Chapter 6

How to develop JSPs

Objectives

Applied

1. Create business classes that are JavaBeans.
2. Code and test JavaServer Pages that use any of the features described in this chapter.
3. Use EL to display properties of JavaBeans.
4. Use include files in your JSPs at compile-time or runtime.

Objectives (continued)

Knowledge

1. List the three rules for defining a JavaBean.
2. List the four scopes that EL searches in the sequence used by EL
3. List and describe one type of JSTL tag.
4. List the five types of old JSP tags and describe why they aren't typically used for new development.
5. Distinguish between EL and standard JSP tags.
6. Describe the use of include files.

The User bean class

```
package murach.business;

import java.io.Serializable;

public class User implements Serializable {

    private String firstName;
    private String lastName;
    private String email;

    public User() {
        firstName = "";
        lastName = "";
        email = "";
    }

    public User(String firstName, String lastName, String email) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }
```

The User bean class (continued)

```
    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

How to code a JavaBean

- A *JavaBean*, or *bean*, is a Java class that
  1. Provides a zero-argument constructor
  2. Provides get and set methods for all of its private instance variables that follow standard Java naming conventions
  3. Implements the Serializable or Externalizable interface.
- Since JavaBeans are just Java classes, they are a type of *plain old Java object* (*POJO*).

How to display an attribute

Syntax

```
${attribute}
```

Servlet code

```
GregorianCalendar currentDate = new GregorianCalendar();
int currentYear = currentDate.get(Calendar.YEAR);
request.setAttribute("currentYear", currentYear);
```

JSP code

```
<p>The current year is ${currentYear}</p>
```

How to display the property of an attribute

Syntax

```
${attribute.property}
```

Servlet code

```
User user = new User(firstName, lastName, email);
request.setAttribute("user", user);
```

JSP code

```
<p>Hello ${user.firstName}</p>
```

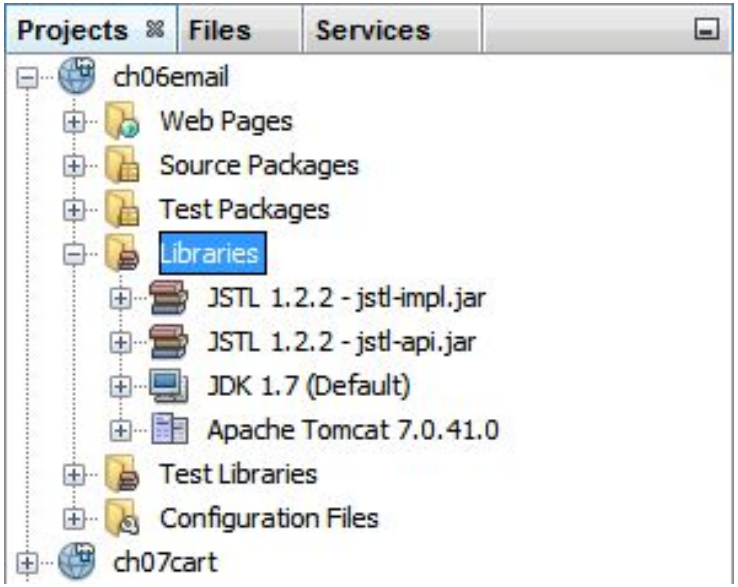
The sequence of scopes that Java searches to find the attribute

Scope	Description
`page`	The implicit PageContext object.
`request`	The HttpServletRequest object.
`session`	The HttpSession object.
`application`	The ServletContext object.

EL

- The *JSP Expression Language* (*EL*) makes it easy to access attributes and JavaBean properties from a request object.
- When you use the *dot operator* with a JavaBean, the code to the left of the operator specifies the *JavaBean*, and the code to the right of the operator specifies a *property* of the JavaBean.
- When you use this syntax, EL looks up the attribute starting with the smallest *scope* (page scope) and moving towards the largest scope (application scope).
- Attributes that have application scope are not *thread-safe*.

NetBeans after the JSTL 1.2 library was added



The taglib directive for the JSTL core library

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

A JSTL if tag for a validation message

