Please know Quiz 2 covers:  
Chapter 5 How to develop servlets   
Chapter 6 How to develop JavaServer Pages  
Chapter 7 How to work with sessions and cookies  
Chapter 8 How to use EL  
Chapter 9 How to use JSTL  
Chapter 10 How to use custom JSP tags

**Chapter 5 How to develop servlets**

Knowledge

1. **Describe servlets and servlet mapping, and their use of request and response objects.**

(page 40) A servlet is a special type of Java class that runs on a serverand does the processing for the dynamic web pages of a web application.

Each servlet is a Java class that extends the HttpServlet class.

Servlet mapping: servlets must be mapped to a URL before you can request them via web.xml or annotation (130)

Within the servlet, the doGet and doPost methods both provide an HttpServletRequest object. The object is typically named request (139)

You can use the methods of the request object to get the values of the parameters that are stored in the request

All servlets inherit the GenericServlet class so getServletContext method is available to all servlets

The ServletContext can be used to get real path, to read global initialization parameters, work with variables and write data to logs

Request attributes are reset between requests. If you store an User object a s a request attribute & forward it to a JSP, that User object is only available to JSP & not available later in the session.(ch 7 how to store object so its available to any servlet or JSP in current session)

RequestDispatcher object used to forward a request (144)

Forward request

Redirect response with sendRedirect

1. **Describe how parameters are passed to a servlet with the HTTP GET method.**

**Three ways to append parameters to a GET request** ( 135)

1. Enter the URL into the browsers address bar

<http://localhost:8080/ch05email/email/emailList?action=add&firstName=John>

<http://www.murach.com/email/list?action=add&firstName=John>

1. Code a form that uses the GET method(default)

<form action=”emailList”>

<form action=”emailList” method=”get”>

1. Code an anchor tag

<a href=”emailList?action=join”>Display Email Entry Test</a>

1. **List three reasons for using the HTTP POST method instead of the HTTP GET method.**

Use POST when…

* To post (write ) data to the server
* Executing the request multiple times may cause problems
* You don’t want to include the parameters in the URL for security reasons
* You don’t want users to be able to include parameters when they bookmark a page
* You need to transfer more than 4KB of data

For all other reasons GET is preferred (faster, user can bookmark with parammeters)

Testing with anchor tag or url uses GET (136 error message if no Get method)

Use GET when…

* The request reads data from the server
* The request can be executed multiple times without causing any problems

1. **Describe how the ServletContext object is used to get the path for a file. ()**

getRealPath method of servletContext object

**Describe the use of the methods in a servlet: (161)**

* init: A server loads and initializes the servlet by calling init

(creates first instance , only called once, can override)

public void init() throws ServletException

* service : servlet handles each browser request by calling the service method. This method calls another method to handle the specific HTTP request type

(not recommended to override service method, instead create appropriate doPost or doGet)

public void service(HttpServletRequest request, HttpServlet Response response) throws IOException, ServletException

* doGet:
* doPost
* destroy : server removes servlet by calling destroy. Occurs when the servlet has been idle for some time or when the server is shutdown

public void destroy()

1. **Explain why you should never use instance variables in servlets**. (163)

An instance variable of a servlet belongs to one instance of the servlet and is shared by any threads that request the servlet

Instance variables are not thread-safe. 2 threads may conflict when they try to read, modify and update the same instance variable at the same time (lost updates & problems)

1. **Describe the use of debugging data written to the console or log file**. (166)

Used messages to help track the methods that are executed or to view the value of a variable

Use println methd of System.out and System.err

When checking value of variables, include name of class and variable (easier to understand and easier to remove debugging code)

Key debugging data that you want to keep can be written to a log file (168)

Notes:

1. How to create & map a servlet (page 128)

There are 2 steps:

* Code the class for the servlet
* Map that class to a URL (via web.xml or @WebServlet annotation)

**Basic Structure for a web servlet that handles both HTTP GET and POST requests:**

1st statement specifies the package for the servlets class . This must correspond to the directory that contains the servlet.

package murach.email;

2nd import statements for classes required by servlet. Most are required by all servlets. The Printwriter class is only necessary for a servlet that needs to send text like HTML or XML to a client

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

3rd Class declaration provides the name for the servlet’s class and indicates that it extends the HttpServlet class. In theory, they can extend the GenericServlet class. In practice, all servlets extend the HttpServlet class.

public class TestServlet extends HttpServlet{

4th the doGet and doPost methods in this figure accept the same arguments and throw the same exceptions. Within these methods , you can use the methods of the request object to get incoming data and you can use the methods of the response object to set outgoing data.

