error pages, inclusion of external HTML fragments

> ...and lot of **HTML-embedded styling** → see how pages can easily become **messy**!!!

jstl.jsp (2/2)

> Page structure:

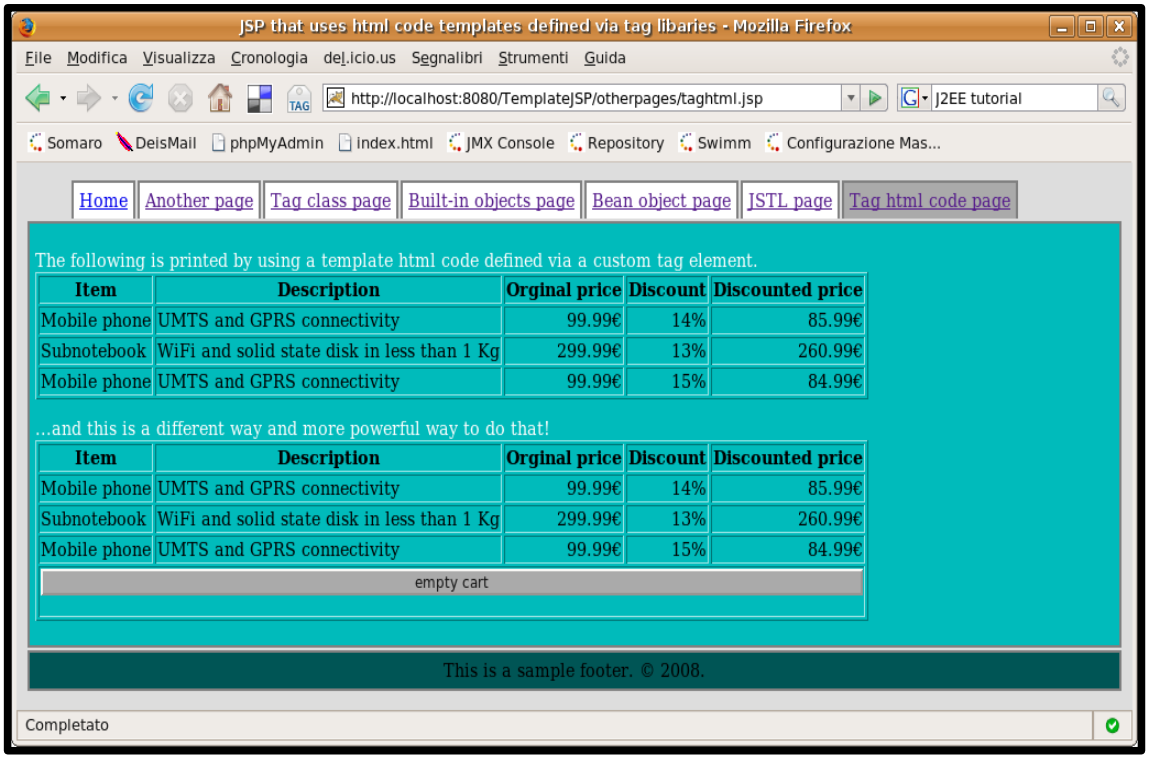
- A table holds form input fields
- Each discount value is calculated by invoking a custom function
- Form sends selected items along with a new action request to the current page
- Request parameters are read to add items in the session-scoped cart
- The 'empty' parameter, if present, is handled by 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



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**...

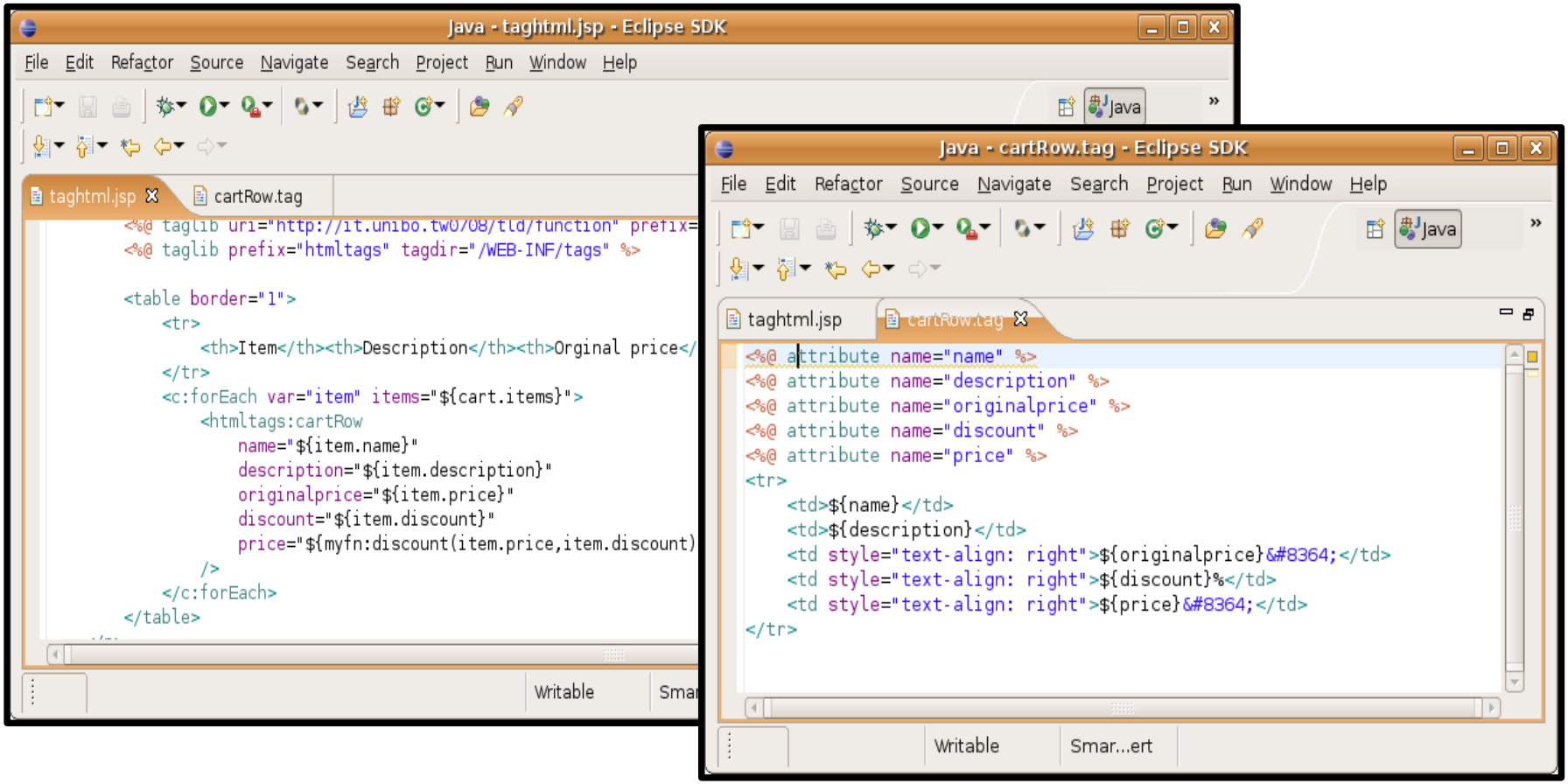


- ...whose **definitions** (**.tag* files) **can be found at the location** (*WEB-INF/tags/*) **specified** by the JSP page itself
- ...and whose **tag names correspond to the *.tag file names** at that location

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



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!