SimpleFormServlet.java

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

/\*\*

\* A basic servlet that outputs a form and uses context

\* and initialization parameters.

\*/

public class SimpleFormServlet extends HttpServlet {

private String bankName;

private String bankAddress;

private String bankURL;

private String siteAdmin;

String url = "/servlet/SimpleBankForm";

public void init() {

//get the initialization parameters

ServletConfig config = getServletConfig();

bankName = config.getInitParameter("bankName");

bankAddress = config.getInitParameter("bankAddress");

//get the configuration parameters

ServletContext context = this.getServletContext();

bankURL = context.getInitParameter("bankURL");

siteAdmin = context.getInitParameter("siteAdmin");

}

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws IOException, ServletException

{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +

"Transitional//EN\">");

out.println("<html>");

out.println("<body bgcolor=\"#F3FBFE\">");

out.println("<head>");

out.println("<title>" + "Simple Form Servlet" + "</title>");

out.println("</head>");

out.println("<body>");

out.println("<h1>" + "Simple Bank Form" + "</h1>");

out.println(bankName + ", ");

out.println(bankAddress + "<br><hr>");

out.println("<FORM METHOD=POST ACTION=" + url + ">");

out.println("Please input your name: <INPUT NAME=userName" +

" SIZE=40><br>");

out.println("Please input your address: <INPUT NAME=" +

"userAddress SIZE=60><br>");

out.println("Please input your account number: <INPUT NAME" +

"=userAccount SIZE=10><br><br>");

out.println("<INPUT TYPE=SUBMIT VALUE='Submit Details'>");

out.println("</FORM>");

out.println("<hr>Home page: " + bankURL + "<br>");

out.println("Administrator: " + siteAdmin);

out.println("</body>");

out.println("</html>");

}

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws IOException, ServletException

{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String name = request.getParameter("userName");

String address = request.getParameter("userAddress");

int account = Integer.parseInt(request.getParameter

("userAccount"));

out.println("<html>");

out.println("<body bgcolor=\"#F3FBFE\">");

out.println("<h1>" + "Customer Account Details" + "</h1>");

out.println(bankName + ", ");

out.println(bankAddress + "<br><hr>");

out.println("<p>The Name you entered was: " + name + "</p>");

out.println("<p> The Address you entered was: " + address +

"</p>");

out.println("<p> The Account number you entered was: " +

account + "</p>");

out.println("<hr>Home page: " + bankURL + "<br>");

out.println("Administrator: " + siteAdmin);

out.println("</body>");

out.println("</html>");

}

}

/\* 1. Import the necessary packages for the servlet.\*/  
  
/\*\*  
\* A basic servlet that processes log-in information.  
\*/  
  
/\* 2. Set up the servlet class to inherit from the correct class. \*/  
public class LoginFormServlet   {  
     
   String url = "/servlet/LoginFormServlet";  
     
   public void init() {  
   }  
  
/\* 3. Implement the doGet function to output a HTML form to obtain  
   the user's details. Use the following print statements to create  
   the form.  
  
   out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +  
         "Transitional//EN\">");  
      out.println("<html>");  
   out.println("<body bgcolor=\"#F3FBFE\">");  
   out.println("<head>");  
      out.println("<title>" + "Bank application log-in page" + "</title>");  
       out.println("</head>");  
      out.println("<body>");  
      out.println("<h1>"+ "Bank application log-in page"+ "</h1>");  
      out.println("<hr><br>");  
      out.println("Please enter your log-in information:");  
      out.println("<FORM METHOD=POST ACTION=" + url + ">");  
      out.println("Please input your name: <INPUT NAME=userName" + " SIZE=40><br>");  
      out.println("Please input your password: <INPUT NAME=" + "password SIZE=10><br><br>");  
      out.println("<INPUT TYPE=SUBMIT VALUE='Submit Details'><br>");  
      out.println("</FORM>");  
      out.println("<hr><br>");  
      out.println("</body>");  
   out.println("</html>");  
   \*/  
  
   /\* 4. Implement the doPost function to retrieve the user's details  
      from the form and print them out. Use the following print  
      statements for the HTML. Note, this code assumes the user's name  
      has been stored in a variable called name, and the password stored  
      in a variable called password.  
  
      out.println("<html>");  
      out.println("<body bgcolor=\"#F3FBFE\">");  
      out.println("<h1>" + "Log-in Details" + "</h1>");  
  
      out.println("<p>The name you entered was: " + name + "</p>");  
      out.println("<p> The password you entered was: " + password + "</p>");  
  
      out.println("</body>");  
      out.println("</html>");  
   \*/  
     
}

**CourseSelectorServlet**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import javax.naming.\*;

import CourseSelectorHome.\*;

import javax.rmi.PortableRemoteObject;

/\*\*

\* A servlet client that accesses the CourseSelector session bean to

\* manage courses in a university application.

\*/

public class CourseSelectorServlet extends HttpServlet {

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws IOException, ServletException {

String message = "The courses have been successfully added";

try {

//locate & instantiate the CourseSelector bean

Context context = new InitialContext();

Object ref = context.lookup("courses");

CourseSelectorHome home =

(CourseSelectorHome) PortableRemoteObject.narrow

(ref, CourseSelectorHome.class);

CourseSelector cs = home.create();

// Now add some courses to the CourseSelector bean

Course course = new Course("PHIL03", "Modern Philosophy",

"Undergraduate", "Philosophy");

cs.addCourse(course);

course = new Course("POL02", "European Politics",

"Undergraduate", "Politics");

cs.addCourse(course);

//remove a course from the list

cs.removeCourse(course);

//remove the CourseSelector bean

cs.remove();

} catch (Exception ex) {

message = "Error: The courses not have been added.";

}

//output a page to show the results of accessing

//the CourseSelector bean

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("<html>");

out.println("<body bgcolor=\"#F3FBFE\">");

out.println("<h1>" + "CourseSelector bean" + "</h1>");

out.println(message);

out.println("</body>");

out.println("</html>");

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws IOException, ServletException {

doGet(request, response);

}

}

/\*\*

\* Created October 29th, 2002

\*/

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

/\*\*

\* Front-controller servlet for an online travel agency.

\* Directs booking requests to the appropriate JSP.

