

JSP .....	2
Required setup .....	2
Download and install Apache Tomcat.....	2
Download and install 'Sysdeo Eclipse Tomcat Launcher plugin' .....	3
Download and install 'Solar Eclipse' .....	4
A Simple JSP example : WhatTimeIsIt.....	5
Building and debugging a lot-of-tiers J2EE web application .....	9
View-Controller (the model will come later) .....	9
A first view .....	9
Some More Views (we would not need a controller otherwise) .....	10
Then comes the controller .....	12
Add the Model .....	18
Store and get preferences and settings in XML files .....	18
What's next .....	18

## JSP

Lets move from the server to the client now. Eclipse, together with some freely available plug-ins, allows us to write, debug and deploy web sites using J2EE technologies (servlets and Jsp, it is). To deploy J2EE web applications, we need a web server that is able to read, unzip, compile and execute servlets. To run jsp pages, we need a web server that can transform jsp pages into servlets, compile and execute them.

Although JBoss comes with Jetty installed (Jetty is such a web server), we will use a standalone Tomcat installation for our webserver. The reason for this is that I do not know of any Eclipse plug-in that lets you debug jsp pages when running Jetty within Jboss. I do know about a plug-in that lets you debug jsp pages when running with a standalone Tomcat. So that is what we will to.

Later, when we know how to make and debug simple JSP pages, we will go back to Jboss and make our JSP client communicate with our sessionbean.

Here is an outline of what we will be doing next:

- Downloads and installs:
  - Apache Tomcat
  - Sysdeo Tomcat plugin
  - Solar Eclipse
- Make and deploy a simple JSP page.
- Make and deploy a 2 page JSP application with a little controller servlet.
- Share data between these 2 pages using java beans

### Required setup

#### Download and install Apache Tomcat

<http://jakarta.apache.org/>

I downloaded the file tomcat-4.1.18.zip and unzipped it into my Applications folder. And that is basically all there is to do. Be sure that the environment variable 'JAVA\_HOME' is declared. I added the declaration of JAVA\_HOME to my .tcshrc file :

```
setenv JAVA_HOME /Library/Java/Home
```

The file named '.tcshrc' is in your home folder (if it is not, you must create it). You'll need an editor that can read hidden files (like Metrowerks Codewarrior or BBedit or plain old VI - from the command line) in order to do so. If you use another shell, you probably know enough of Unix to know what file you must modify to create environment variables.

To start Tomcat, open a terminal, navigate to applications/jakarta-tomcat-4.1.18/bin, and enter the following:

```
Prompt> cd applications/jakarta-tomcat-4.1.18/bin
Prompt> ./startup.sh
Using CATALINA_BASE:   /applications/jakarta-tomcat-4.1.18
Using CATALINA_HOME:  /applications/jakarta-tomcat-4.1.18
Using CATALINA_TMPDIR: /applications/jakarta-tomcat-4.1.18/temp
Using JAVA_HOME:      /Library/Java/Home
```

Next, you can test your installation by opening a web browser and navigate to <http://localhost:8080/>. (or <http://127.0.0.1:8080/> if your browser says that dns failed to locate localhost, as my chimera did). If everything went well, you should see this:

If Tomcat fails to start, complaining that port 8080 is already occupied, you probably have Jboss still running. Jboss comes with Jetty, which is another Web server that is configured to use port 8080. So be sure that you shut down Jboss before starting Tomcat.

Once you have Tomcat working, shut it down, As we will be control it from within JBoss.

```
Prompt> cd applications/jakarta-tomcat-4.1.18/bin
Prompt> ./shutdown.sh
Using CATALINA_BASE:   /applications/jakarta-tomcat-4.1.18
Using CATALINA_HOME:   /applications/jakarta-tomcat-4.1.18
Using CATALINA_TMPDIR: /applications/jakarta-tomcat-4.1.18/temp
Using JAVA_HOME:       /Library/Java/Home
Prompt>
```

## Download and install 'Sysdeo Eclipse Tomcat Launcher plugin'

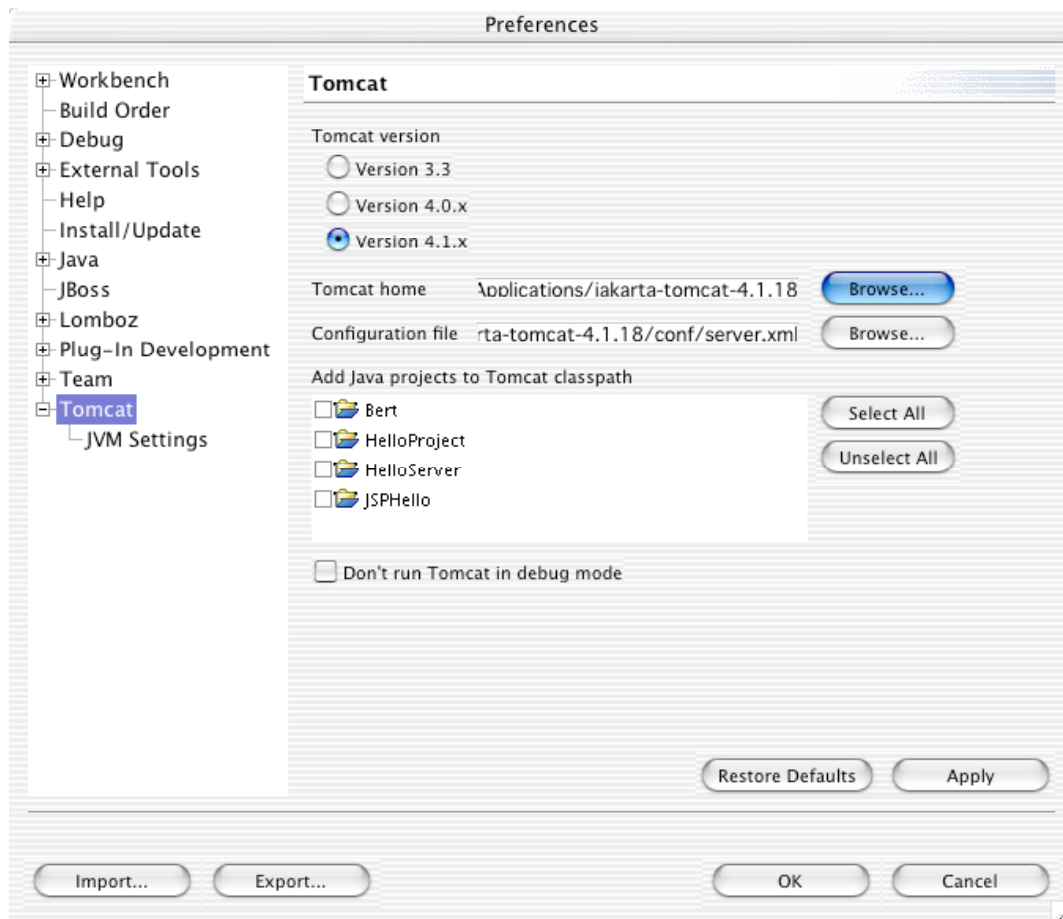
The 'Sysdeo Eclipse Tomcat Launcher plugin' (we'll call it Sysdeo for the rest of this article) does to Tomcat what the EASY JBoss Plugin does to JBoss. It allows you to start and stop Tomcat from within JBoss, and to debug Servlets and Jsp pages. It also lends a hand when setting up a web application project in Eclipse. J2EE web applications do have a predefined structure. Sysdeo can set up this structure for you.

