

Anno Accademico 2007-2008

Laboratorio di Tecnologie Web

Sviluppo di applicazioni web

Servlet

Before we go on...

> From the course web site you can download an Eclipse project ([TW0708.zip](#)) resembling the course web site structure:

- Download it and import it as usual + set up *environment.properties*
- You can sniff scripts, styles and pages behaviour...
- ...or try do it better!

The image is a collage of three screenshots illustrating the development and deployment environment for the course website.

- Top Left:** A screenshot of the Eclipse IDE. The Package Explorer on the left shows a project named 'TW0708' with a 'war' folder containing 'classes', 'home.html', 'laboratorio.html', and 'ValentineServlet'. The main editor shows a JavaScript file named 'hidshow.js' with the following code:

```
// <!-- Hide and shows div childs of a given element
function hideChilds(p) {
    var e = document.getElementById(p);
    var childs = e.getElementsByTagName("div");
    for ( var name in childs ) {
        try {
            if ( childs[name] instanceof HTMLDivElement )
                childs[name].style.visibility = "hidden";
        } catch ( err ) {
            // IE does not define HTMLDivElement
        }
    }
    //alert(name + ' = ' + childs[name]);
    if ( childs[name] != null && childs[name] != null )
        try {
            childs[name].style.visibility = "hidden";
            childs[name].style.display = "none";
        } catch ( err ) { }
    }
}
/*
Note: for each is since Javascript 1.6 and only FF and IE currently support
for each ( var child in childs ) {
    if ( child instanceof HTMLDivElement ) {
        child.style.visibility = "hidden";
        child.style.display = "none";
    }
}
*/
```
- Top Right:** A screenshot of a Mozilla Firefox browser window displaying the course website. The address bar shows 'http://localhost:8080/TecnologieWeb0708/'. The page content includes:
 - University name: [Università degli Studi di Bologna - Facoltà di Ingegneria - Deis - Lia](#)
 - Course: [Corso di Tecnologie WEB - A.A. 2007-2008](#)
 - Prof. Giuliano Franceschi: gfranceschi@deis.unibo.it
 - Ing. Samuele Pasini: spasini@deis.unibo.it
 - Course Objectives: [Obiettivi del corso](#)
 - Course Schedule: [Orario Lezioni](#)
 - Didactic Material: [Materiale didattico](#)
 - Laboratory: [Laboratorio](#)
 - Exams: [Esami](#)
 - Contacts: [Contatti](#)
- Bottom Right:** A close-up photograph of a person's hands typing on a laptop keyboard.

Inside the course web site replica project: security

> The *web.xml* descriptor shows how to secure part of the site

- either via explicit login/error pages or popup forms (commented)
- by leveraging Tomcat users/roles, for the sake of simplicity (by now)

The image displays two overlapping windows. The left window is the Eclipse IDE, showing the `web.xml` file with the following XML configuration:

```
<!-- Security ----- -->
<security-constraint>
  <display-name>Example Security Constraint</display-name>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <!-- Define the context-relative URL(s) to be protected -->
    <url-pattern>/laboratorio.html</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <!-- Anyone with one of the listed roles may access this area -->
    <role-name>manager</role-name>
  </auth-constraint>
</security-constraint>

<!-- Default login configuration uses form-based authentication -->
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/loginpages/login.html</form-login-page>
    <form-error-page>/loginpages/forbidden.html</form-error-page>
  </form-login-config>
</login-config>

<!-- Security roles referenced by this web application -->
<security-role>
  <role-name>manager</role-name>
</security-role>

</web-app>
```

The right window is a Mozilla Firefox browser showing an authentication page titled "Richiesta autenticazione - Mozilla Firefox". The page content includes:

Università degli Studi di Bologna - Facoltà di Ingegneria - Deis - Lia

Corso di Tecnologie WEB - A.A. 2007-2008

Prof. Giuliano Franceschi: gfranceschi@deis.unibo.it
Ing. Samuele Pasini: spasini@deis.unibo.it

Home

Indietro

Accesso non consentito

Devi autenticarti come utente accreditato per accedere alla risorsa richiesta.

Username:

Password:

Invia richiesta

Inside the course web site replica project: ANT & tunnel

> There's a particular ANT build file (*build-tunnel.xml*) that runs a configurable tcp tunnel in a GUI (= monitor) mode.

- You can listen on port, say, 9999 and forward requests to 8080
- Run the ANT target, browse to <http://localhost:9999> instead of <http://localhost:8080> and see what happens

The screenshot displays the Eclipse IDE interface. The main editor shows the `build-tunnel.xml` file with the following content:

```
<?xml version="1.0" encoding="UTF-8" ?>
<project name="run.tcptunnel" >
  <input
    message="Please enter source port (default = 8880):"
    addproperty="source.port"
    defaultvalue="8880"
  />
  <input
    message="Please enter destination host (default = localhost)"
    addproperty="destination.host"
    defaultvalue="localhost"
  />
  <input
    message="Please enter destination port (default = 8080):"
    addproperty="destination.port"
    defaultvalue="8080"
  />
  <java
    classname="org.apache.soap.util.net.TcpTunnelGui"
    fork="true"
  >
    <classpath>
      <!-- main class -->
      <pathelement location="ant/lib/soap.jar"/>
    </classpath>
    <arg value="\${source.port}"/>
    <arg value="\${destination.host}"/>
    <arg value="\${destination.port}"/>
  </java>
</project>
```