\*/

public class TravelInfo extends HttpServlet {

public void init() {

}

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

String userName = request.getParameter("userName");

String password = request.getParameter("password");

HttpSession session = request.getSession(true);

session.setAttribute("userName", userName);

if (request.getParameter("flights") != null) {

gotoPage("/travel-info/Flights.jsp",

request, response);

} else if (request.getParameter("cars") != null) {

gotoPage("/travel-info/Cars.jsp",

request, response);

} else if (request.getParameter("hotels") != null) {

gotoPage("/travel-info/Hotels.jsp",

request, response);

} else if (request.getParameter("account") != null) {

gotoPage("/travel-info/UserInfo.jsp",

request, response);

} else {

gotoPage("/travel-info/Error.jsp",

request, response);

}

}

private void gotoPage(String address,

HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

RequestDispatcher dispatcher =

getServletContext().getRequestDispatcher(address);

dispatcher.forward(request, response);

}

}

# AccessLogFilter.java

package filters;

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.util.\*;

/\*\*

\* A basic filter that updates the log file

\* whenever an associated servlet or JSP is accessed.

\*/

public class AccessLogFilter implements Filter {

private FilterConfig config;

public void init(FilterConfig config)

throws ServletException {

this.config = config;

}

public void doFilter(ServletRequest request,

ServletResponse response,

FilterChain chain)

throws ServletException, IOException {

String filterName = config.getFilterName();

HttpServletRequest req = (HttpServletRequest)request;

ServletContext context = config.getServletContext();

context.log(new Date() + ": " + "Access to: " +

req.getRequestURL() +

" requested by: " +

req.getRemoteHost() +

". (Reported by " + filterName + ".)");

chain.doFilter(request,response);

}

public void destroy() {

this.config = null;

}

}

# HitCountFilter.java

package filters;

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.util.\*;

/\*\*

\* A basic filter that accesses a hit counter and updates

\* the log file whenever an associated servlet or JSP is accessed.

\*/

public class HitCountFilter implements Filter {

private FilterConfig config;

public void init(FilterConfig config) throws ServletException {

this.config = config;

}

public void doFilter(ServletRequest request,

ServletResponse response,

FilterChain chain)

throws ServletException, IOException {

String filterName = config.getFilterName();

HttpServletRequest req =(HttpServletRequest)request;

ServletContext context = config.getServletContext();

//retrieve the counter from the ServletContext

Integer ctr =(Integer)(context.getAttribute

("counter"));

//update the log with the value of the counter

context.log(new Date() + ": The current hit count is "

+ ctr + ". (Reported by " + filterName + ".)");

chain.doFilter(request,response);

}

public void destroy() {

this.config = null;

}

}

# web.xml

<?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>

<context-param>

<param-name>bankURL</param-name>

<param-value>www.imagenie.com</param-value>

</context-param>

<context-param>

<param-name>siteAdmin</param-name>

<param-value>webmaster@www.imagenie.com</param-value>

</context-param>

<filter>

<filter-name>HitCount</filter-name>

<filter-class>filters.HitCountFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>HitCount</filter-name>

<servlet-name>SimpleBankForm</servlet-name>

</filter-mapping>

<servlet>

<servlet-name>SimpleBankForm</servlet-name>

<display-name>SimpleBankForm</display-name>

<servlet-class>SimpleFormServlet</servlet-class>

<init-param>

<param-name>bankName</param-name>

<param-value>Imagenie</param-value>

</init-param>

<init-param>

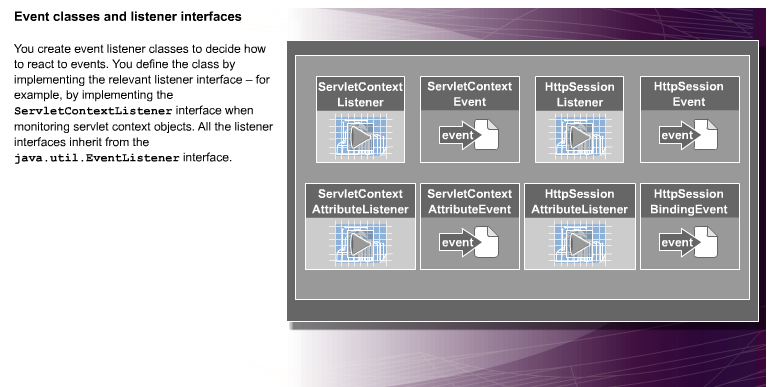
<param-name>bankAddress</param-name>

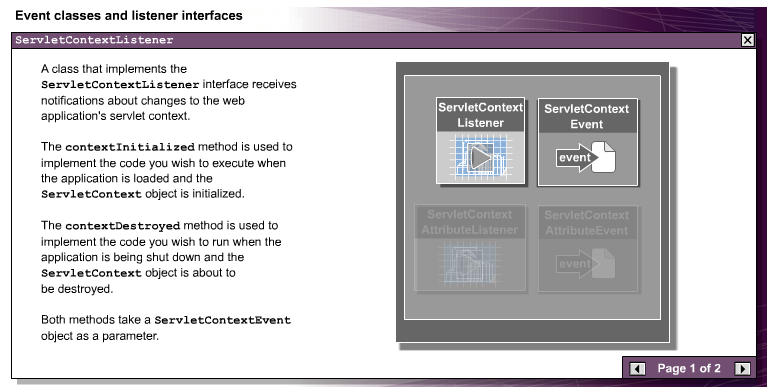
<param-value>123 Madison Drive</param-value>