## Download and install

You can download Sysdeo from <http://www.sysdeo.com/eclipse/tomcatPlugin.html>. I downloaded the file named tomcatPluginV201.zip. and unzipped it into the Eclipse plugin folder (Eclipse/Contents/Resources/Java/plugins). You open the Eclipse/Contents folder by <ctrl><Click>ing the Eclipse executable.

## Setup

- Select (when being in resource perspective) 'Window->Customize Perspective' from the menu.
- Check 'Tomcat' in the category 'Others'.
- Repeat these steps for all the perspectives from where you want to start or stop Tomcat.
- Select the menu 'Window->Preferences' and tell Sysdeo where it can find Tomcat.



You should now have a new menu called 'Tomcat' from where you can start and stop Tomcat. (yes, you can try it now).



## Download and install 'Solar Eclipse'

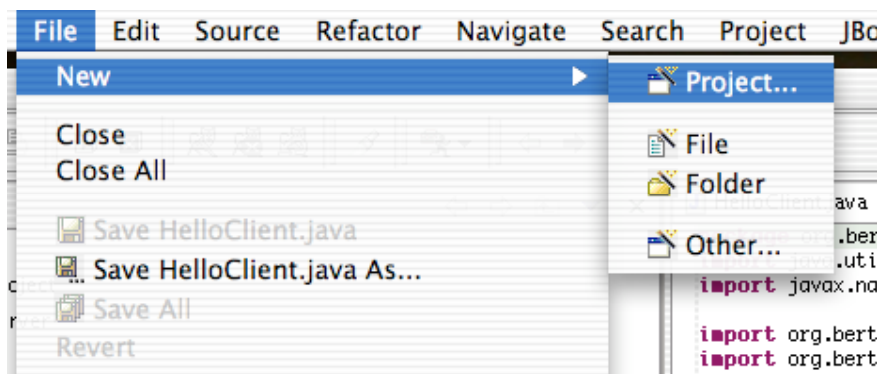
Solar eclipse is an Eclipse plug-in that does syntax highlighting for jsp and for XML files. You can download it from <http://sourceforge.net/projects/solareclipse/>. When using IE 5.2, the name of the download file gets truncated. (IE5.2 actually launches GraphicConvertor to handle the download on my Mac), so you must manually restore its full name (net.sf.solareclipse\_0.3.1.bin.dist.zip) if you want to unzip it via a double-click.

To install Solar Eclipse, just unzip it and move the four plugin folders into the plugin folder of the Eclipse package. (<Ctrl-Click> on the eclipse icon and choose 'Show package contents' from the contextual menu to get into the Eclipse package). Alternatively you could download the Lombok plugin. Jsp syntax colouring is just a very small part of its feature set.

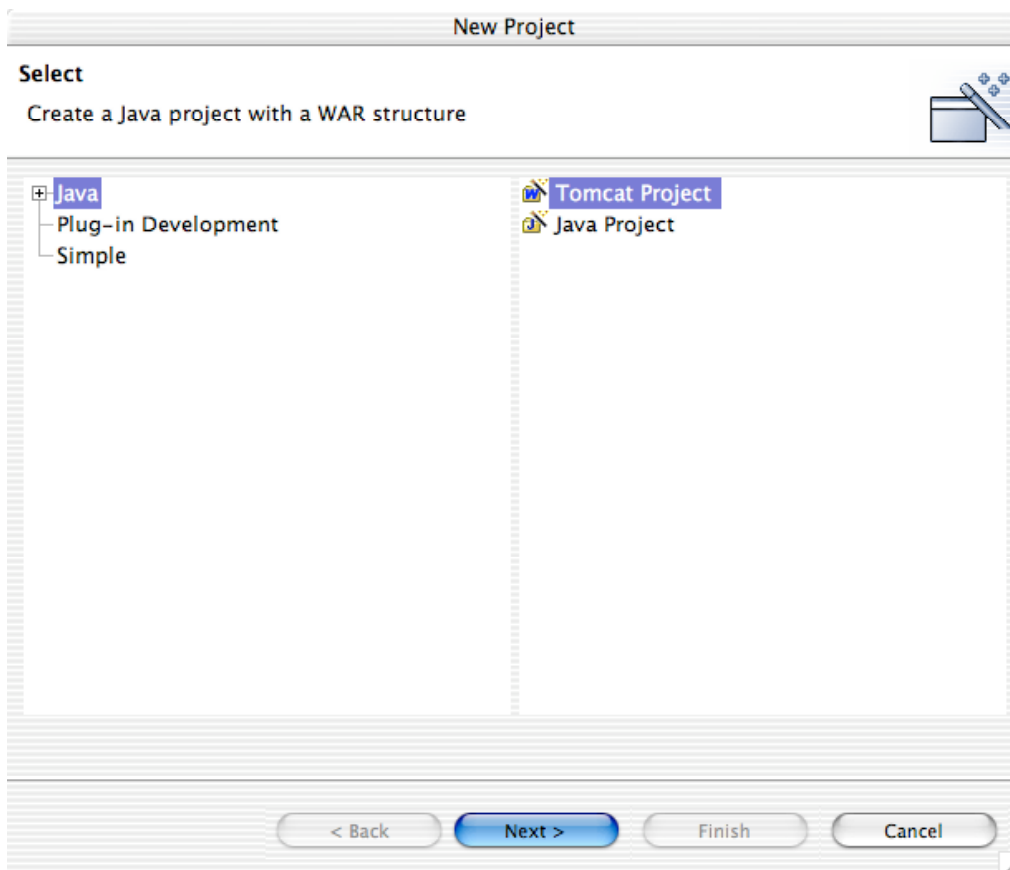
## A Simple JSP example : WhatTimeIsIt.