```
<c:if test="${message != null}">
    <p><i>${message}</i></p>
</c:if>
```

A JSTL if tag that tests for a string value

```
<c:if test="${user.wantsUpdates == 'Yes'}">
    <p>This user wants updates!</p>
</c:if>
```

JSTL

- The *JSP Standard Tag Library* (*JSTL*) provides tags for common JSP tasks.
- You must make the jstl-impl.jar and jstl-api.jar files available to the application before you can use JSTL tags.
- To add the JSTL library to a NetBeans project, switch to Projects tab, right-click on Libraries folder, select Add Library, and select the JSTL library.
- You must code a taglib directive that identifies the JSTL library and its prefix before you can use JSTL tags within a JSP.
- You can use the if tag to perform conditional processing that's similar to an if statement in Java.

The five types of JSP tags

Tag	Name	Purpose
`<%@ %>`	JSP directive	To set conditions that apply to the entire JSP.
`<% %>`	JSP scriptlet	To insert a block of Java statements.
`<%= %>`	JSP expression	To display the string value of an expression.
`<%-- --%>`	JSP comment	To tell the JSP engine to ignore code.
`<%! %>`	JSP declaration	To declare instance variables and methods for a JSP.

A directive, scriptlet, and expression

```
<%@ page import="java.util.GregorianCalendar, java.util.Calendar" %>
<%
    GregorianCalendar currentDate = new GregorianCalendar();
    int currentYear = currentDate.get(Calendar.YEAR);
%>
<p>&copy; Copyright <%= currentYear %>
Mike Murach &amp; Associates</p>
```

JSP tags for a validation message

```
<%
    String message = (String) request.getAttribute("message");
    if (message != null) {
%>
    <p><i><%= message %></i></p>
<% } %>
```

JSP tags

- To import classes in a JSP, use the import attribute of the *page directive*.
- To get the values of attributes or parameters that are passed to a JSP, use the getAttribute or getParameter method of the *implicit request object* named request. These methods work the same as methods that are available from the request object that's available to the doGet and doPost methods of a servlet.

An HTML comment in a JSP

```
<!--
<p>This email address was added to our list on <%= new Date() %></p>
-->
```

A JSP comment

```
<%--
<p>This email address was added to our list on <%= new Date() %></p>
--%>
```

Java comments in a JSP scriptlet

```
<%
    // get parameters from the request
    String firstName = request.getParameter("firstName");
    String lastName = request.getParameter("lastName");
    String emailAddress = request.getParameter("emailAddress");

    /*
    User user = new User(firstName, lastName, emailAddress);
    UserDB.insert(user);
    */
%>
```

Comments

- When you code HTML comments, the comments are compiled and executed, but the browser doesn't display them.
- When you code *JSP comments*, the comments aren't compiled or executed.
- When you code Java comments within a scriptlet, the comments aren't compiled or executed.

Code that uses JSP tags to access the User bean

```
<%@ page import="murach.business.User" %>
<%
    User user = (User) request.getAttribute("user");
    if (user == null) {
        user = new User();
    }
%>
<label>Email:</label>
<span><%= user.getEmail() %></span><br>
<label>First Name:</label>
<span><%= user.getFirstName() %></span><br>
<label>Last Name:</label>
<span><%= user.getLastName() %></span><br>
```

The same code using standard JSP tags

```
<jsp:useBean id="user" scope="request" class="murach.business.User"/>
<label>Email:</label>
<span><jsp:getProperty name="user" property="email"/></span><br>
<label>First Name:</label>
<span><jsp:getProperty name="user" property="firstName"/></span><br>
<label>Last Name:</label>
<span><jsp:getProperty name="user" property="lastName"/></span><br>
```

The same code using EL

```
<label>Email:</label>
<span>${user.email}</span><br>
<label>First Name:</label>
<span>${user.firstName}</span><br>
<label>Last Name:</label>
<span>${user.lastName}</span><br>
```

Advantages of standard JSP tags

- Standard JSP tags create a JavaBean if it doesn't already exist.
- Standard JSP tags provide a way to set properties.

Advantages of EL

- EL has a more elegant and compact syntax.
- El allows you to access nested properties.
- EL does a better job of handling null values.
- El provides more functionality.

Notes

- Standard JSP tags make it easier for non-programmers to use beans.
- EL makes it even easier for non-programmers to use beans.
- You typically only need to use standard JSP tags if you're working on legacy applications.

The useBean tag

Syntax