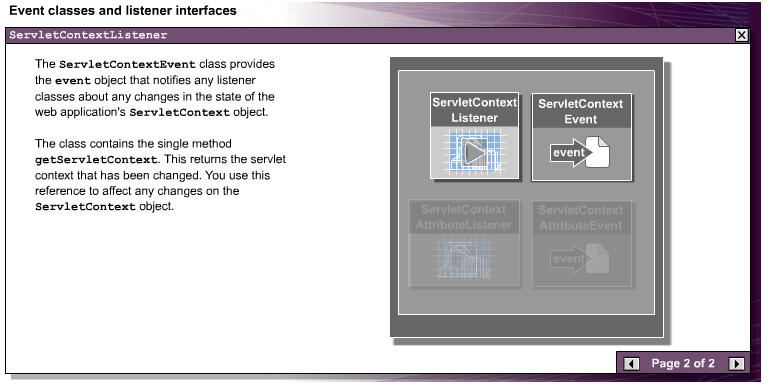
</init-param>

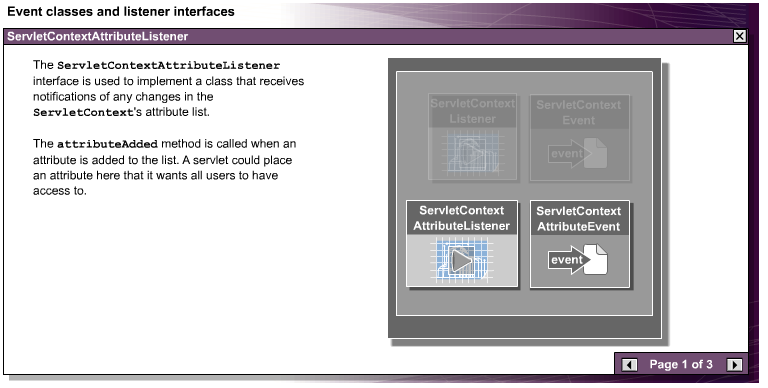
</servlet>

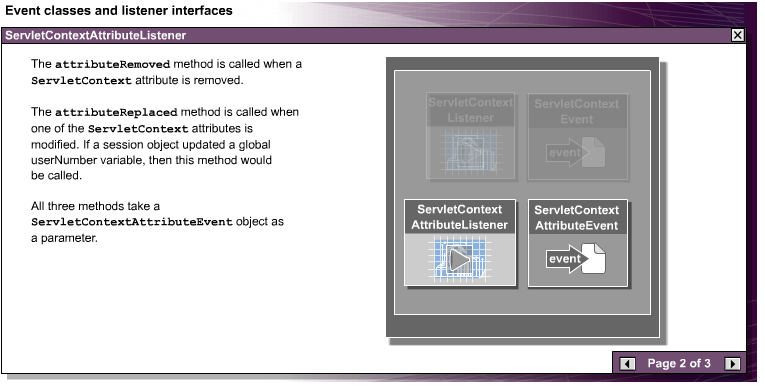
</web-app>

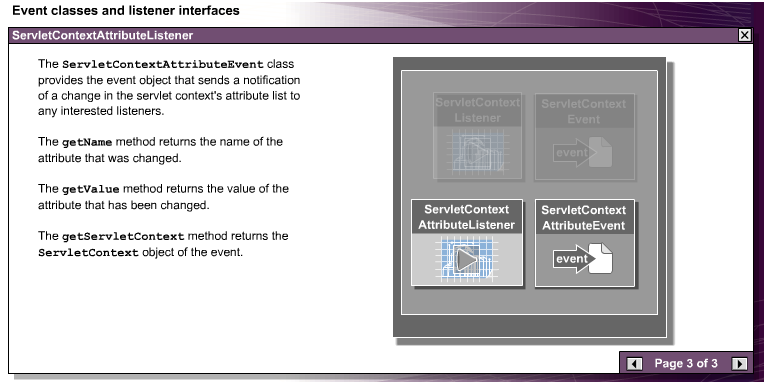


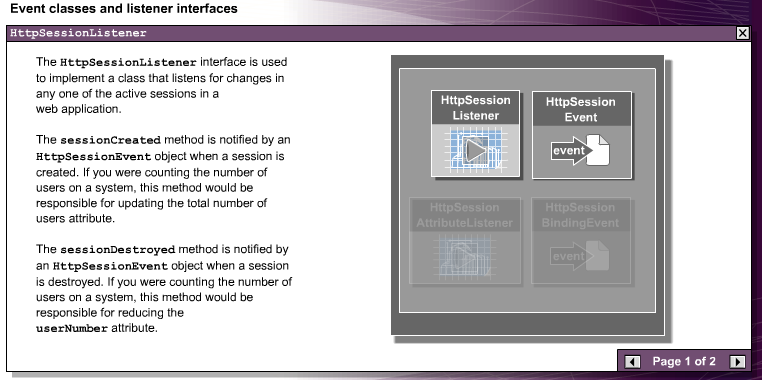


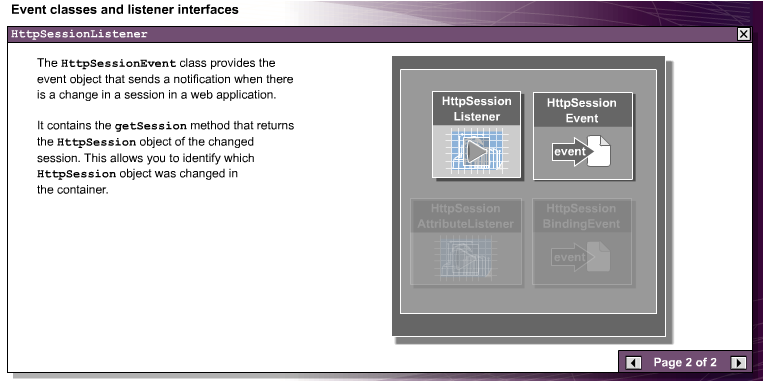


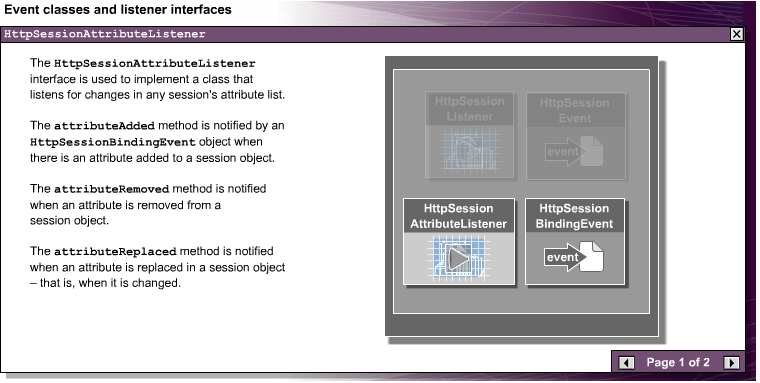


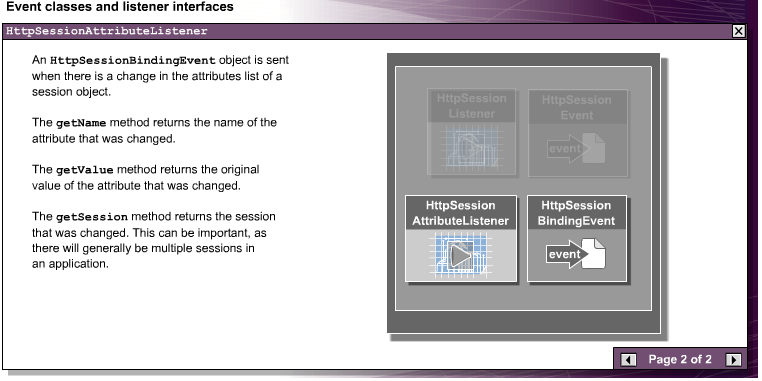


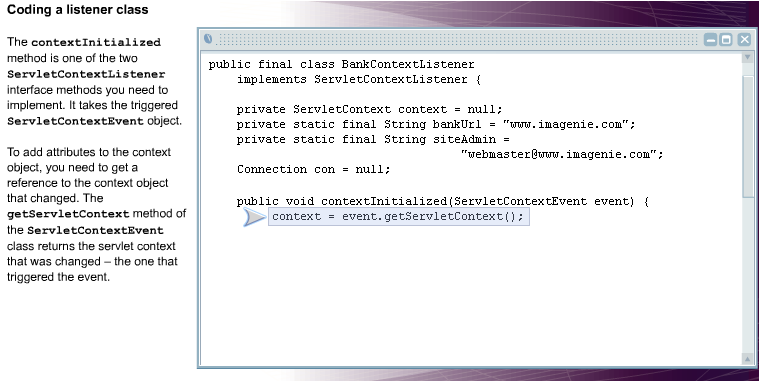


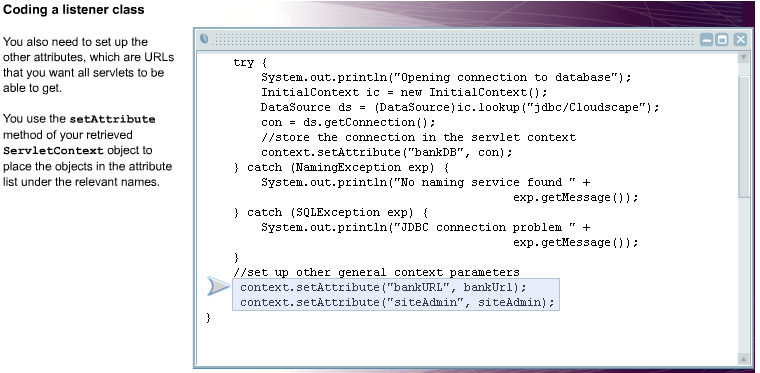


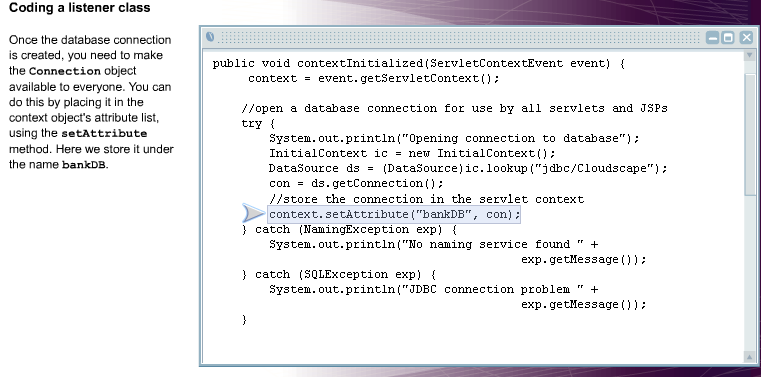


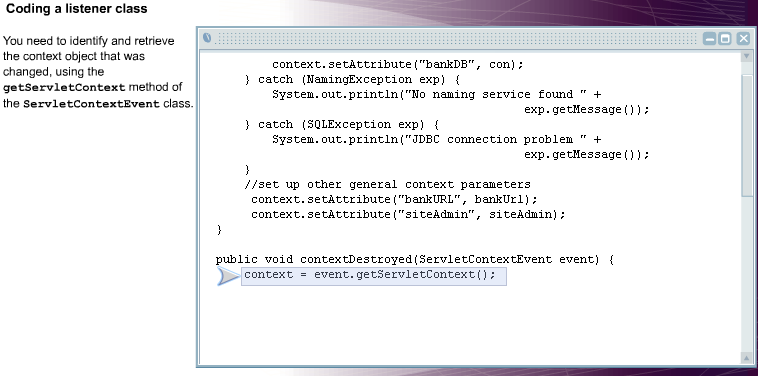


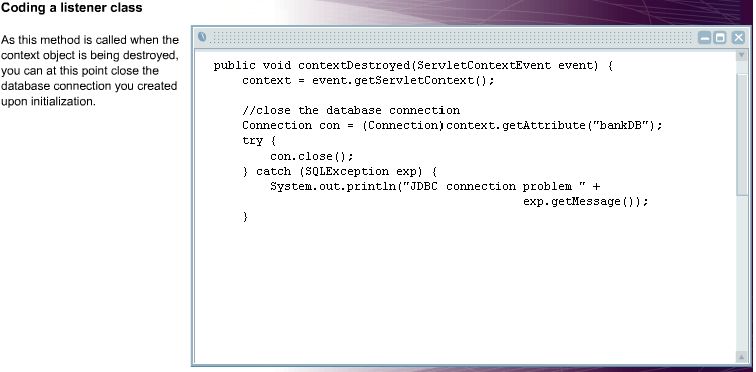


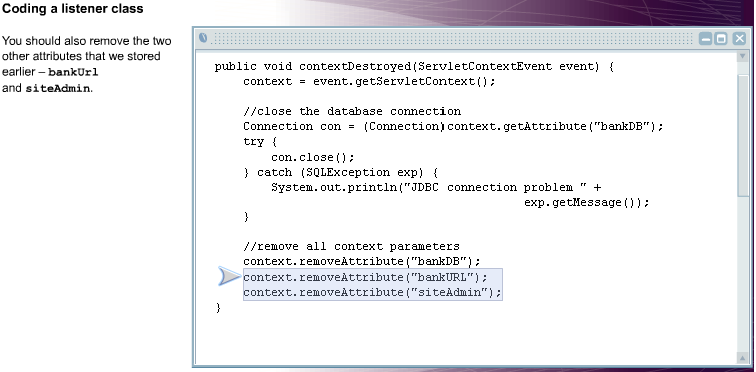


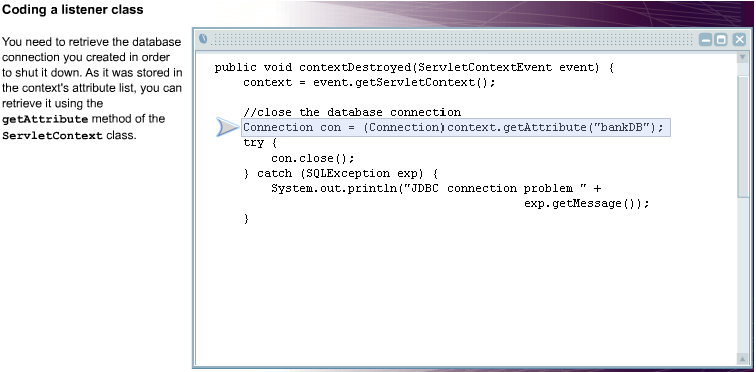


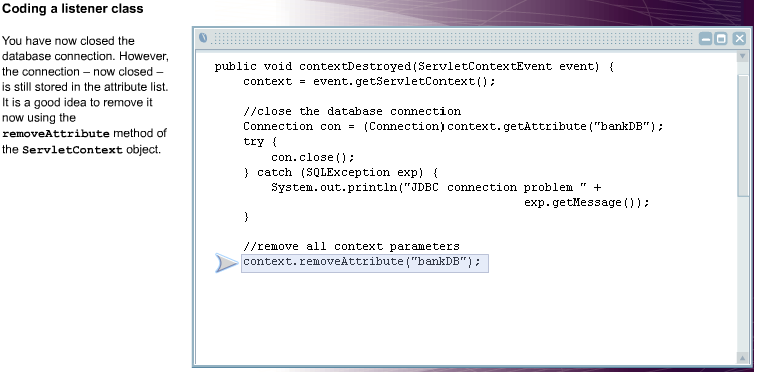












# BankContextListener.java

/\*

\* Simple example of a listener for a ServletContext.

\*/