The first JSP example is mainly copied from the Sysdeo documentation. You can read the original – in French – at the following address : <http://www.eclipse totale.com/articles/tomcat/tomcatPluginDocFR.html>. Here we go :

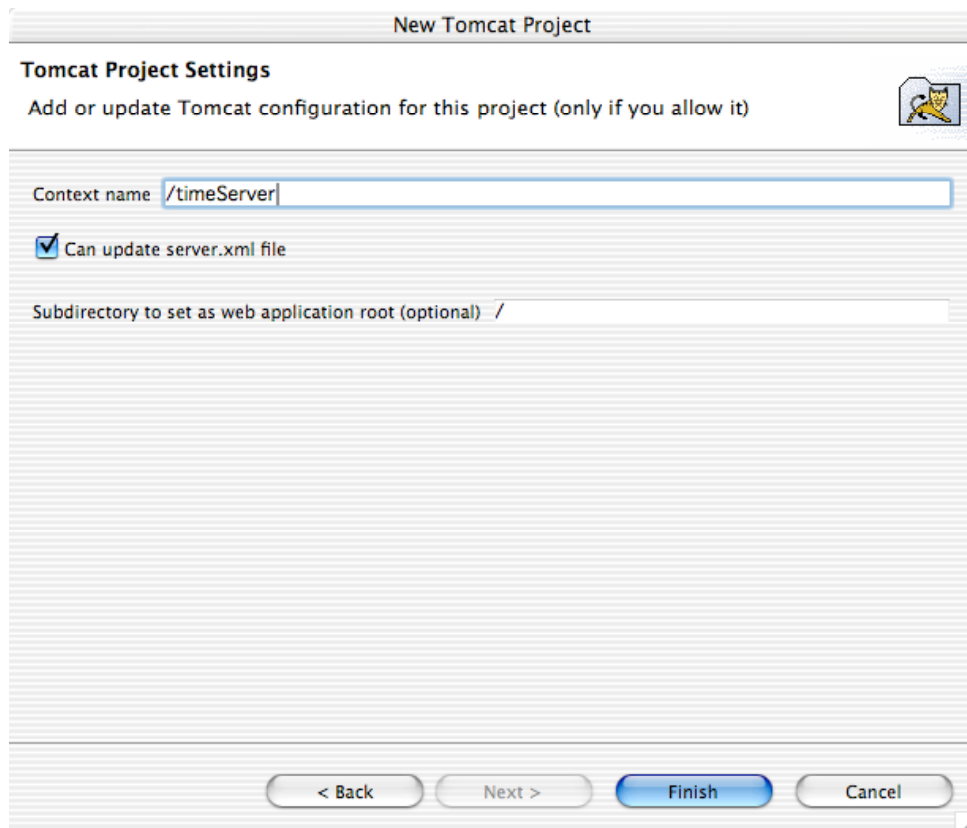
- Choose File->New Project



- Select 'Tomcat Project' from the dialog that opens next.



- Enter a name for your project (I called it WhatTimeIsIt) and press 'Next>'.
- The dialog that opens next opens only when you choose 'Tomcat Project' from the 'New' dialog. It is part of the Sysdeo plug-in. As the context name, I entered 'timeServer'. We will have to use that context name later to connect to the web application we are building. I also checked the 'Can update server.xml' checkbox. If you check this, the Sysdeo plug-in will make all the changes necessarily to deploy our project in Tomcat for us. If you are an experienced Tomcat administrator, you can leave this checkbox unchecked.

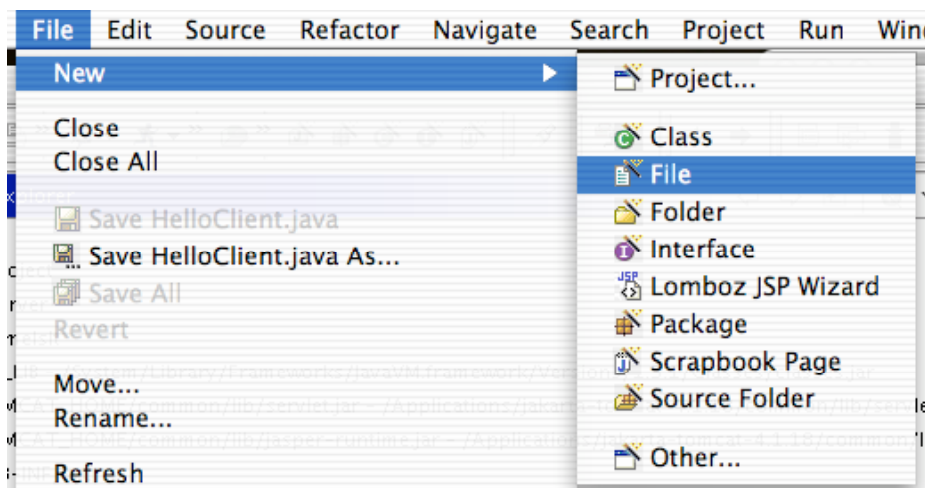


- Press 'Finish'.
- If you now look at the Server.xml configuration file of Tomcat, you will see that the line

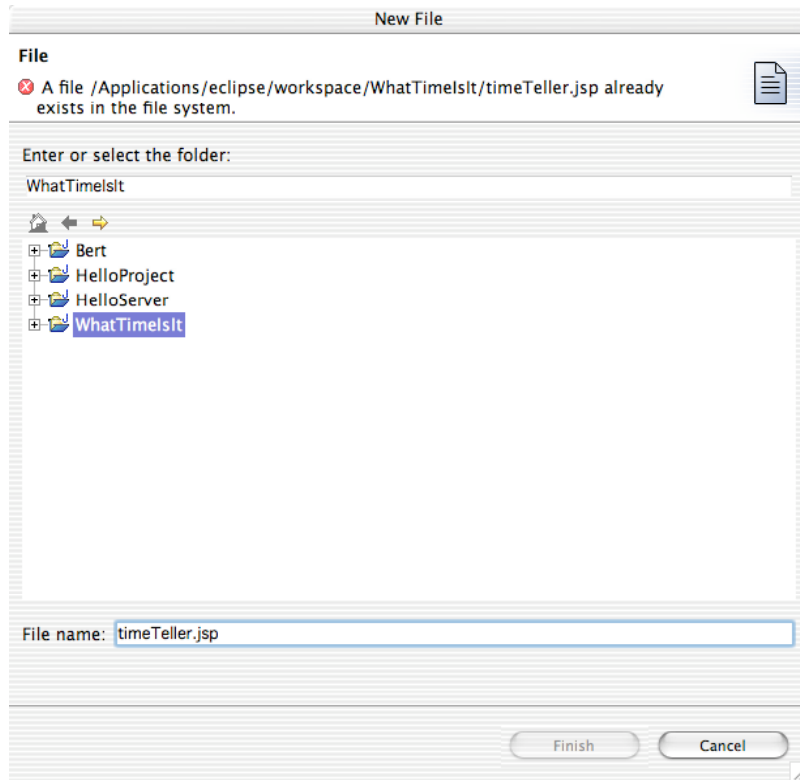
```
<Context path="/timeServer" docBase="/Applications/eclipse/workspace/WhatTimeIsIt"
workDir="/Applications/eclipse/workspace/WhatTimeIsIt/work/org/apache/jsp" />
```

was added just before the </Host></Engine></Service> closing tags. Sysdeo did this because you checked the 'Can update server.xml' checkbox.