In this example the doGet method calls the getWriter method of the response object to get a PrintWriter named object out. Once you get this object, you can use one println statement or a series of print and println statements to return HTML or other text to the browser. Since it can throw an IOexception, it is in a try block

The last statement closes and flushes the output stream and releases any resources that are being used by the PrintWriter object. Since you always want to execute this statement it is in a finally block.

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Servletexception, IOException

{

response.setContentType(“text/html”);

PrintWriter out = response.getWriter();

try{

out.println(“<h1>HTML from servlet</h1>”);

}

finally{

out.close();

}

}

@Override

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

doPost(request, response);

}

}

**How to map a servlet with the web.xml file:**

Advantage: works with all versions of Tomcat

3.0 Servlet specification in Tomcat 7.0 or later : easier to use annotations

Web.xml file contains XML elements that map two servlets

1st servlet element in example declares the name that refers to the EmailListServlet class

The servlet-name element provides an unique name for the class. This name is used internally by the web.xml file. Then the servlet-class element uses a fully-qualified name to identify the class the class for the servlet. In the example, the servlet-name entry is the same as the name for the class, but isn’t qualified by the package name. (it is a common naming convention). If the same servlet name is used in 2+ packages, you can use servlet elements to specify a unique name for each servlet.

The second servlet element uses the same naming convention as the first element. It declares an internal name of TestServlet for a servlet named TestServlet.

The first servlet-mapping element maps the EmailListServlet to a single URL that’s available from the root directory in the application. This enables a user to request this servlet by specifying a URL:

<http://localhost:8080/ch05email/emailList>

Note that the UrL removes the word Servlet from the end of the servlet name ( hides the fact that app uses servlets and shortens URL)

The second servlet-mapping element uses a wildcard character(\*) to map the EmailListServlet to any URL that resides within the email directory. This allows theis servlet to be requested by multiple URLs.

<http://localhost:8080/ch05email/email/add>

or

<http://localhost:8080/ch05email/email/addToList>

Note: this servlet mapping works even if the email directory is a virtual directory and doesn’t exist on the server

The third servlet mapping element maps theTest Servlet to a single URL

XML tags that add servlet mapping to the web.xml file.

<!--the definition for services -->

<servlet>

<servlet-name>EmailListServlet</servlet-name>

<servlet-class>murach.email.EmailListServlet</servlet-class>

</servlet>

<servlet>

<servlet-name>TestServlet</servlet-name>

<servlet-class>murach.email.TestServlet</servlet-class>

<servlet>

<!--the mapping for the services -->

<servlet-mapping>

<servlet-name>EmailListServlet</servlet-name>

<url-pattern>/emailList</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>EmailListServlet</sevlet-name>

<url-pattern>/email/\*</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>TestServlet</servlet-name>

<url-pattern>/test</url-pattern>

</servlet-mapping>

======================page 131===================

<servlet-class> specifies the class. Includes package and name for class but not the .class extension

<servlet-name> specifies a unique name for the servlet to identify it within the web.xml file. Required by both the servlet element and servlet mapping element … maps servlet-mapping element to a servlet element

<url-pattern> specifies the URL or URLs that are mapped to the the specified servlet. Pattern must begin with a front slash but the URL pattern ca specify a virtual directory

/emailList Specifies the emailList URL in the root dir of app

/email/\* Specifies any URL in the email dir

/email/add Specifies the add Url in the email dir

**How to map a servlet with an annotation:**

Less code & easier

1. Add an import statement for the WebServlet class

Import javax.servlet.annotation.WebServlet;