package listeners;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.sql.\*;

import javax.sql.\*;

import javax.naming.\*;

public final class BankContextListener

implements ServletContextListener {

private ServletContext context = null;

private static final String bankUrl = "www.imagenie.com";

private static final String siteAdmin =

"webmaster@www.imagenie.com";

Connection con = null;

public void contextInitialized(ServletContextEvent event) {

context = event.getServletContext();

//open a database connection for use by all servlets

//and JSPs

try {

System.out.println("Opening connection to database");

InitialContext ic = new InitialContext();

DataSource ds =

(DataSource)ic.lookup("jdbc/Cloudscape");

con = ds.getConnection();

//store the connection in the servlet context

context.setAttribute("bankDB", con);

} catch (NamingException exp) {

System.out.println("No naming service found " + exp.getMessage());

} catch (SQLException exp) {

System.out.println("JDBC connection problem " + exp.getMessage());

}

//set up other general context parameters

context.setAttribute("bankURL", bankUrl);

context.setAttribute("siteAdmin", siteAdmin);

}

public void contextDestroyed(ServletContextEvent event) {

context = event.getServletContext();

//close the database connection

Connection con =

(Connection)context.getAttribute("bankDB");

try {

con.close();

} catch (SQLException exp) {

System.out.println("JDBC connection problem " + exp.getMessage());

}

//remove all context parameters

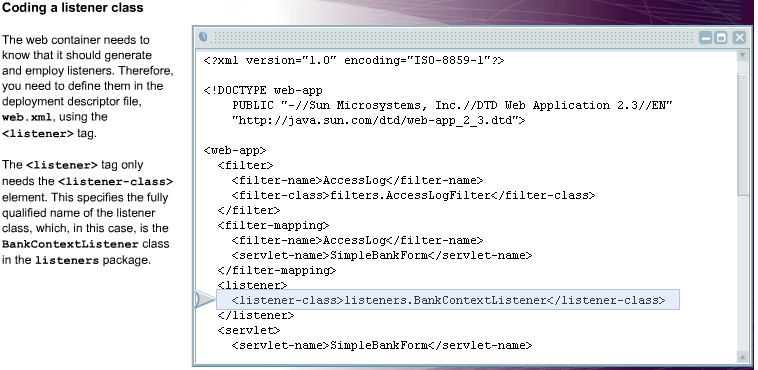
context.removeAttribute("bankDB");

context.removeAttribute("bankURL");

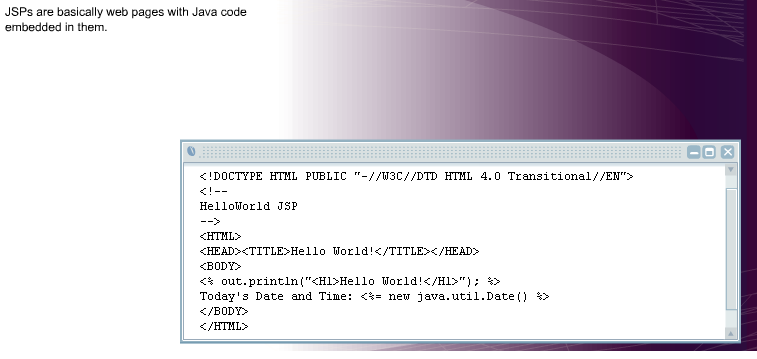
context.removeAttribute("siteAdmin");