- Next, we create a simple JSP page. Choose <file->new->file> from the menu.



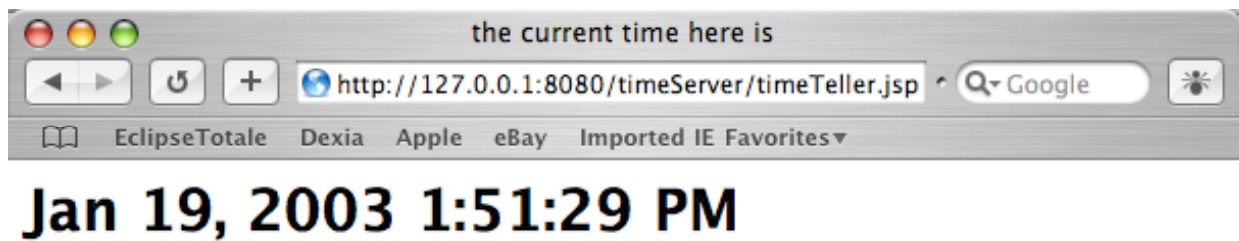
- Create the new file at the root of our new project. I named the new file 'timeTeller.jsp'.



- Press 'Finish'. You will now see the contents (empty) of timeTeller.jsp in the edit pane of Eclipse. Enter the following and save the file :

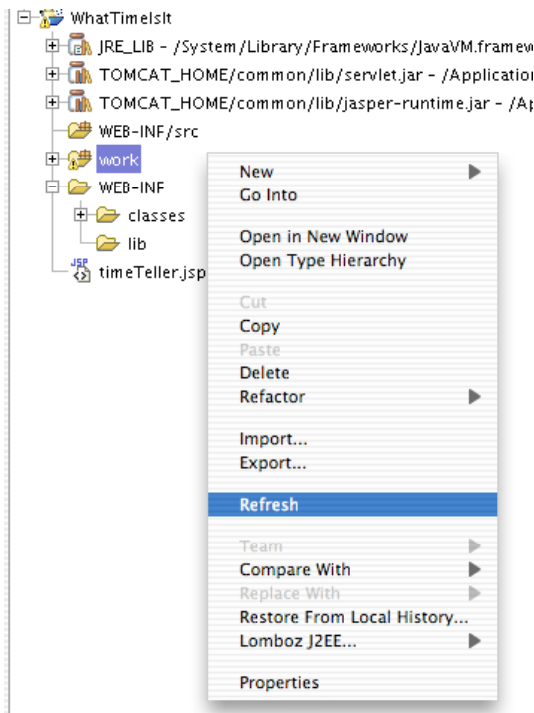
```
<%@ page info="Tells what time it is at the servers location" %>
<html>
<head><title>the current time here is</title></head>
<H1>
<%= (new java.util.Date()).toLocaleString() %>
</H1>
</html>
```

- Now start Tomcat
- And open the page <http://127.0.0.1:8080/timeServer/timeTeller.jsp> in your favorite browser. You should see the following:



JSP pages are transformed into servlets before they are compiled and executed. So debugging JSP at the source level is not possible. If you use the Sysdeo plugin, you can debug the servlet code that was generated from the JSP page. To do so, you must follow the next steps :

- Select the work-directory of your project and select 'refresh' from the contextual menu



- Now navigate into the org.apache.jsp tree until you arrive at the file name 'timeTeller\_jsp.java'. That file contains the servlet code that was generated from your jsp. You can put breakpoints in that file as you can do in any other java file. Just place a breakpoint before the line that says `out.print( (new java.util.Date()).toLocaleString() );`



- Refresh the page in your browser...
- ... and eclipse will switch to debug perspective and open the servlet code where you placed the breakpoint. Be sure to browse around a bit in the variables pane : it gives a fairly good idea on the internal structure and data of servlets.

## Building and debugging a lot-of-tiers J2EE web application

So we pretty much covered up the basic tools and techniques. Lets move on now and glue everything together.

Our first application will have a front end written in HTML/Jsp and use a controller servlet to navigate between the different pages. Later, we will add a model component that will implement some business intelligence.

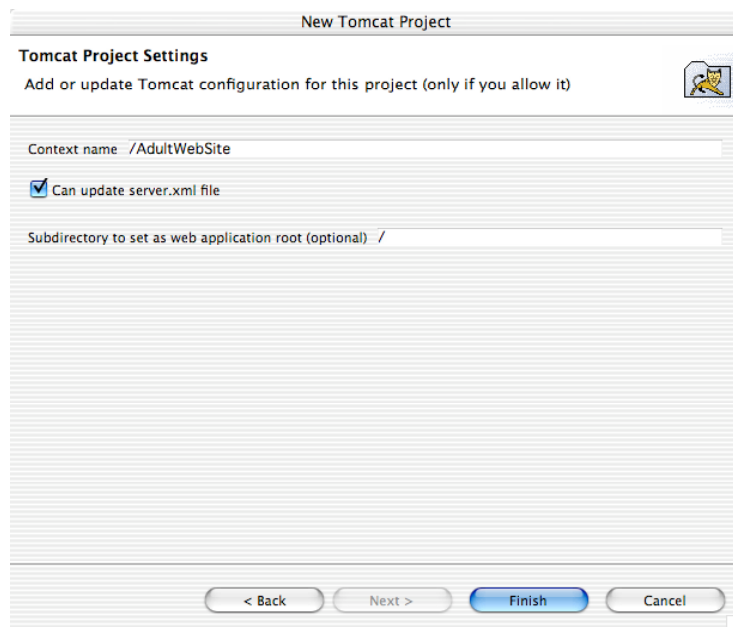
Our example application is a simple one. A first page will ask for some personal information. Based on what we enter, you'll get a second page for adults, or one for children.

### ***View-Controller (the model will come later)***

In a first iteration of this example, we will create 4 objects and modify some XML.

- A file 'GetInfo.jsp' will ask some very personal details.
- A servlet 'AdultWebSiteController' will take a look at these details and open, after some investigation, a second page
- The page 'AdultPage.jsp' shows adult contents.
- The page 'ChildrenPage.jsp' is intended for younger public.
- The Javabeen 'PersonalInfo.java' will hold whatever the user entered.