1. Add the @webServlet(“/dir”) jist before the class declaration

* Adding the url in () maps to a single dir
* Multiple dir: @WebServlet(urlPatterns={“/emailList”,”/email/\*”})
* Specify name for servlet: @WebServlet(name=”MurachTestServlet”, urlPatterns={“/test”})
* By default the internal name that’s used for the servlet is the same as the class name of the servlet… if it leads to a conflict you can use the @WebServlet to specify a unique internal name: example: TestServlet in 2 different packages
* Usually use 1 way, if use both web.xml overrides annotation

**How to Request a Servlet :**

After creating & mapping, request the servlet to test it (HTTP GET or POSt request)

**HTTP GET method (2 ways)**

1. Code a form tag or an anchor(a) tag that requests the servlet
2. If web server & servlet engine ar running, enter the URL directly into browseer

To test the servlet, you often need to pass parameters to it by appending the parameters to the end of the url

The question mark after the servlets url indicates that one or more parameters will follow

Then the parameter name = value for each parameter passed separated with &

Append parameters to a URL:

emailList?action=add

emailList?firstName=John&lastName=Smith

**Three ways to append parameters to a GET request**

1. Enter the URL into the browsers address bar

<http://localhost:8080/ch05email/email/emailList?action=add&firstName=John>

<http://www.murach.com/email/list?action=add&firstName=John>

1. Code a form that uses the GET method(default)

<form action=”emailList”>

<form action=”emailList” method=”get”>

1. Code an anchor tag

<a href=”emailList?action=join”>Display Email Entry Test</a>

**How to use the HTTP POST method:**

In the form method, use the method attribute to specify POST:

<form action=”emailList” method=”post”>

The browser doesn’t display the parameters for the request in its URL

**How to work with servlets: get the values of the parameters (138)**

Two methods available from the HttpServletRequest object:

1. getParameter(String param)

Returns the value of the specified parameter as a string if it exists or null if it doesn’t. Often this is the value defined in the value attribute of the control in the Html or JSP file

1. getParameterValues(String param)

Returns an array of String objects containing all of the values that the given request parameter has or null if the parameter doesn’t have any values

Servlet code that gets text from a text box

String firstName = request.getParameter(“firstName”);

Servlet code that determines if a checkbox is checked:

//returns the value or “on” if checked,, null otherwise

String rockCheckBox = request.getParameter(“rock”);

If (rockCheckBox != null)

{

//rock music was checked

}

Servlet code that reads and processes multiple values from a list box

//returns the values of the items selected in a list box

String[] selectedCountries = request.getParameterValues(“country”);

For (String country : selectedcountries)

{

//code to process each country

}

**How to get the real path for a file: (140)**

You typically use relative paths

To get the real path:

1. call getServletContext() method to get a ServletContent object that contains info about the apps context
2. Call the getRealPath(String path) method of the ServletContext object to return the real path for the specified file ( a front slash navigates to the root dir of the current app:

getRealPath(“/EmailList.txt”) )

returns a String object for the absolute path of the specified relative path

Code that gets the absolute path for a file:

ServletContext sc = this.getServletContext();

String path = sc.getRealPath(“/WEB-INF/EmailList.txt”);

More concise way to get abs path:

String path = this.getServletContext().getRealPath(“/WEB-INF/emailList.txt”);

A possible value for the real path variable:

C:\murach\servlet\_and\_jsp\netbeans\book\_apps\ch05email\build\web\WEB\_INF\EmailList.txt

**How to get and set request attributes (143)**

To store any object in the request object, you can use the setAttribute method.

To create a User object named user and stores it in the request object with a name of “user”

**Set Request Attribute:**

User user = new User(firstName, lastName, email);

Request.setAttribute(“user”, user);

**Get a request attribute:**

User user – (User) request.getAttribute(“user”);

**Set a request attribute for a primitive type**

int id =1;

Request.setAttribute(“id”, new Integer(id));

**Get request attribute of a primitive type**

int id = (Integer) request.getAttribute(“id”);