```
<jsp:useBean id="beanName" class="package.Class"
             scope="scopeValue" />
```

Example

```
<jsp:useBean id="user" class="murach.business.User"
             scope="request" />
```

Scope values

Value	Bean is stored in the…
`page`	PageContext object.
`request`	HttpServletRequest object.
`session`	HttpSession object.
`application`	ServletContext object.

The getProperty tag

Syntax

```
<jsp:getProperty name="beanName" property="propertyName" />
```

Example

```
<jsp:getProperty name="user" property="firstName" />
```

The setProperty tag

Syntax

```
<jsp:setProperty name="beanName" property="propertyName"
                 value="value" />
```

Example

```
<jsp:setProperty name="user" property="firstName"
                 value="John" />
```

Standard JSP tags (useBean tags)

- The useBean tag accesses a bean and, if necessary, creates a bean from the JavaBean class.
- *Scope* of a bean refers to the object that stores the bean. This controls how long the bean is available to the rest of the application.
- Because standard JSP tags use XML syntax, these tags are case-sensitive, a front slash indicates the end of the opening tag, and all attributes must be enclosed by single or double quotes.
- Name attribute for the getProperty and setProperty tags must match the ID attribute in the useBean tag.

Escape sequences within attributes

Character	Escape sequence
'	\'
"	\"
\	\\
<%	<\%
%>	%\>

How to use an escape sequence

```
<jsp:setProperty name='user' property='lastName' value='O\'Neil' />
```

How to avoid an escape sequence

```
<jsp:setProperty name="user" property="lastName" value="O'Neil" />
```

A header file named header.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Murach's Java Servlets and JSP</title>
    <link rel="stylesheet" href="styles/main.css" type="text/css"/>
</head>
<body>
```

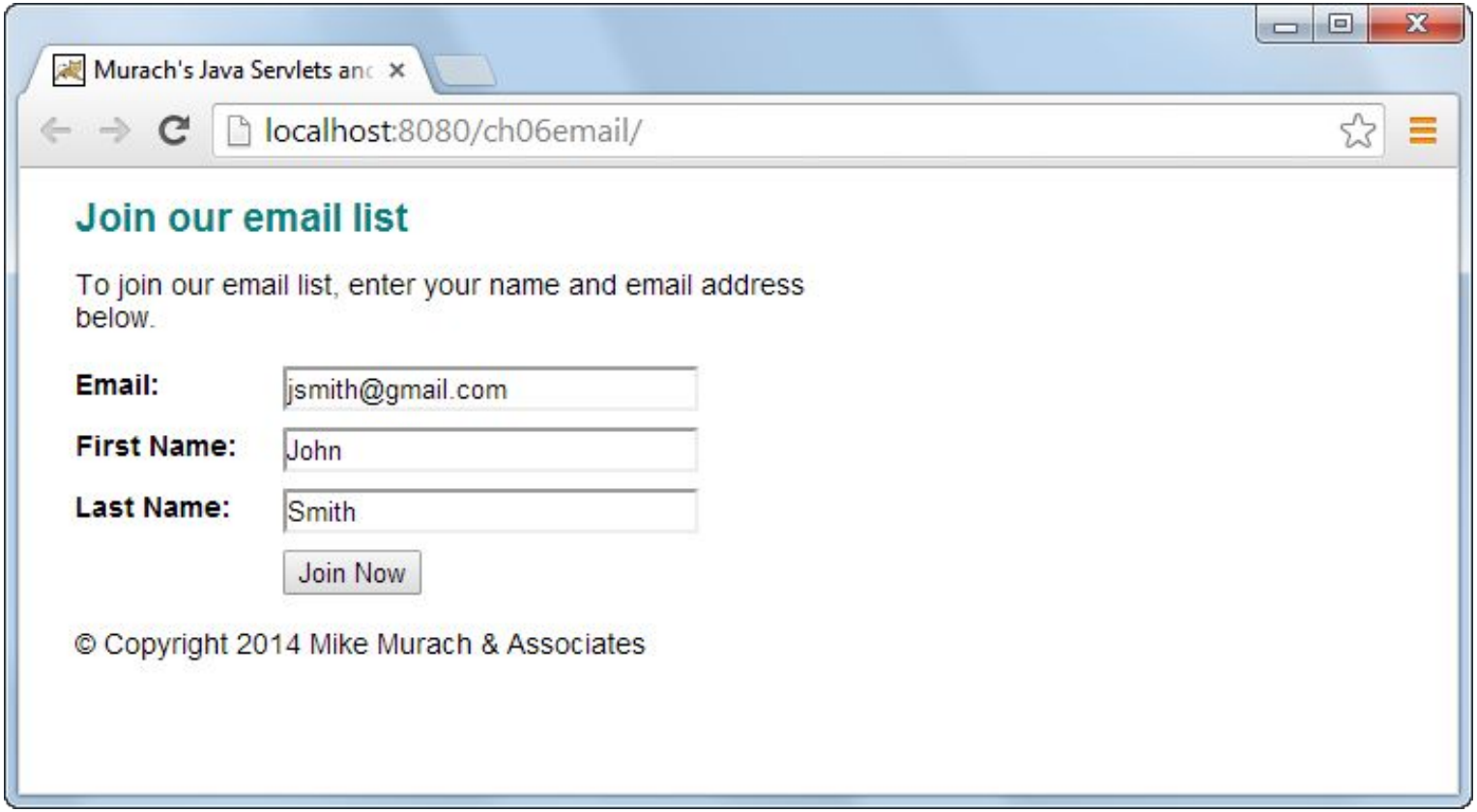
A footer file named footer.jsp

```
<%@ page import="java.util.GregorianCalendar, java.util.Calendar" %>
<%
    GregorianCalendar currentDate = new GregorianCalendar();
    int currentYear = currentDate.get(Calendar.YEAR);
%>
<p>&copy; Copyright <%= currentYear %> Mike Murach &amp; Associates</p>
</body>
</html>
```