So lets start by creating a new project. Create a new Tomcat project and name it AdultWebSite. To make our life easy, we let Eclipse update our Server.xml file




### **A first view**

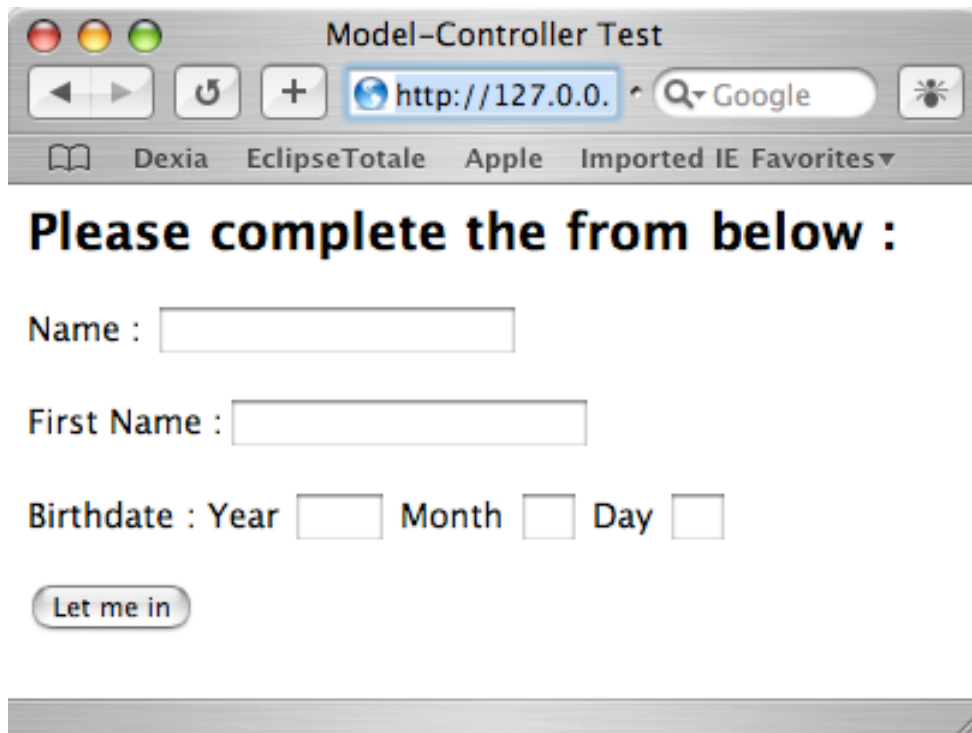
Once our project is created, we add the page 'GetInfo.jsp' at the root of our project.

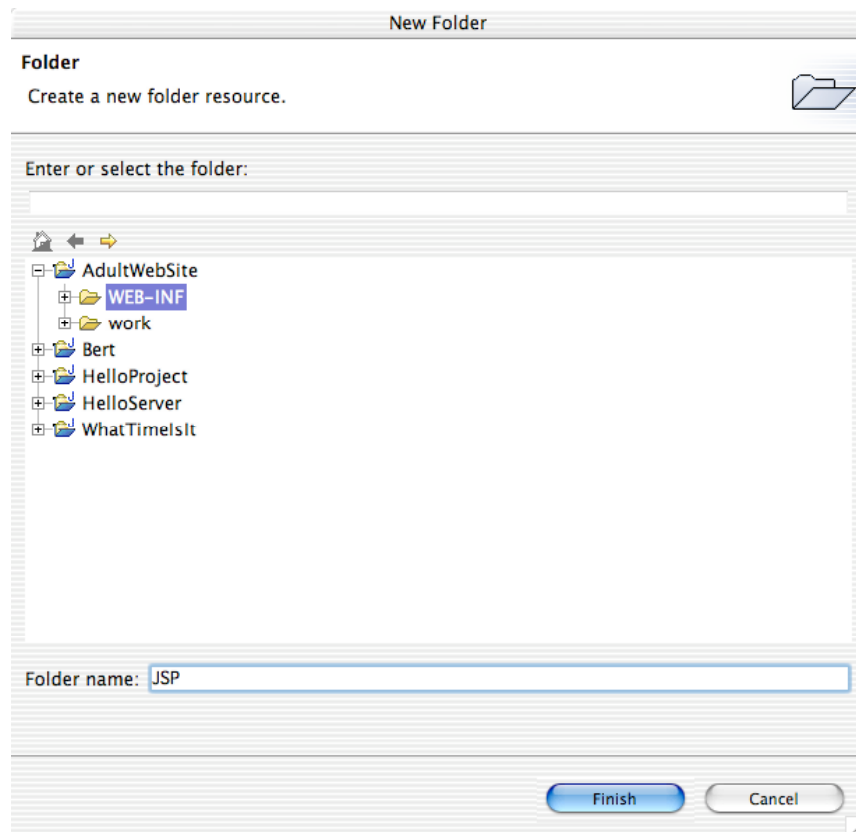
- <Control-Click> the AdultWebSite project and choose 'New file' from the contextual menu.
- Choose the project root as the place to store the file, and enter 'GetInfo.jsp' as the name of the new file.
- The new file will be opened in the editor pane. Enter the following code :

```
<%@ page contentType="text/html; charset=windows-1252"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1252">
<title>Model-Controller Test
</title>
</head>
<body>
<h2>
Please complete the form below :
</h2>
<form action="/AdultWebSite/SiteController/WhichPage" name="sender">
```

```
method="post">
<p>
Name : <input type="text" name="Name" maxlength="50">
</p><p>
First Name :<input type="text" name="FirstName" maxlength="50">
</p><p>
Birthdate : Year <input type="text" name="year" size="4" maxlength="4">
Month <input type="text" name="month" size="2" maxlength="2">
Day <input type="text" name="day" size="2" maxlength="2">
</p>
<input type="submit" name="submit" value="Let me in">
</body>
</html>
```

- To test our page, we start Tomcat (click the  button).
- And browse with our favorite browser to <http://127.0.0.1:8080/AdultWebSite/GetInfo.jsp>.
- If everything went well, you'll see something like this (pretty ugly he):





- Then <Ctrl-Click> the newly created JSP folder. Select <New->File> from the contextual menu and name the file Adults.jsp. Then put something like this in that file :

```
<%@ page contentType="text/html; charset=windows-1252"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Model-Controller Test
</title>
</head>
<body>
<h2>
This page is only to be viewed by mentally healthy people over 18.
</h2>
<p>
</p>
</body>
</html>
```

- Save the file
- Create another file 'Children.jsp with the following contents

```
<%@ page contentType="text/html; charset=windows-1252"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Model-Controller Test
</title>
</head>
<body>
<h2>
This is a page suited for children
</h2>
<p>
</p>
</body>
</html>
```

- And one last page named 'InvalidInput.jsp'.

```
<%@ page contentType="text/html; charset=windows-1252"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Model-Controller Test
</title>
</head>
<body>
<h2>
Whatever you entered was invalid or incomplete. I could not make out where to go.
</h2>
<p>
</p>
</body>
</html>
```