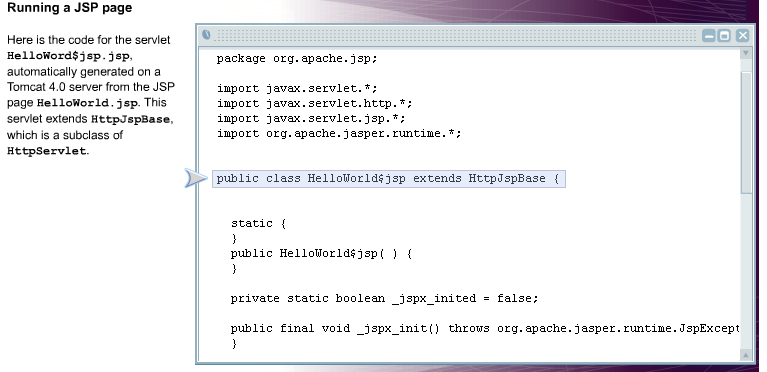
}

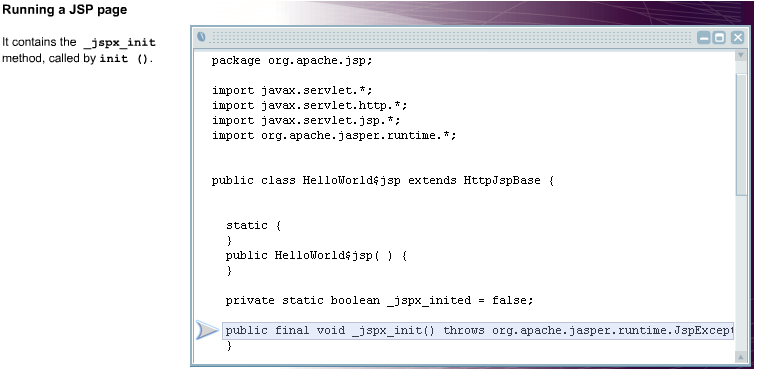
}

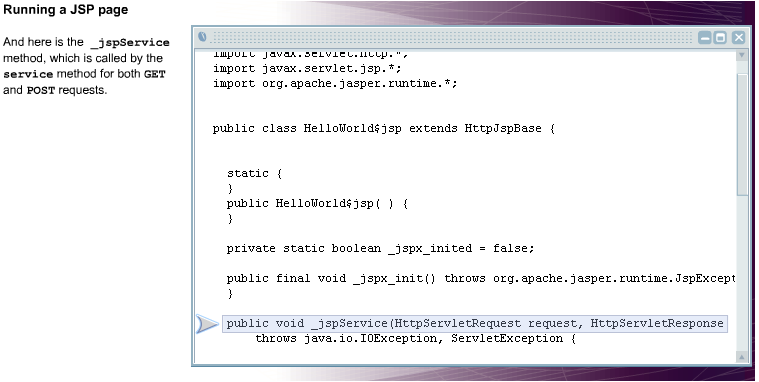


**JSPs(Java Server Pages)**

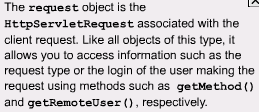
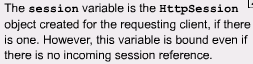


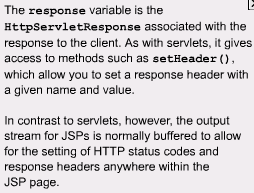
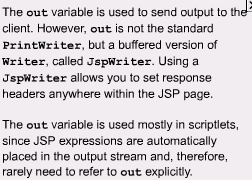