The right-hand side of the image shows the TCP Tunnel/Monitor window, which is tunneling localhost:9999 to localhost:8080. The window displays the following log output:

```
From localhost:9999
GET / HTTP/1.1
Host: 127.0.0.1:9999
User-Agent: Mozilla/5.0 (X11; U; Linux i686; it; rv:1.8.1.12) Gecko/20060601 Firefox/3.0
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive

GET /tomcat.gif HTTP/1.1
Host: 127.0.0.1:9999
User-Agent: Mozilla/5.0 (X11; U; Linux i686; it; rv:1.8.1.12) Gecko/20060601 Firefox/3.0
Accept: image/png,*/*;q=0.5
Accept-Language: it-it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://127.0.0.1:9999/

GET /jaf-logo-wide.gif HTTP/1.1
Host: 127.0.0.1:9999
User-Agent: Mozilla/5.0 (X11; U; Linux i686; it; rv:1.8.1.12) Gecko/20060601 Firefox/3.0
Accept: image/png,*/*;q=0.5
Accept-Language: it-it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://127.0.0.1:9999/

GET /tomcat-power.gif HTTP/1.1
Host: 127.0.0.1:9999
User-Agent: Mozilla/5.0 (X11; U; Linux i686; it; rv:1.8.1.12) Gecko/20060601 Firefox/3.0
Accept: image/png,*/*;q=0.5
Accept-Language: it-it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://127.0.0.1:9999/

Listening for connections on port 9999 ...
```

The right-hand side of the window shows the response from localhost:8080:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=ISO-8859-1
Transfer-Encoding: chunked
Date: Mon, 18 Feb 2008 10:33:23 GMT

2000
<!--
Licensed to the Apache Software Foundation (ASF) under one or more
contributor license agreements. See the NOTICE file distributed with
this work for additional information regarding copyright ownership.
The ASF licenses this file to You under the Apache License, Version 2.0
(the "License"); you may not use this file except in compliance with
the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
-->
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Apache Tomcat/5.5.25</title>
    <style type="text/css">
      /*  */
      body {
        color: #000000;
        background-color: #FFFFFF;
        font-family: Arial, "Times New Roman", Times, serif;
        margin: 10px 0px;
      }
    &lt;/style&gt;
  &lt;/head&gt;
  &lt;body&gt;
    &lt;div style="text-align: center;"&gt;
      &lt;img alt="Apache Tomcat logo" data-bbox="400 400 600 600"/>
    &lt;/div&gt;
  &lt;/body&gt;
&lt;/html&gt;</pre></div><div data-bbox="71 915 191 937" data-label="Page-Footer"><p>|Tecnologie Web L-A</p></div><div data-bbox="684 899 743 978" data-label="Page-Footer"><img alt="Logo of Alma Mater Studiorum University of Bologna"/></div><div data-bbox="743 919 923 954" data-label="Page-Footer"><p>ALMA MATER STUDIORUM<br/>UNIVERSITÀ DI BOLOGNA</p></div>
```

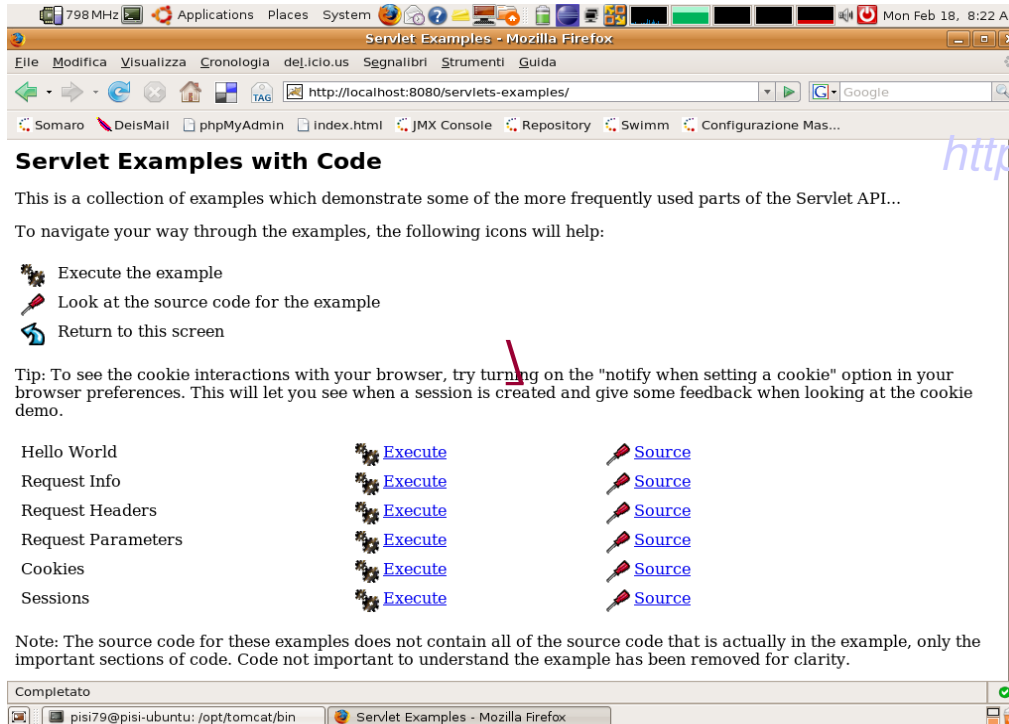
Server-side code: servlets

- > Servlets are J2EE modular components that run on the server-side and process requests to programmatically generate dynamic responses
- > How to create a servlet
 - Write code and descriptors (for instance within an Eclipse project)
 - Add static resources (HTML pages, styles) and client-side scripts if needed
 - Package all the stuff into a .war file
 - Deploy it to the web server

You can use the *TemplateWebapp* project as a starting point
or the *TemplateServlet* one (with just some sample code inside, in addition,
just to get you started immediately)

Servlet examples

> Tomcat provides, out-of-the-box, some servlet (and JSP) examples that can be useful to learn how to deal with Servlet's API and facilities



- You can invoke examples on

<http://localhost:8080/servlets-examples/>

or by following links on your Tomcat local home page

- Source code for the examples is partially accessible via web (through the example pages themselves) and completely available on you file system on

[\\$TOMCAT_HOME/webapps/servlet-examples](#)

For instance: request info (1/2)

- > The 'Request Info' servlet shows information about the request being handled
- > You can check request arguments and headers against the tunnel GUI

The image shows two overlapping windows. The left window is a Mozilla Firefox browser displaying the 'Request Information Example' page. The page content is as follows:

```
Request Information Example

Method:      GET
Request URI: /servlets-examples/servlet/RequestInfoExample
Protocol:    HTTP/1.1
Path Info:   null
Remote Address: 127.0.0.1
```

The right window is a 'TCP Tunnel/Monitor: tunneling localhost:9999 to localhost:8080' application. It shows the raw HTTP request and response. The request from localhost:9999 is:

```
GET /servlets-examples/servlet/RequestInfoExample HTTP/1.1
Host: localhost:9999
User-Agent: Mozilla/5.0 (X11; U; Linux i686; it; rv:1.8.1.12) Gecko/20060601 Fire
Accept: text/xml,application/xml,application/xhtml+xml;text/html;q=0.9;text/plain
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://localhost:9999/servlets-examples/
```

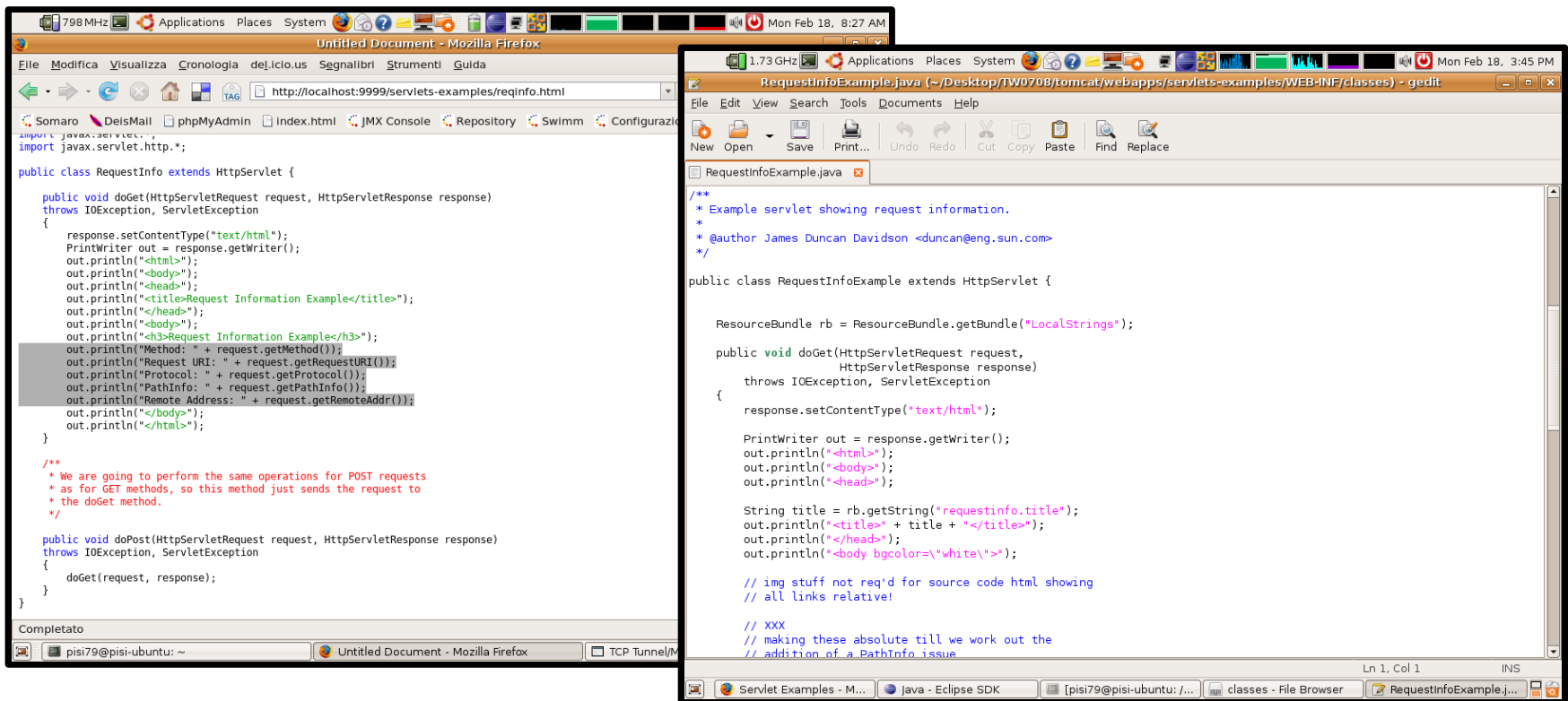
The response from localhost:8080 is:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 689
Date: Mon, 18 Feb 2008 07:25:59 GMT

<html>
<body>
<head>
<title>Request Information Example</title>
</head>
<body bgcolor="white">
<a href="." />Request Info Example</a>
Request Info Example</img>
<a href="." />Request Info Example</a>
Request Info Example</img>
<h3>Request Information Example</h3>
<table border="0"> <tr> <td>
Method:
</td> <td>
GET
</td> </tr> <tr> <td>
Request URI:
</td> <td>
/servlets-examples/servlet/RequestInfoExample
</td> </tr> <tr> <td>
Protocol:
</td> <td>
HTTP/1.1
</td> </tr> <tr> <td>
Path Info:
</td> <td>
null
</td> </tr> <tr> <td>
Remote Address:
</td> <td>
127.0.0.1
</td> </tr> </table>
```


For instance: request info (2/2)

- > The 'screwdriver' link brings you to code's relevant excerpts
- > Alternatively, you can open the *RequestInfoExample.java* source file on your file system (path: $\$TOMCAT_HOME/webapps/servlets-examples/WEB-INF/classes$)



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

public class RequestInfo extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<head>");
        out.println("<title>Request Information Example</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h3>Request Information Example</h3>");
        out.println("Method: " + request.getMethod());
        out.println("Request URI: " + request.getRequestURI());
        out.println("Protocol: " + request.getProtocol());
        out.println("PathInfo: " + request.getPathInfo());
        out.println("Remote Address: " + request.getRemoteAddr());
        out.println("</body>");
        out.println("</html>");
    }

    /**
     * We are going to perform the same operations for POST requests
     * as for GET methods, so this method just sends the request to
     * the doGet method.
     */

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
    {
        doGet(request, response);
    }
}
```

```
/**
 * Example servlet showing request information.
 * @author James Duncan Davidson <duncan@eng.sun.com>
 */

public class RequestInfoExample extends HttpServlet {

    ResourceBundle rb = ResourceBundle.getBundle("LocalStrings");

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<head>");

        String title = rb.getString("requestinfo.title");
        out.println("<title>" + title + "</title>");
        out.println("</head>");
        out.println("<body bgcolor=\<white>");

        // img stuff not req'd for source code html showing
        // all links relative!