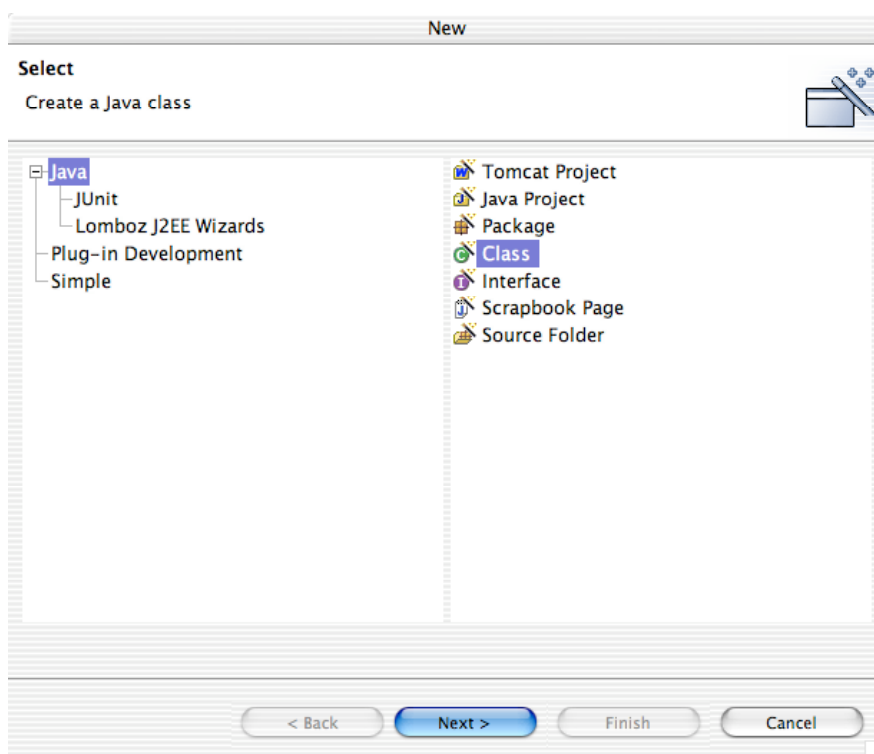
## Then comes the controller

### Make a first version of the controller servlet

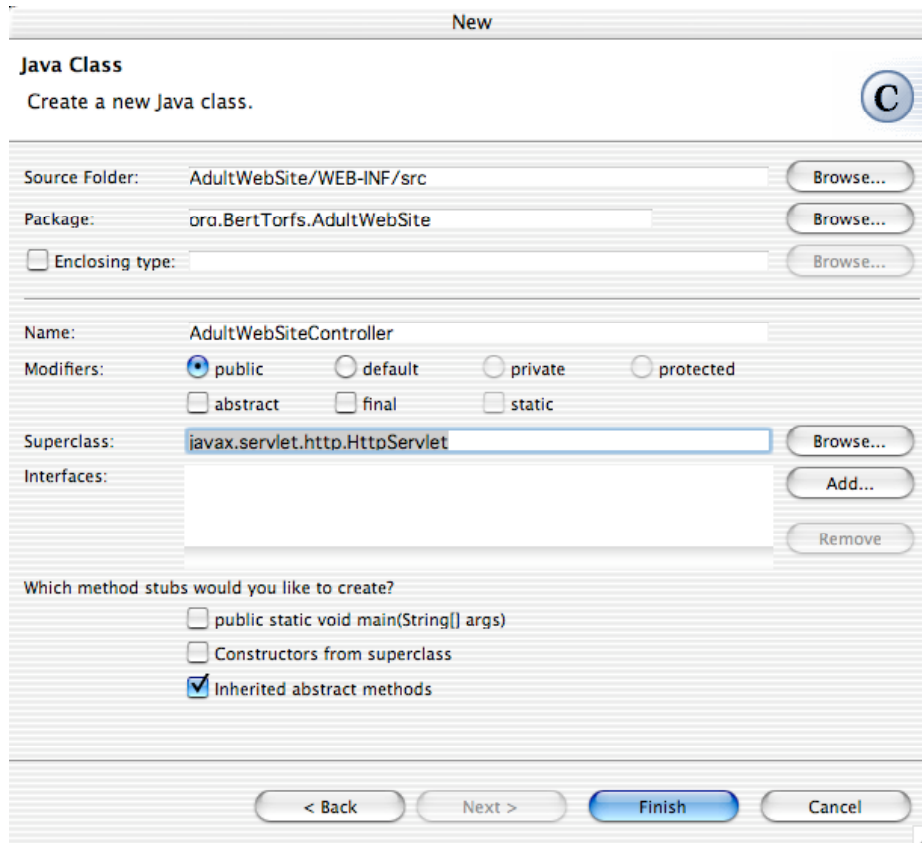
Designers should make HTML pages. If not, the pages risk to look like the one above. The code to test if somebody is an adult should be written by a developer. Stuff written by designers and stuff written by developers should not be mixed – they do not match and are created with different tools. Mixing a complex HTML document with a complex Java program gives an overly complex jsp file that nobody understands. Please consult an article on Model-View-Controller if you want to learn more on separating interface and business logic.

So lets create a controller object. For faceless HTML communication, J2EE offers us Servlets. To add a servlet to our project, we right-click the AdultWebSite project, and select 'New->Other...'

- Select Java and Class. Press 'Next'



- Give the class a name and a location as shown in the screenshot below. Make the class inherit from HttpServlet.



- And press 'Finish'.
- The following file will be created :

```

package org.BertTorfs.AdultWebSite;
import javax.servlet.http.HttpServlet;
/**
 * @author bert
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 * To enable and disable the creation of type comments go to
 * Window>Preferences>Java>Code Generation.
 */
public class AdultWebSiteController extends HttpServlet
{
}

```

- Just add the following in between the two brackets:

```

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
}

```

- Then place the caret just after 'HttpServletRequest' and press <Command-Space>. Do the same with the insertion point after HttpServletResponse, ServletException and IOException. This will make Eclipse add the necessary imports to the file.
- Copy the 'doPost' method and paste it in again. Name the new method 'doGet'.
- Paste again, and name the new method doAnything. Forward both toGet and doPost to doAnything. Output some http in the 'doAnything' method. Here is what my code looks like:

```

protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
    doAnything(request, response);
}

```

```

protected void doGet(HttpServletRequest arg0, HttpServletResponse arg1)
    throws ServletException, IOException
{
    doAnything(arg0, arg1);
}

protected void doAnything(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
{
    response.setContentType( "text/html" );
    PrintWriter out = response.getWriter();
    out.println( "<html>" );
    out.println( "<head>" );
    out.println( "<title>A Sample Servlet</title>" );
    out.println( "</head>" );
    out.println( "<body>" );
    out.println( "<h1>A Sample Servlet</h1>" );
    out.println( "</body>" );
    out.println( "</html>" );
    out.close();
}

```

- Eclipse will probably complain about PrintWriter. Just place the insertion point after "PrintWriter" and press <cmd-space> to add the necessary imports.