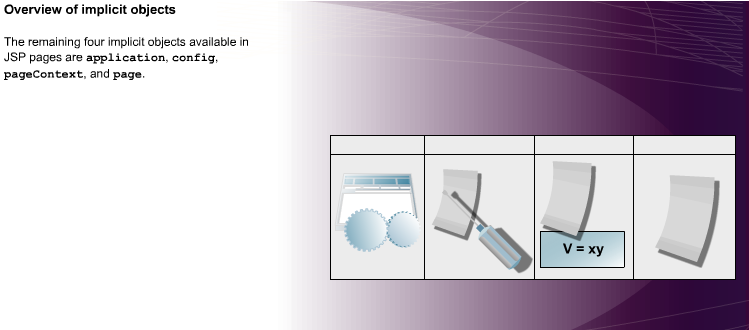


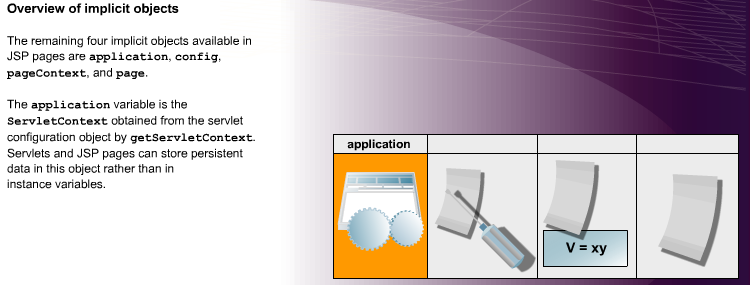


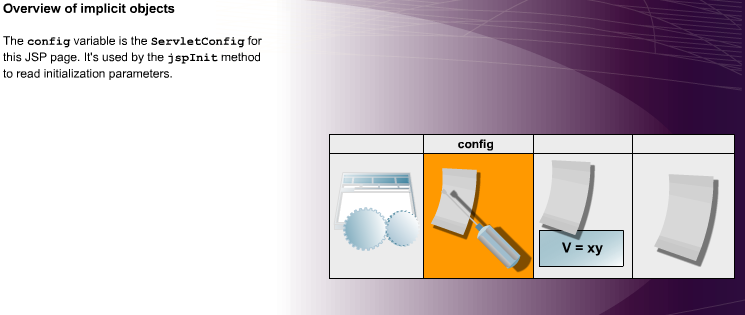


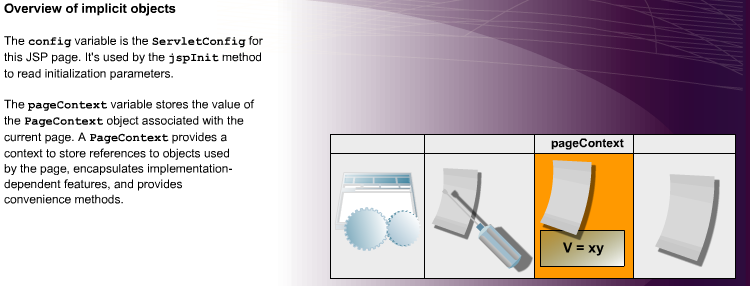
 

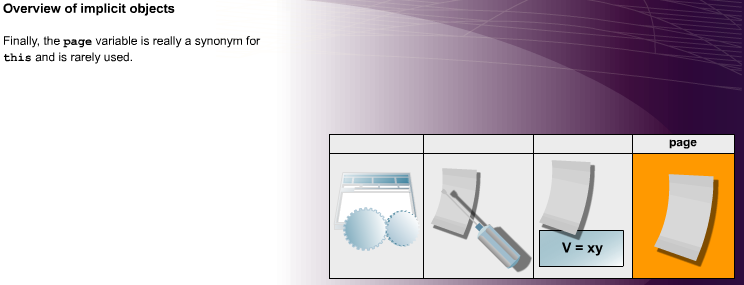
 

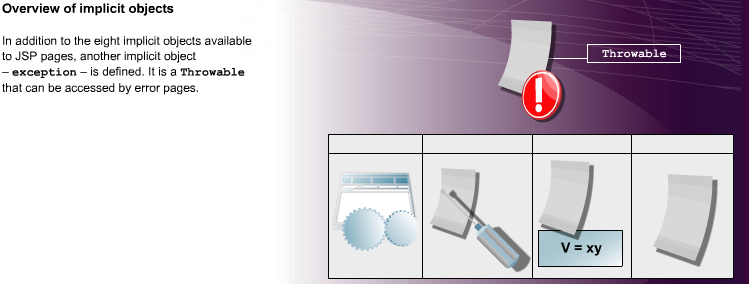












# JSP directives

## Abstract

This white paper discusses the three JavaServer Page (JSP) directives – page, include, and taglib. It provides details of the attributes and capabilities of each directive, and the syntax that is used for each.

## Function of JSP directives

JavaServer Page (JSP) directives are one of the three types of JSP constructs that are used in JSP pages. They are messages to the JSP container that allow you to affect the overall structure of the servlet that results from the JSP page. Directives do not produce any output into the current out stream, unlike other types of constructs such as JSP expressions.

### Syntax

JSP directives are enclosed within opening <%@ and closing %> tags. They consist of the directive name, one or more attributes, and their associated values:

<%@ directive attribute ="value " %>

or

<%@ directive attribute1 ="value1 " attribute2 ="value2 " ... attributeN ="valueN " %>

You must include quotation marks around the attribute value, but you can use single or double quotes. This alternative XML syntax can also be used:

<jsp:directive.directivename attribute1 ="value1 " attribute2 ="value2 " ... attributeN ="valueN " />

However, you cannot mix the two versions in the same JSP page.

### Three types of directive

Three types of JSP directive can be used in JSP pages – page, which allows you to define a number of page-dependent properties, include, which lets you insert a file into the servlet class, and taglib, which is used in conjunction with JSP custom tags. Let's take a look at each of these directives, and the kinds of attributes they take.

## Configuring the servlet with the page directive

The page directive allows you to control the structure of the servlet derived from the JSP page. These directives can be placed anywhere within the page and the JSP container integrates them when the page is being translated into the servlet. In JSP 1.2, the page directive can take 12 distinct attributes – import, extends, session, isThreadSafe, buffer, autoFlush, contentType, pageEncoding, errorPage, isErrorPage, language, and info.

### The import attribute

The import attribute is the JSP version of a standard import statement. It lets you specify the packages that are imported by the servlet that results from the translation of the JSP page. The import attribute can appear multiple times within the same JSP page – it's the only page attribute allowed to do so – but most developers find it convenient to group any directives taking this attribute near the top of the page. It takes the following forms:

<%@ page import="package.class " %>  
<%@ page import="package.class1,...,package.classN " %>

Here's an example in which the java.util and java.sql packages are imported:

<%@ page import="java.util.\*, java.sql.\*" %>

You don't need import attributes for the java.lang.\*, javax.servlet.\*, javax.servlet.jsp.\*, and javax.servlet.http.\* packages, because these are imported automatically.

### The extends attribute

The extends attribute allows you to specify the superclass of the servlet into which the JSP page is translated:

<%@ page extends="somepackage.someclass " %>

This attribute should only be used by experienced developers, as it restricts the ability of the JSP container to provide specialized superclasses that may improve the quality of service.

### The session attribute

This attribute is used to indicate whether the JSP page participates in an HTTP session. Therefore, it has only two possible values, true – the default – or false.

<%@ page session="true" %>  
<%@ page session="false" %>

If the value is true, the implicit object session (an HttpSession) exists.

### The isThreadSafe attribute

By default, servlets create multiple threads in response to simultaneous user requests; each thread concurrently accessing the service method of the same servlet instance. This behavior assumes the servlet is thread safe. The isThreadSafe attribute takes only two values:

<%@ page isThreadSafe="true" %>

or

<%@ page isThreadSafe="false" %>

The default value is true, which means that the servlet is thread safe and can handle concurrent access to the service method. If the value is false, however, the JSP container will queue multiple outstanding client requests to the page implementation for processing.

### The buffer attribute

This attribute specifies the size of the buffer used by the out implicit object, which is a JspWriter. It can specify that there is no buffering, by using the none value, or that a specific buffer size, measured in kilobytes, is used.

<%@ page buffer="none" %>  
<%@ page buffer="sizekb" %>

The kb suffix is mandatory. Because this value is a minimum, servers can use a larger buffer than you specify, but not a smaller one. The default buffer size is server-specific, but must be at least 8 KB.

### The autoFlush attribute

The autoFlush attribute takes two values:

<%@ page autoFlush="true" %>  
<%@ page autoFlush="false" %>

The default value – true – indicates that the output buffer should be flushed automatically when the buffer is filled. A value of false indicates that an exception should be raised when the buffer overflows. Clearly, you can't set autoFlush to false if you have set the size of the buffer to zero.

### The contentType attribute

The value of this attribute sets the corresponding HTTP 1.1 response header Content-Type, which gives the multipurpose Internet mail extensions (MIME) type of the document being sent to the client. This attribute can be used as follows:

<%@ page contentType="MIME-Type " %>  
<%@ page contentType="MIME-Type ; charset=Character-Set " %>

So, for example, the directive

<%@ page contentType="video/mpeg" %>

would indicate that an MPEG video clip is being returned. If you are sending text, you can also indicate the character set of this text. This is useful for backwards compatibility, but JSP 1.2 allows you to specify the character set directly using the pageEncoding attribute.

### The pageEncoding attribute

This attribute defines the character encoding for the JSP page, and its values are of the form "CHARSET", which is the Internet Assigned Numbers Authority (IANA) value for a character encoding. So, for example, this directive

<%@ page pageEncoding="GB\_2312-80" %>

might be used in a Chinese JSP page. The default value is ISO\_8859-1, the standard Latin character set.