        // XXX
        // making these absolute till we work out the
        // addition of a PathInfo issue
    }
}
```


For instance: cookies

- > Cookies are pieces of information, kept client-side
- > Each cookie is associated to the domain that has saved it
- > The '**Cookie Example**' servlet shows how to handle cookies
- > The browser settings can show you the cookies in your browser

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

public class CookieExample extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // print out cookies

        Cookie[] cookies = request.getCookies();
        for (int i = 0; i < cookies.length; i++) {
            Cookie c = cookies[i];
            String name = c.getName();
            String value = c.getValue();
            out.println(name + " = " + value);
        }

        // set a cookie

        String name = request.getParameter("cookieName");
        if (name != null && name.length() > 0) {
            String value = request.getParameter("cookieValue");
            Cookie c = new Cookie(name, value);
            response.addCookie(c);
        }
    }
}
```

Cookie Name: JSESSIONID
Cookie Value: 84AF934B03B3B160F1D80007BD4799D5

You just sent the following cookie to your browser:
Name: nome
Value: valore

Create a cookie to send to your browser

Name:
Value:

I seguenti cookie sono memorizzati sul computer:

Sito	Nome cookie
localhost	nome
localhost	JSESSIONID

Nome: nome
Contenuto: valore
Server: localhost
Percorso: /servlets-examples/servlet/
Invia per: Qualunque tipo di connessione
Scadenza: a fine sessione



For instance: session

- > Session holds pieces of information, kept server-side
- > Clients can retrieve their sessions by means of session identifiers (kept in cookies or made available by rewriting URLs)

The image shows a screenshot of a web browser displaying a Java Servlet session example. The browser window is titled "Sessions Example - Mozilla Firefox" and shows the URL `http://localhost:8080/servlets-examples/servlet/SessionExample`. The page content displays session information:

```
Session ID: 84AF934B03B3B160F1D80007BD4799D5
Created: Mon Feb 18 10:01:51 CET 2008
Last Accessed: Mon Feb 18 10:01:51 CET 2008
```

The page also shows a form for setting session attributes:

The following data is in your session:

Name of Session Attribute:
Value of Session Attribute:

GET based form:

Name of Session Attribute:
Value of Session Attribute:

[URL encoded](#)

A "Cookie" dialog box is open, showing the following cookie details:

Sito	Nome cookie
localhost	JSESSIONID

Nome: JSESSIONID
Contenuto: 84AF934B03B3B160F1D80007BD4799D5
Server: localhost
Percorso: /servlets-examples
Invia per: Qualunque tipo di connessione
Scadenza: a fine sessione

Buttons: Rimuovi cookie, Rimuovi tutti i cookie, Chiudi

The background shows a code editor with the following Java code:

```
public class SessionExample extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        HttpSession session = request.getSession(true);

        // print session info

        Date created = new Date(session.getCreationTime());
        Date accessed = new Date(session.getLastAccessedTime());
        out.println("ID " + session.getId());
        out.println("Created: " + created);
        out.println("Last Accessed: " + accessed);

        // set session info if needed

        String dataName = request.getParameter("dataName");
        if (dataName != null && dataName.length() > 0) {
            String dataValue = request.getParameter("dataValue");
            session.setAttribute(dataName, dataValue);
        }

        // print session contents

        Enumeration e = session.getAttributeNames();
        while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            String value = session.getAttribute(name).toString();
            out.println(name + " = " + value);
        }
    }
}
```

For instance: searching for documentation (1/2)

> Do you remember what URL encoding provides for?

- The code shows it is generated by invoking `response.encodeURL()`
- Where to find more information? → ***Browse the API documentation!!***

The image shows a composite screenshot illustrating a search for documentation. On the left, a Mozilla Firefox browser window displays Google search results for 'HttpServletResponse encodeURL'. The top result is 'Interface HttpServletResponse' from the Java EE 5 SDK, which is highlighted with a blue circle. Below it, a link to 'Difference between HttpServletResponse.encodeURL' is also circled. A blue speech bubble with the text 'Try googling on method name!' is overlaid on the search results. On the right, an IDE window shows the 'SessionExample.java' file. The code includes a call to `response.encodeURL("SessionExample?dataname=foo&datavalue=bar")`, which is highlighted in pink. The IDE also shows a file browser on the left with a tree view of the project structure.