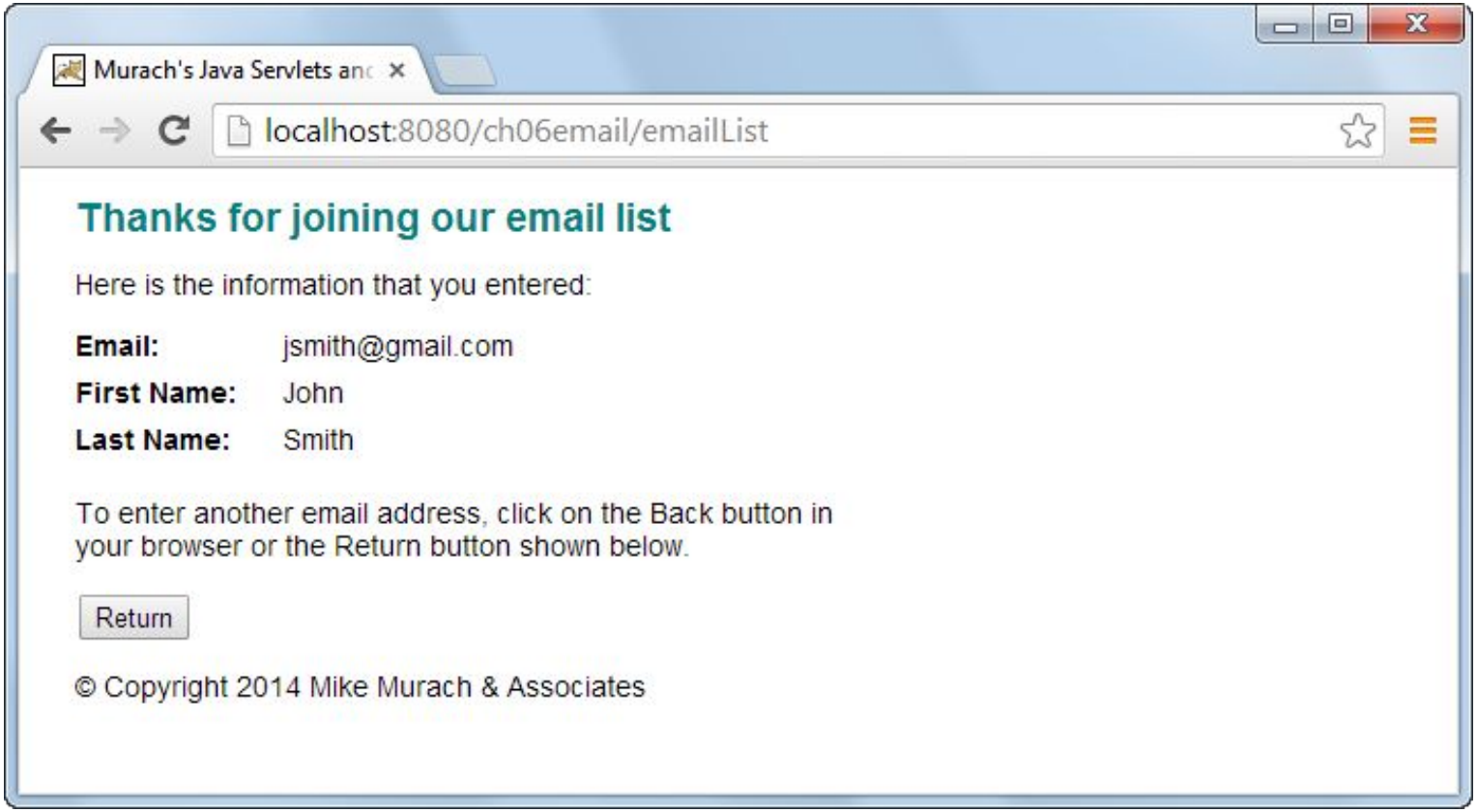
A JSP file that uses both include files

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="/includes/header.html" />

<h1>Join our email list</h1>
<p>To join our email list, enter your name and
   email address below.</p>

<c:if test="${message != null}">
    <p><i>${message}</i></p>
</c:if>
<form action="emailList" method="post">
    <input type="hidden" name="action" value="add">
    <label class="pad_top">Email:</label>
    <input type="email" name="email" value="${user.email}"><br>
    <label class="pad_top">First Name:</label>
    <input type="text" name="firstName" value="${user.firstName}"><br>
    <label class="pad_top">Last Name:</label>
    <input type="text" name="lastName" value="${user.lastName}"><br>
    <label> </label>
    <input type="submit" value="Join Now" class="margin_left">
</form>

<c:import url="/includes/footer.jsp" />
```

The web page displayed in a browser



Another page with the same header and footer



How to include a file at compile-time

Syntax

```
<%@ include file="fileLocationAndName" %>
```

Examples

```
<%@ include file="/includes/header.html" %>
<%@ include file="/includes/footer.jsp" %>
```

How to include a file at runtime

…with the include action

Syntax

```
<jsp:include page="fileLocationAndName" />
```

Examples

```
<jsp:include page="/includes/header.html" />
<jsp:include page="/includes/footer.jsp" />
```

…with the JSTL import tag

Syntax