**How to forward requests from servlet to html page (144)**

1. Call get ServletContext from HttpServlet class returns ServletContext object
2. Call getRequestDispatcher method of ServletCOntext object returns a RequestDispatcher object
3. In this method, code a URL that starts with a slaxh so it is relative to root of app
4. Use the forward method to forward the request and response objects to the HTML page, JSP or servlet specified by the JSP
5. The first statement below sets the string for URL,
6. Second statement uses method chaining to:
7. gets ServletCOntext object
8. Gets request dispatcher object for url
9. Forwards the request

**Forward the request to an HTML page:**

String url = “/index.html”;

getServletContext().getRequestDispatcher(url).forward(request, response);

**Forward request to a JSP**

String url = “thanks.jsp”;

getServletContext().getRequestDispatcher(url).forward(request, response);

**Forward to servlet**

String url =”/cart/displayInvoice”;

getServletContext().getRequestDispatcher(url).forward(request, response);

**How to redirect responses** (145)

1. Use the sendRedirect method of the response object
2. You use this technique when you want to transfer control ti a URL outside of your app
3. You often supply an absolute URL, but can supply relative url & servlet can convert to abs
4. If you begin with a slash, the servlet path relative to root dir of servlet
5. When you call sendRedirect, the server sends an absolute url to browser
6. Then browser sends a request for url
7. Processing occurs on client side, not as efficient as forwarding.
8. Also sendRedirect doesn’t transfer the request and response objects
9. You should only use sendRedirect when you want to redirect to an URL that’s available from another web app

Examples:

response.sendRedirect(“email”);

response.sendRedirect(“/musicStore/email”);

response.sendRedirect(<http://www.murach.com/email/>);

**How to validate data:**

Usually validate on client side browser with Javascript

1. More responsive to user
2. Less workload on server

**To Validate data on the server:**

Data is also always validated on the server.

1. JavaScript may not be able to do all the required validation (email address already in db)
2. JavaScript may be disabled in clients browser

Index.jsp on page 147

doPost on page 149 Validates data on server

**web.xml (150)**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Chapter 6\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Knowledge

* **List the three rules for defining 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*).

* **List the four scopes that EL searches in the sequence used by EL (177)**

**Scope Description**

page The bean is stored in the implicit PageContext object (smallest)

request The bean is stored in the HttpServletRequest object

session The bean is stores in the HttpSession object

application The bean is Stored in the ServletContext object (largest)

* **List and describe one type of JSTL tag.**

JSTL if tag for validation

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>

* **List the five types of old JSP tags and**

**Five Types of JSP tags**

<%@ %> 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

* **describe why JSP tags aren’t typically used for new development.**

EL makes it easier for non-programmers to use beans

JSP tags are on legacy applications

* **Distinguish between EL and standard JSP tags.**

**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.
* **Describe the use of include files.**

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

.

**EL (expression language) & JSTL (JSP Standard tag library)**

Both were introduced in JSP 2.0 specification

Tags have several advantages over older JSP tags.You often use EL with JavaBeans

Example of a JavaBean: User class

Implements Serializable (some servlet engines cansave and restore this object if that’s necessary.)

**EL (Expression Language)**

* 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

**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.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Chapter 6\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Knowledge

* **Describe the way HTTP works without session tracking.**
  + HTTP is a *stateless protocol*. Once a browser makes a request, it drops the connection to the server. To maintain *state*, a web application must use *session tracking*.
* **Describe the way cookies are used for session tracking.**
  + By default, the servlet API uses a cookie to store a session ID in each browser. The browser passes the cookie to the server with each request.
  + To store the data for each session, the server creates a *session object*.
  + To provide session tracking when cookies are disabled in the browser, you can use *URL encoding* to store the session ID in the URL for each page of an application. However, this is not considered a best practice.
  + *Persistent cookies* are stored on the user’s PC. *Per-session cookies* are deleted when the session ends.
* **Describe why it’s generally considered a best practice to use cookies for session tracking instead of using URL encoding.**

**Limitations:**

* Most browsers limit the number of characters that can be passed by a URL to 2,000 characters.
* It’s difficult to include spaces and special characters such as the ? and & characters in parameter values.