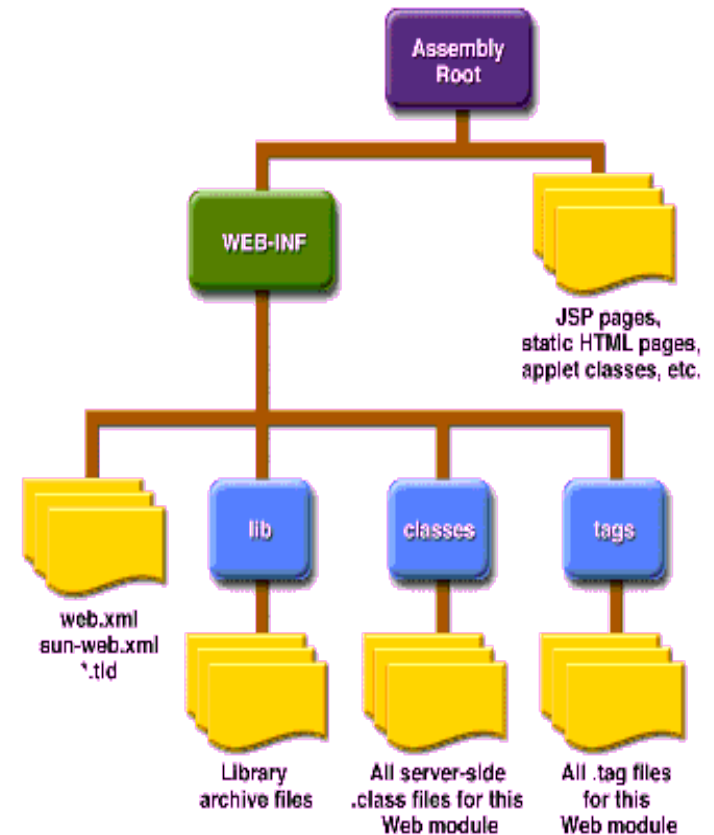
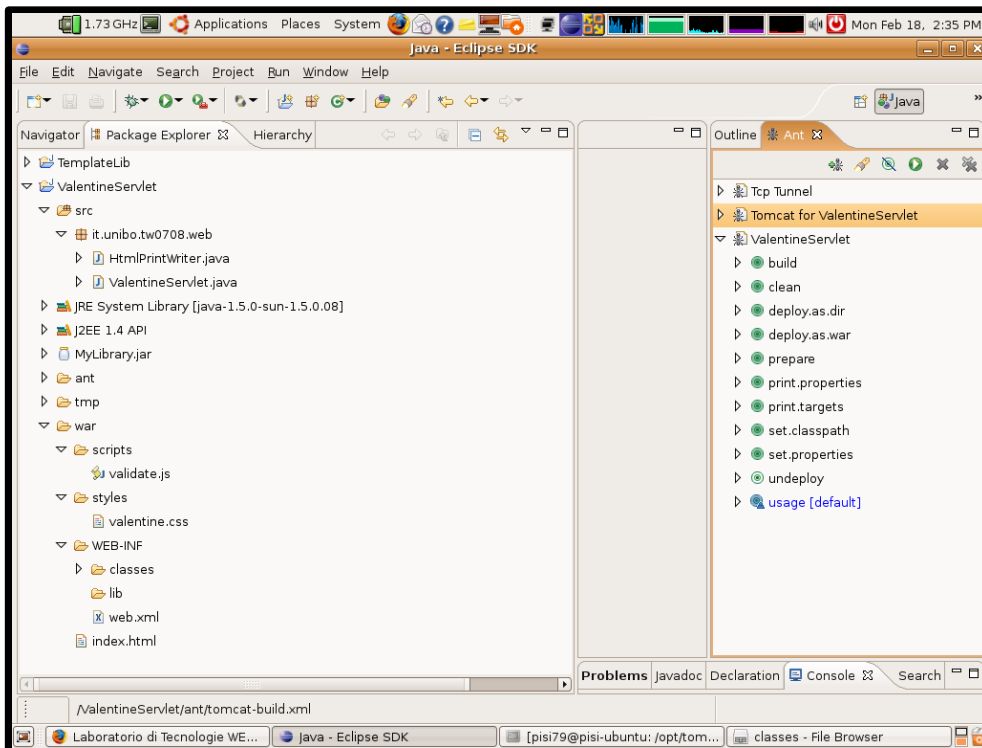
For instance: searching for documentation (2/2)

> Besides, you can find a local copy of JEE API documentation on the course site

The image displays two screenshots of a Mozilla Firefox browser window. The left screenshot shows a search for 'HttpServletRequest' on the website 'Laboratorio di Tecnologie WEB - A.A. 2007-2008'. The search results are displayed in a sidebar, listing links for 'Tomcat', 'Hypersonic', 'Documentazione', and 'Varie'. The right screenshot shows the 'Overview (Java EE 5) - Mozilla Firefox' page for the 'Java Platform Enterprise Edition, v 5.0 API Specifications'. The page features a navigation menu with 'Overview', 'Package', 'Class', 'Tree', 'Deprecated', 'Index', and 'Help'. The main content area is titled 'Java EE 5 Platform Packages' and lists several packages with brief descriptions: 'javax.activation', 'javax.annotation', 'javax.annotation.security', 'javax.ejb', 'javax.ejb.spi', and 'javax.el'. The 'javax.el' package is highlighted, indicating it is the current selection.

Valentine's servlet: project structure

- > Download [ValentineServlet.zip](#) project from the course site and import it in Eclipse
- > This silly servlet computes the chances of... love affairs (*..wrote it on Feb 14th ...*)