## Route the action to the servlet

Out welcome page contains the following <form> tag.

```

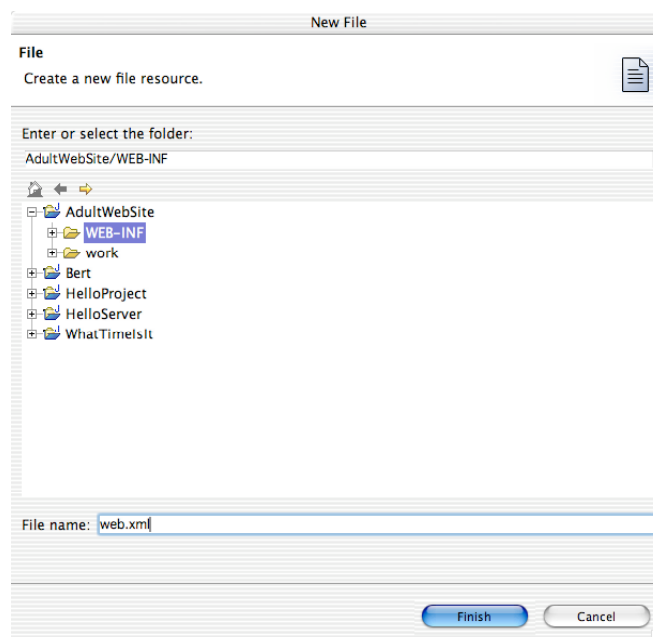
<form action="/AdultWebSite/SiteController/WhichPage" name="sender"
method="post">

```

Whenever we push the submit button, the URL <http://127.0.0.1/AdultWebSite/SiteController/WhichPage> will be called. If we want this URL to point to our servlets 'doPost' method, we have to add a web.xml file to our project.

There is one already one Web.xml in the Tomcat folder (tomcat/conf/web.xml). That file contains already some default servlet declarations with their mappings. We have to add our own. Project (or application) specific web-xml files are stored in the WEB-INF folder of the application (or project).

- <Ctrl-Click> (or right-click if you have a multi-button mouse) the WEB-INF folder of the project and choose <New->File> from the menu.
- Name the new file web.xml and click 'Finish'.



- Then edit the file and add the following XML (I copied the initial version from the web.xml file from the conf directory).

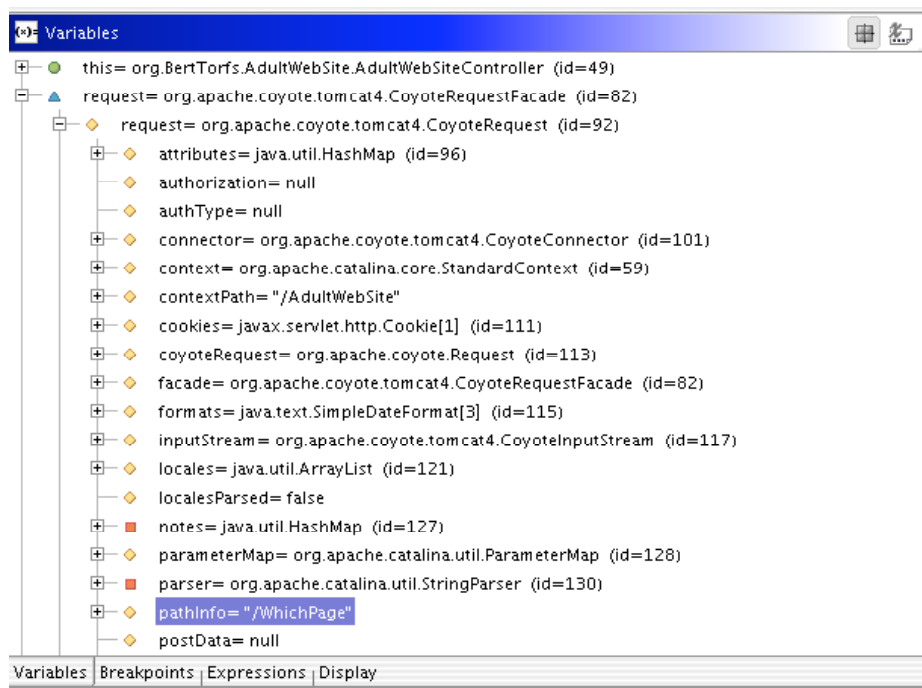
```


<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <servlet>
    <servlet-name>Process</servlet-name>
    <servlet-class>org.BertTorfs.AdultWebSite.AdultWebSiteController</servlet-class>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
  </servlet>

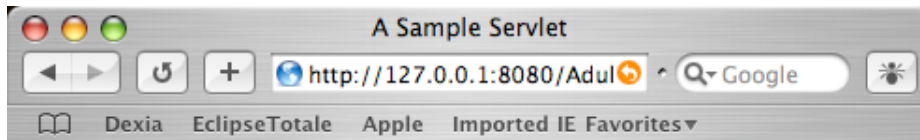
  <servlet-mapping>
    <servlet-name>Process</servlet-name>
    <url-pattern>/SiteController/*</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>GetInfo.jsp</welcome-file>
  </welcome-file-list>
</web-app>

```

- The first <servlet> tag and its matching <servlet-mapping> tag links the servlet named org.BertTorfs.AdultWebSite.AdultWebSiteController to the URL http://machine:Port/AdultWebSite/SiteController/\*. The '\*' can be anything, so we can have multiple URLs that all go to our controller servlet. You do not have to specify the context root (/AdultWebSite) in the servlet mapping.
- We also added an entry to the welcome-file list. That way, just typing AdultWebSite will bring us to our opening page.
- Now put a breakpoint inside the 'doAnything' method of the servlet, fire up Tomcat, and open your favorite browser on the page <http://127.0.0.1:8080/AdultWebSite/> (I am behind a proxy, so localhost does not work for me. Your mileage may vary). You can omit the 'index.jsp' from the URL, as that was specified as the welcome-file.
- When the page opens, press the 'Let me in' button.
- Your servlet will be opened in the source pane with the line where you placed a breakpoint selected. As your debugging holds up the execution of the request, chances are that your browser will give an error, telling that your site could not be opened in a reasonable time.
- Open the variables pane of the debug perspective, and click on the small '+' next to 'Request'. You'll see something like this :



- Please notice the pathInfo attribute of the request parameter. It contains the last part (the wildcard part) of the url we used to access this page. If our controller is used by a lot of pages, we can use this mechanism to tell the servlet where we came from and what we want.
- Remove the breakpoint (doubleClick the blue ball in the margin of the source pane) and press the  button to resume execution.
- As our web browser lost his patience by now, there will probably an error message about time-outs in the browser. So we try again, now without being held up by the debugger. So browse to <http://127.0.0.1:8080/AdultWebSite/> again and press 'Let Me In'. This will be the result:



## A Sample Servlet

- Just for fun, try to go to <http://127.0.0.1:8080/AdultWebSite/WEB-INF/JSP/Adults.jsp>. You'll get an error message from Tomcat. Pages placed inside the WEB-INF folder cannot be viewed or downloaded from a browser. Only you can serve them, no one can get them. People on the web can never look at the source code of your JSP's when they are placed inside WEB-INF. Basic security at no cost!