### The errorPage attribute

This attribute defines a URL for a resource, to which any Throwable objects that are thrown, but not caught, by the current JSP page are forwarded for error processing. It's used as follows:

<%@ page errorPage="Relative URL " %>

This attribute is used for page-specific error pages.

### The isErrorPage attribute

This attribute specifies whether the current JSP page is an error page for some other JSP page. Like many page attributes, it takes two values, true or false, which is the default. The implicit exception object is available in error pages (errorPage=true), but not in other pages, where any reference to exception will result in a fatal translation error.

### The language attribute

This attribute defines the scripting language to be used in any JSP scripting elements that appear in the current JSP page. At the moment, it's somewhat redundant, as the only defined and required value in JSP 1.2 for this attribute is java. However, future versions of the JSP specification may define additional values for the language attribute and all such values are reserved.

### The info attribute

This attribute defines a string that is incorporated into the servlet that results from the translation of the JSP page. It takes the following form

<%@ page info="Some Message " %>

The string can be retrieved from the servlet by means of the getServletInfo method.

### XML Syntax for the page directive and its attributes

XML syntax can be used with all of the page attributes for all JSP 1.2 and some JSP 1.1 containers. For example, this directive

<jsp:directive.page buffer="32kb" />

sets the size of the buffer to 32 KB, whereas this directive

<jsp:directive.page isErrorPage="true" />

indicates that the current JSP page is an error page.

## Including files – the include directive and related actions

In JSP 1.2, you have two main mechanisms for including external data in your JSP pages – the include directive, which allows you to include JSP code, and the jsp:include action, which is easier to use but excludes the possibility of using any code. In addition, you can use the jsp:plugin action to insert those Java applets that are not supported by the HTML APPLET tag into your JSP page.

### The include directive

This directive allows you to insert a file into your JSP page at translation time, when the page is translated into a servlet. This directive takes a single attribute, file, which defines the location of the included file. Here's the syntax:

<%@ include file="relevantURLspec " %>

There is also an alternative XML syntax:

<jsp:directive.include file="relativeURLspec " />

Because the file is included at translation time, it can contain JSP constructs – expressions, scriptlets, and so on – that affect the main JSP page into which the file is placed. However, if the included file changes, all the JSP pages that use that file may have to change as well. This is a serious disadvantage, because the JSP 1.2 specification does not require that the JSP container is notified if the included file changes. So, if you alter the included file, you may have to manually update the modification dates of all JSP pages that include that file.

### The jsp:include action

The jsp:include action allows you to include the output of another file in the main JSP page at request time. The output is included in the current value of the out object for the main JSP page, so the included file has access only to JspWriter. Although this means that the included file cannot include code that affects the main page, it also implies that you do not have to update the main JSP page if the included file is changed. That is why developers prefer to use jsp:include in almost all cases. This action takes two attributes

<jsp:include page="RelativeURL " flush="true" />

where the required page attribute indicates the location of the file to be included, and the optional flush attribute specifies whether the output stream is to be flushed before the file is included. The default value for this attribute in JSP 1.2 is false (but the only allowed value in JSP 1.1 is true).

### The jsp:plugin action

Unfortunately, some web browsers – Internet Explorer 5.x or 6, or Netscape 4.x, for example – do not support the Java 2 platform. To address this problem, Sun developed a plug-in, which allows you to use Java 2 with such browsers. However, the standard APPLET tag will not work with this plug-in, so if you are developing an applet that uses Java 2, you have to use complex OBJECT (Internet Explorer) and EMBED (Netscape) tags. The function of the jsp:plugin action is to tell the server to generate HTML that contains the appropriate tag. This action can take several attributes, most of which correspond to those that can be used with the APPLET tag.

## Using JSP custom tags with the taglib directive

One of the most powerful features of JSP 1.2 are JSP custom tags, which are user-defined JSP language elements that encapsulate complex behavior. When a JSP page containing a custom tag is translated into a servlet, the tag is converted to operations on an object called a tag handler. The web container then invokes those operations when the JSP page's servlet is executed.

To use a custom tag, you must first create the tag handler class, and a tag library descriptor (TLD) file, which describes the tag's properties. The function of the taglib directive is to indicate, in the JSP page, that a custom tag will be called, and to provide the location of the TLD file. Here's the syntax:

<%@ taglib uri="TagLibraryURI " prefix="someprefix " %>

This directive only takes the two attributes shown here. The value of the uri attribute can be either an absolute or relative URL pointing to the location of the TLD file. The prefix attribute defines a prefix for the custom tag, which will be placed before the tag name, defined in the TLD, when the tag is actually called from the JSP page. Here's an example of how the taglib directive might be used:

<%@ taglib uri=http://www.imagenie.com/mytags  
prefix="tt" %>

And here's how a tag named SimpleTag would be called from within the JSP page:

<tt:SimpleTag />

## Summary

JavaServer Page (JSP) directives are messages to the JSP container that allow you to affect the structure of the servlet that results from the JSP page. Three types of JSP directives can be used – page, include, and taglib.

The page directive defines a number of page-dependent properties and communicates these properties to the JSP container. In JSP 1.2, the page directive can take 12 distinct attributes – import, extends, session, isThreadSafe, buffer, autoFlush, contentType, pageEncoding, errorPage, isErrorPage, language, and info.

In JSP 1.2, there are two main mechanisms for including external data in JSP pages – the include directive and the jsp:include action. The include directive inserts a file at translation time, when the JSP page is translated into a servlet, so you can include JSP code in the included file. With this directive however, you may have to update all JSP pages if the included file is modified. The jsp:include action on the other hand, inserts the output of an external file in the main JSP page at request time, so although you cannot include any code with this action, it's easier to use, as you don't have to alter the main page if the included file is changed. Another action, jsp:plugin, is used to insert Java applets that are not supported by the HTML APPLET tag into a JSP page.

One of the most powerful features of JSP 1.2 are JSP custom tags, which are user-defined JSP language elements that encapsulate complex behavior. The function of the taglib directive is to indicate, in a JSP page, that a custom tag will be called. This directive takes two attributes, one which provides the location of a descriptor file for the tag, and another that defines a prefix for the tag, which is used in when the tag is called in the JSP page.