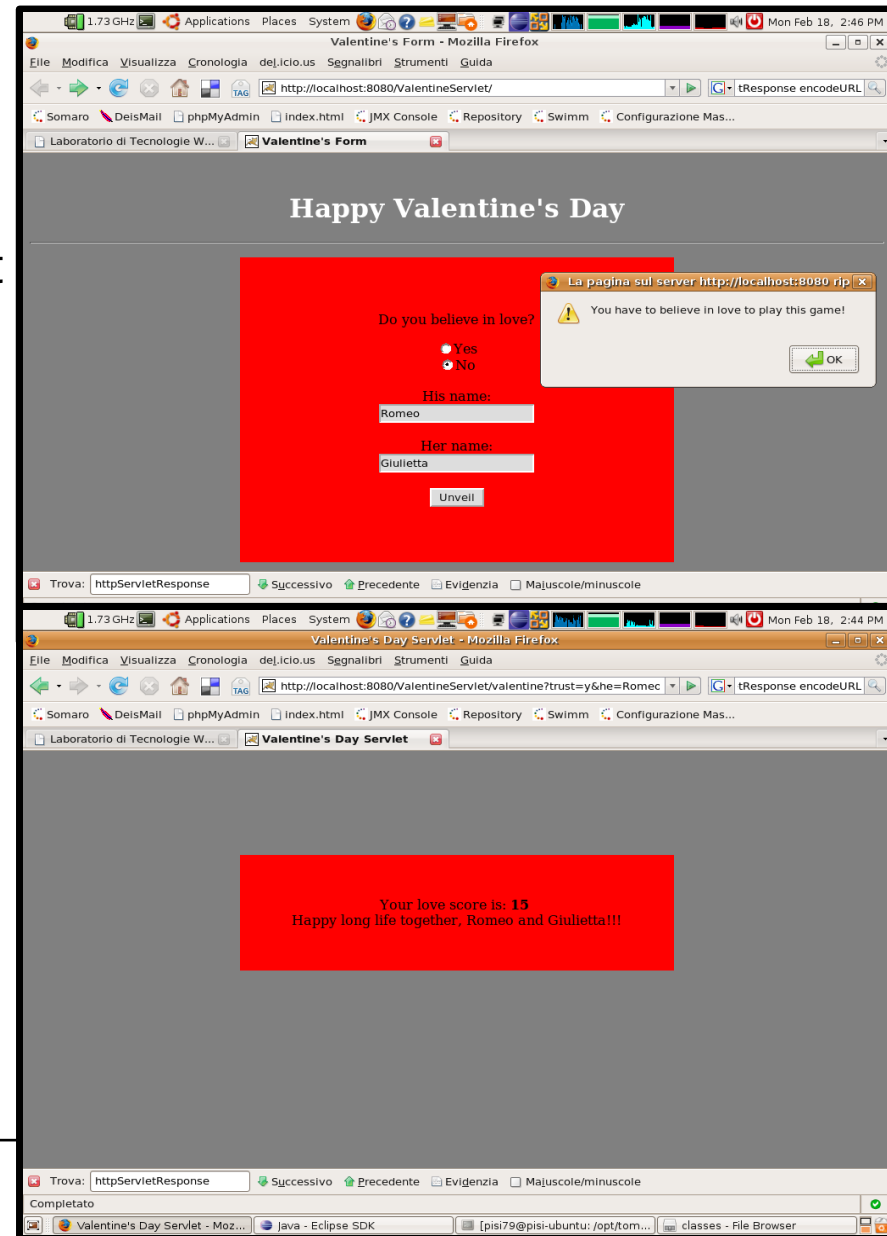


Valentine's servlet: elements

> Just a sample servlet project:

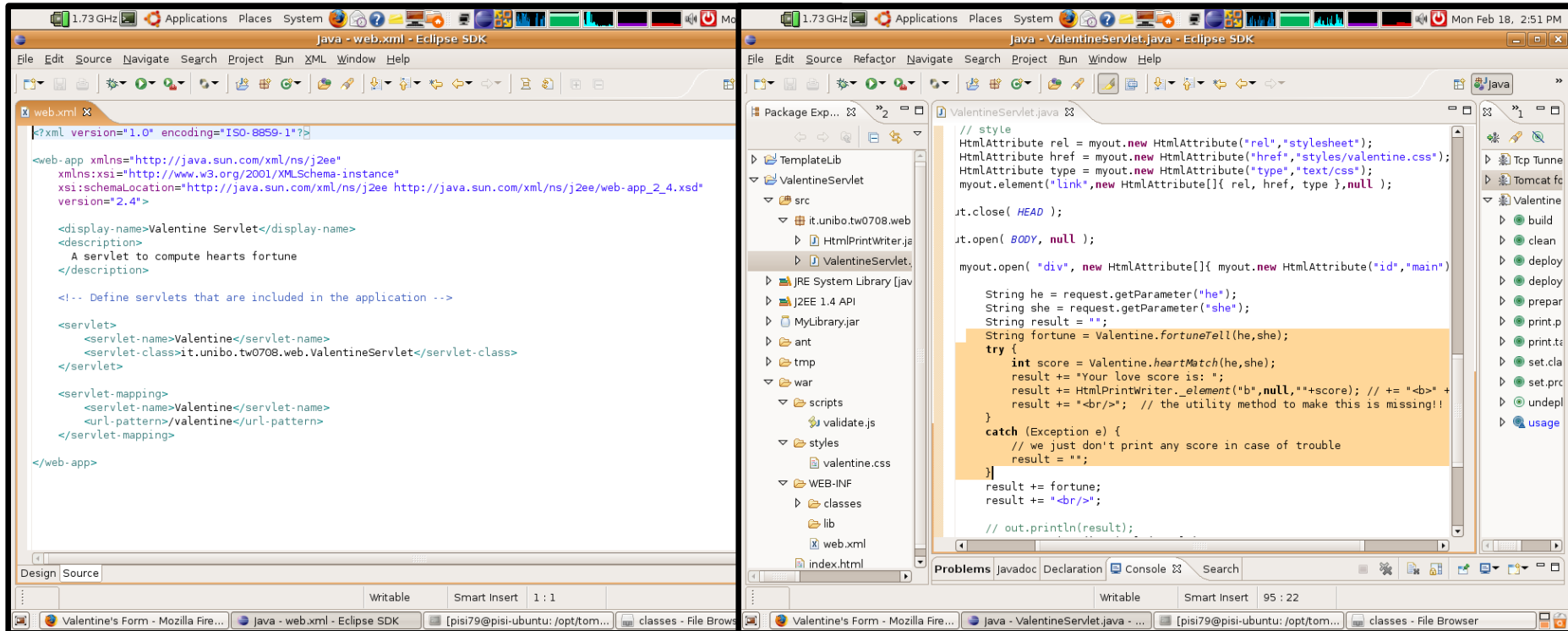
- **main page** displays a form
- **stylesheet** provides for a 'lovely' layout
- **form** action invokes a servlet, by passing input field values as parameters
- **javascript** validates form on the client-side before performing the request
- **servlet** invokes a **library** function to compute results

> Look at the project structure and find out elements!



Valentine's servlet: servlet mapping

- > *web.xml* descriptor maps *ValentineServlet* class to */valentine* path by means of the servlet's *Valentine* name
- > servlet retrieves request parameters and invokes library functions to produce result



- > library is stored in the *WEB-INF/lib/* path of the *.war* archive (note: Eclipse is not going to show it there, in the '*Package Explorer*' view, because it's part of the build-path!!!)