### Make the servlet open the other pages

Now there is one more thing to do. In stead the 'A sample servlet' message, we want to see actual adult content if we are over 18. So lets remove whatever we wrote in the servlets doAnything method, and make it control the other jps's. Here is what you should do :

- Add another method – doForward - to the servlet. Here is the code :

```
protected void doForward(String forward, HttpServletRequest req,
                          HttpServletResponse resp)
{
    RequestDispatcher rd = getServletContext().getRequestDispatcher(forward);
    // Delegate the processing of this request
    try
    {
        rd.forward(req, resp);
    }
    catch (ServletException e)
    {
    }
    catch (IOException e)
    {
    }
}
```

This method basically opens the JSP page whose name and path is passed as a string via the forward parameter. Please ignore the non-existent exception handling – we should give the user a descend error message in stead of a stack trace.

- Next, we enter the following in the 'doAnything' method

```
protected void doAnything(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
    String ls_year = (String) request.getParameter("year");
    String ls_action = (String) request.getPathInfo();
    String ls_NextPage;
```

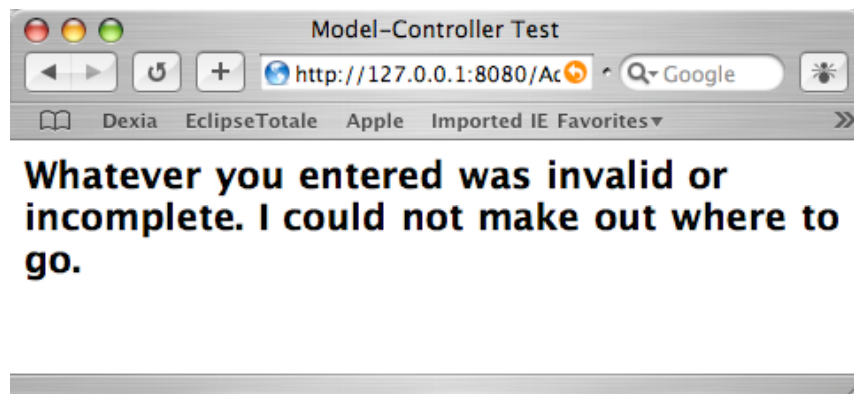
```

ls_NextPage = null;

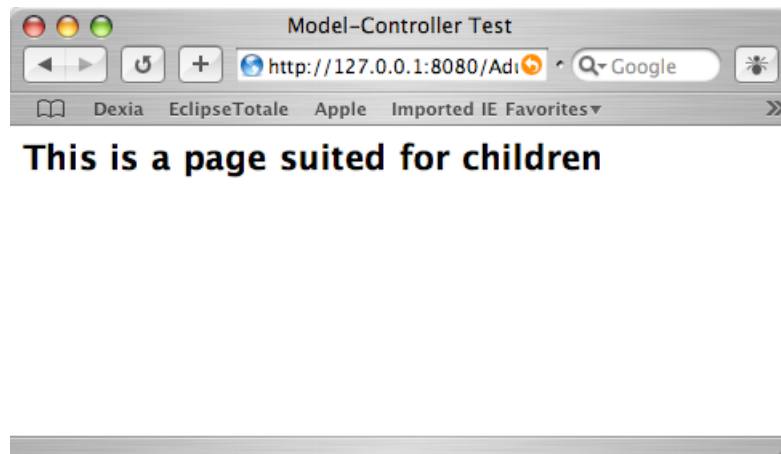
if (ls_action.compareToIgnoreCase("/WhichPage").toString() == 0)
{
    long ll_year;
    try
    {
        ll_year = Long.parseLong(ls_year);
        if (ll_year <= 1985)
        {
            ls_NextPage = new String("/WEB-INF/JSP/Adults.jsp");
        }
        else
        {
            ls_NextPage = new String("/WEB-INF/JSP/Children.jsp");
        }
    }
    catch (NumberFormatException e)
    {
        ls_NextPage = new String("/WEB-INF/JSP/InvalidInput.jsp");
    }
}
if (ls_NextPage == null)
    ls_NextPage = new String("/WEB-INF/JSP/InvalidInput.jsp");
doForward(ls_NextPage, request, response);
}

```

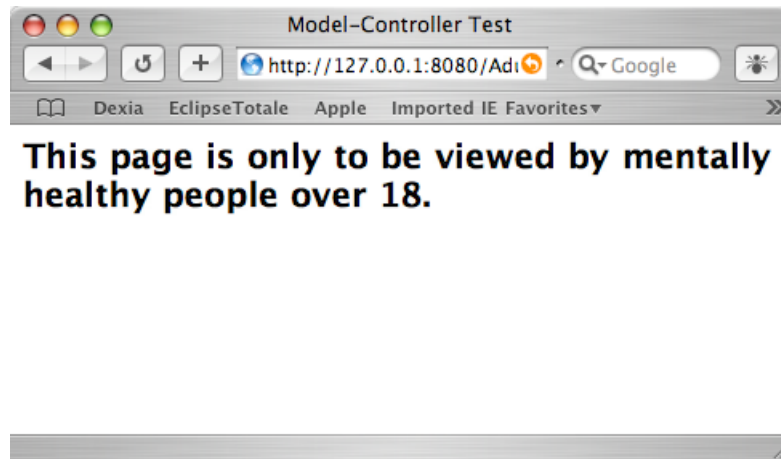
- And test the page. Start Tomcat, open the URL <http://127.0.0.1:8080/AdultWebSite/>, and press the 'Let me In' button without entering anything. You should see this :



- Go back, and enter 2002 as the year of birth. You should see this



- Go back, enter 1875 as the Birthdates year, and you'll get this:



## Add the Model

(to be done – we will create a session bean with one method ‘isAdult(birthday)’ that will be used by the controller servlet)

## Store and get preferences and settings in XML files

To be done. The legal age to be considered ‘adult’ will be read from an XML file. So will the names of the JSP pages our controller forwards to.

## What’s next

Next comes logging, Strut, writing an eclipse plugin that calls Middlegen straight from within a database browser running in Eclipse, communicating with clients running on Palm or Symbian devices over Bluetooth etc.... When time allows, I’ll complement – or write new articles – on the topics described above. If all of you register a copy of [WhereDidAllMyMoneyGo?](#), I could start writing tomorrow :). Otherwise, you will need some more patience.

Bert Torfs