```
<c:import url="fileLocationAndName" />
```

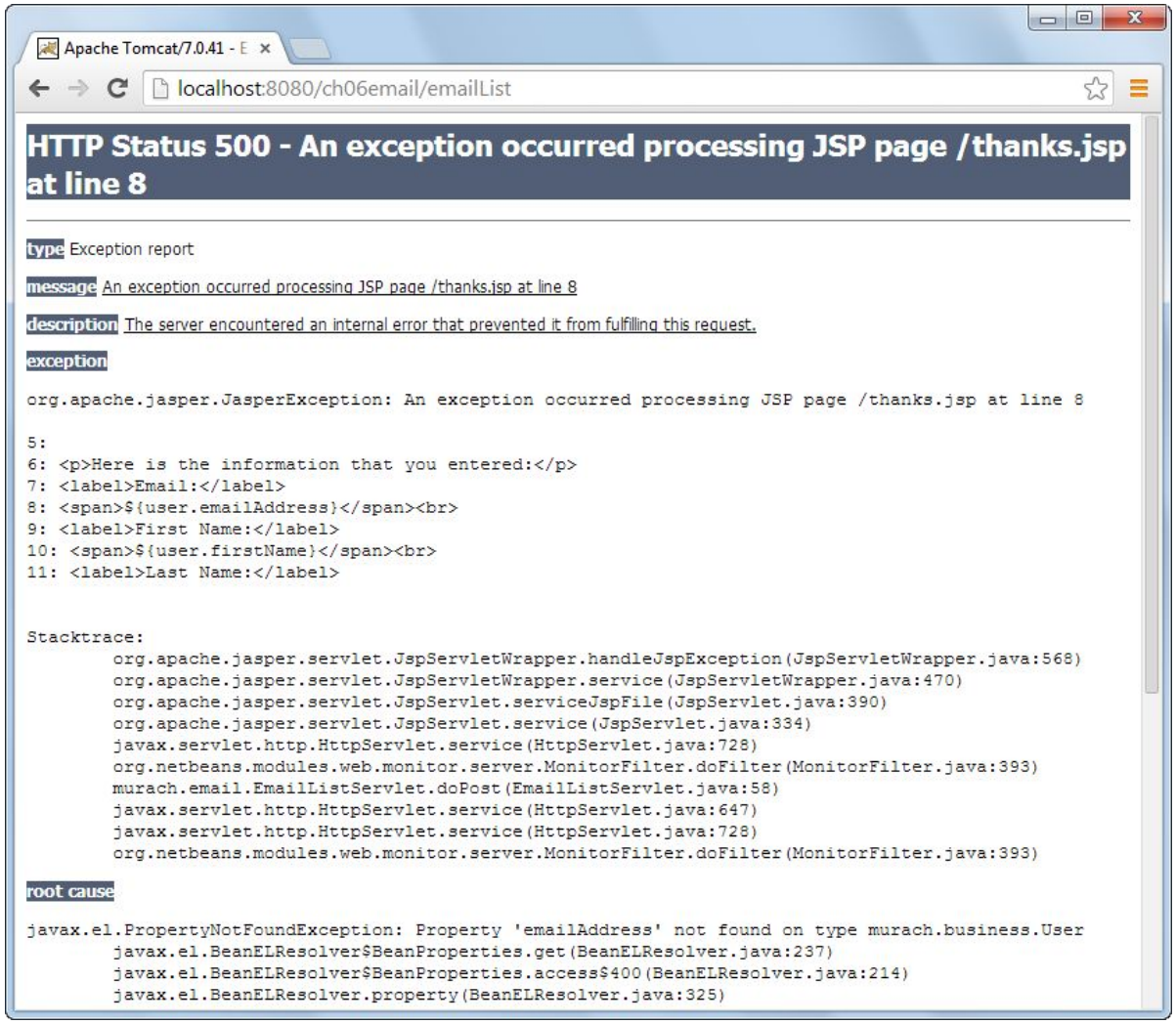
Examples

```
<c:import url="/includes/header.html" />
<c:import url="/includes/footer.jsp" />
<c:import url="http://localhost:8080/murach/includes/footer.jsp" />
<c:import url="www.murach.com/includes/footer.jsp" />
```

File includes

- Use the *include directive* to include a file in a JSP at *compile-time*.
- When you use the include directive, code in the included file becomes part of the generated servlet. As a result, any changes to the included file won't appear in the JSP until the JSP is regenerated and recompiled.
- Use the *include action* or the JSTL import tag to include a file in a JSP at *runtime*.
- When you include a file at runtime, any changes to the included file appear in the JSP the next time it is requested.
- One advantage of the import tag is that it lets you include files from other applications and web servers.

An error page for a common JSP error



Common JSP errors

- HTTP Status 404 – File Not Found Error
- HTTP Status 500 – Internal Server Error

Tips for fixing JSP errors

- Make sure that the URL is valid and that it points to the right location for the requested page.
- Make sure all of the HTML, JSP, and Java class files are in the correct locations.
- Read the error page carefully to get all available information about the error.