Valentine's servlet: not to mess up with output

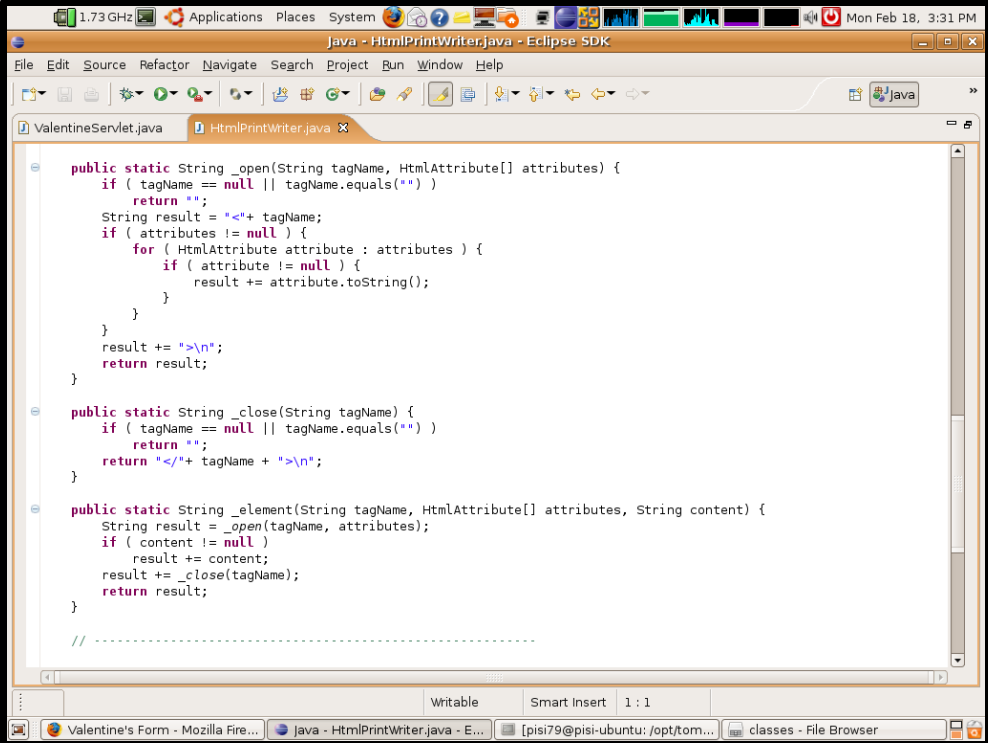
> I just can't stand typing all those "<", ">", "/" and "\" in the code

- so I wrote my own utility class (*HTMLPrintWriter.java*) that arranges tag strings on my behalf

- but you can just write *out.println("<div id=\"anId\">Something</div>");*

if it makes you fill more

confortable



```
public static String _open(String tagName, HtmlAttribute[] attributes) {
    if ( tagName == null || tagName.equals("") )
        return "";
    String result = "<"+ tagName;
    if ( attributes != null ) {
        for ( HtmlAttribute attribute : attributes ) {
            if ( attribute != null ) {
                result += attribute.toString();
            }
        }
    }
    result += ">\n";
    return result;
}

public static String _close(String tagName) {
    if ( tagName == null || tagName.equals("") )
        return "";
    return "</"+ tagName + ">\n";
}

public static String _element(String tagName, HtmlAttribute[] attributes, String content) {
    String result = _open(tagName, attributes);
    if ( content != null )
        result += content;
    result += _close(tagName);
    return result;
}

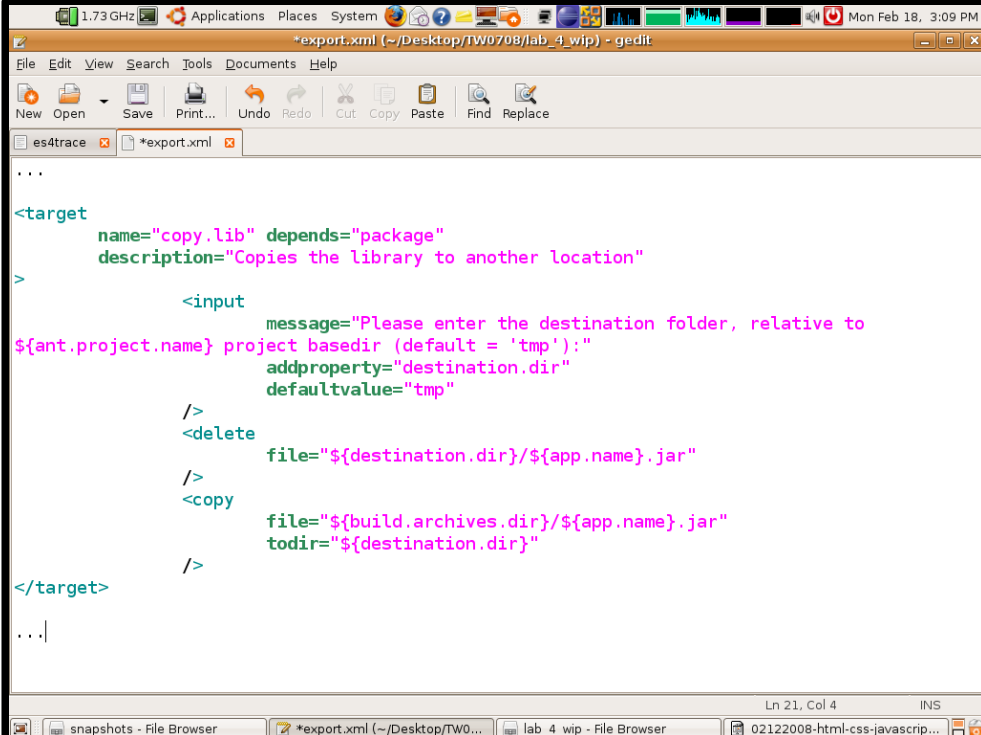
// -----
```

Valentine's servlet: reusing libraries across projects

- > *MyLibrary* is the result of the *'package'* target in the ANT build file of project *TemplateLib* (that you can download from the course web site and import in Eclipse)
- > It's just a simple (not webap) project like the *TemplateApp* one of class #1
- > But... do I have to copy and paste manually that jar library in projects where I need it?

Try to add a new ANT target do to that!

- you can copy **delete**, and **copy** tasks from the *deploy*-related targets of other projects
- you can add a **popup message** asking for the destination by copying from the target that runs the *TcpTunnel*
- and that's all

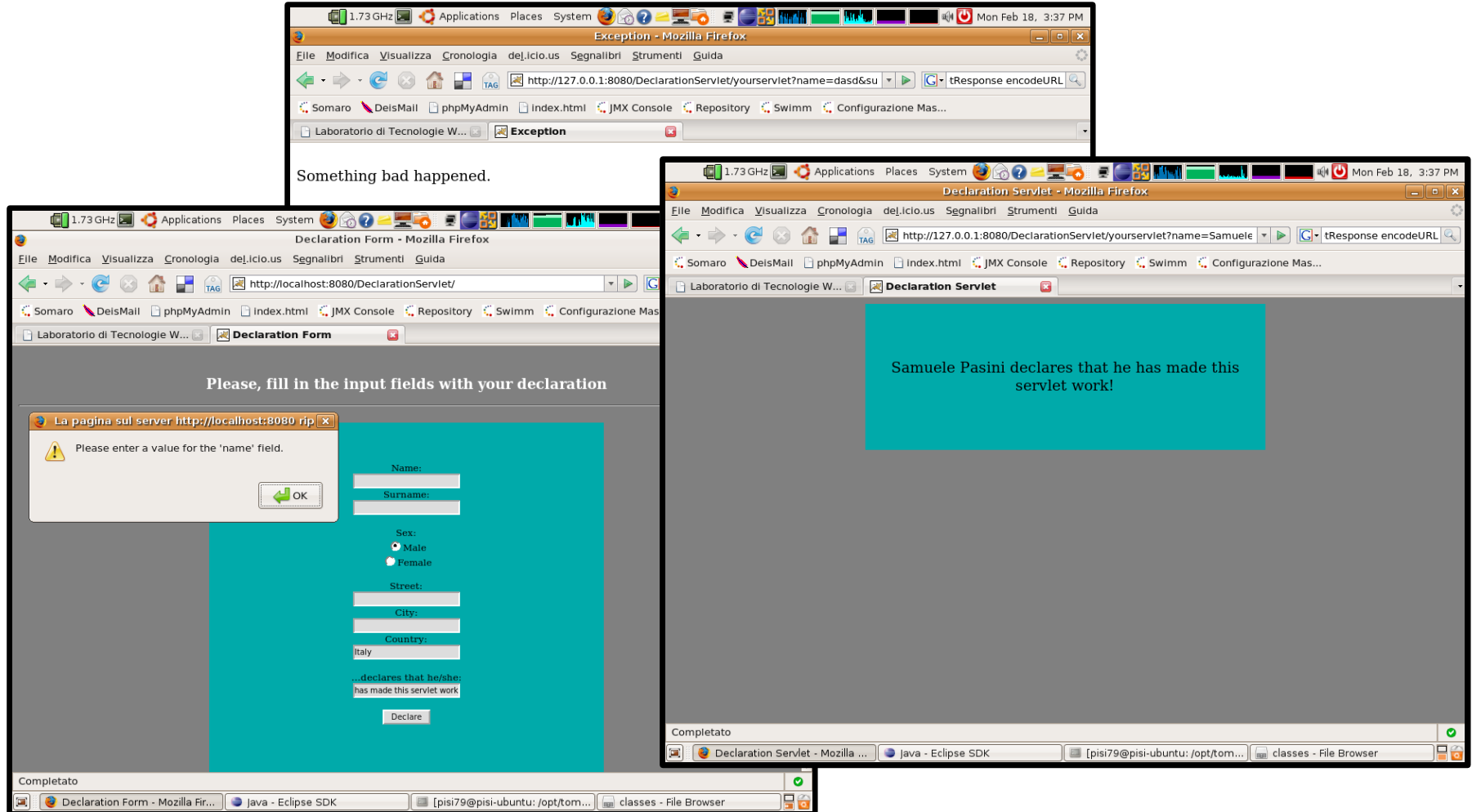


```
...  
<target  
  name="copy.lib" depends="package"  
  description="Copies the library to another location"  
>  
  <input  
    message="Please enter the destination folder, relative to  
    ${ant.project.name} project basedir (default = 'tmp');"  
    addproperty="destination.dir"  
    defaultValue="tmp"  
  />  
  <delete  
    file="${destination.dir}/${app.name}.jar"  
  />  
  <copy  
    file="${build.archives.dir}/${app.name}.jar"  
    todir="${destination.dir}"  
  />  
</target>  
...
```

Declaration servlet: it behaves the same

- > *MyLibrary.jar* also contains a class to assemble declaration statements
 - want to know how it works without looking at the code? try exporting the Javadoc to the */doc* path in the *TemplateLib* project!
- > Try making your own servlet:
 - use the *TemplateServlet* project as a structure skeleton for your project
 - download the zip and import it in Eclipse
 - change project name, project.properties (and build files' project name too, if you want to) and set up your environment.properties file
 - a form that asks for name, surname, address, sex and a declaration
 - javascript code that ensures name and surname fields are non-blank ones
 - you can copy that from the sample pages in *CSSandJavascript.zip*
 - a servlet that processes the request by invoking library methods & returns results
 - some styles to make things look pretty-good
 - perhaps you can handle exceptions via *web.xml* mapping to error pages
(see what happens if people from hell perform declarations...)

Declaration servlet: for instance...



> *DeclarationServlet* sample project will be online after the lab class...