Two security risks with URL rewriting

* Parameter values can leak to third-party sites such as Google Analytics or Facebook.
* Parameter values are stored in the browser history.
* **Distinguish between persistent cookies and per-session cookies.**
  + A per-session cookie that holds the session ID is automatically created for each session. It is used to relate the browser to the session object.
  + You can also create and send other cookies to a user’s browser, and use them to access user-specific data that’s stored in a file or database.

**Session**

* + A session object is created when a browser makes the first request and destroyed when the session ends.
  + A session ends when a specified amount of time elapses without another request or when the user exits the browser.

**To provide thread-safe access to the session object:**

* + Each servlet creates one session object that exists for multiple requests that come from a single client.
  + If the client has one browser window open, access to the session object is thread-safe.
  + If the client has multiple browser windows open, two threads from the same client could access the session object at the same time. As a result, the session object isn’t thread-safe.
  + Since the servlet specification doesn’t guarantee that it will always return the same session object, you can’t make the session object thread-safe by synchronizing on it. Instead, you can synchronize on the session ID string for the session object.

How cookies work:

* + A cookie is a name/value pair stored in a browser.
  + A web application on the server creates a cookie and sends it to the browser. The browser on the client saves the cookie and sends it back to the server every time it accesses a page from that server.
  + Cookies can be set to persist for up to 3 years.
  + Some users disable cookies in their browsers.
  + Browsers generally accept only 20 cookies from each site and 300 cookies total. They can limit each cookie to 4 kilobytes.
  + A cookie can be associated with one or more subdomain names.

**4 Methods of the cookie class:**

**setPath(String path)** To make the cookie available to the entire app, you can set the path to /

**setDomain(String pattern)** By default, the browser only returns a cookie to the host that sent the cookie. To return a cookie to othe hosts within the same domain, set a domain [attern like ads.com. Then, the browser returns the cookie to any subdomain of [www.ads.com](http://www.ads.com)

**setSecure(Boolean flag)** By default,the browser sends a cookie over a regular, or encrypted, connection. To force a secure connection, supply a true value for this method

**setVersion(int version)** By default, Java creates cookies that use version0 of the cookie protocol. However you can secify an int value of 1 for this method to use the new version of the protocol which is 1

* **Distinguish between the use of URL rewriting and the use of hidden fields as ways to pass parameters.**
  + You can use *URL rewriting* to pass parameters to a servlet or JSP. To do that, you add the parameters to the end of the URL.
  + You can use *hidden fields* to pass parameters to a servlet or JSP. To do that, you code hidden fields within a form tag.

**Example of form:**

<form action="cart" method="post">

<input type="hidden" name="productCode"

value="${product.code}">

<input type=text size=2 name="quantity"

value="${lineItem.quantity}">

<input type="submit" name="updateButton"

value="Update">

</form>

One limitation of hidden fields

* Because hidden fields are displayed in the source code for the page returned to the browser, anyone can view the parameters. As a result, hidden fields aren’t appropriate for sensitive data like passwords.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Chapter 8\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Knowledge

1. Describe the advantages and disadvantages of using EL.

**Advantages**

* EL has a more elegant and compact syntax than standard JSP tags.
* EL lets you access nested properties.
* EL lets you access collections such as maps, arrays, and lists.
* EL does a better job of handling null values.
* EL provides more functionality.

Disadvantages of EL

* EL doesn’t create a JavaBean if it doesn’t already exist.
* EL doesn’t provide a way to set properties.

1. **Explain why you might want to disable EL or scripting.**

* For JSP 2.0 and later, the servlet container evaluates any text that starts with ${ and ends with } as an EL expression. Most of the time, this is what you want. If it isn’t, you can ignore EL for a single JSP or for all JSPs in the entire application.
* If you want to remove all scripting from all JSPs in your application, you can modify the web.xml file so it doesn’t allow scripting. Then, if you request a JSP that contains scripting, the servlet container displays an error page.

The dot operator with JavaBeans and maps

* A JavaBean is a special type of object that provides a standard way to access its *properties*.
* A *map* is a special type of collection that’s used to store key/value pairs. A HashMap collection is a map.
* When you use the dot operator, code to the left of the operator must specify a JavaBean or a map. Code to the right of the operator must specify a JavaBean property or a map key.
* 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).

Implicit EL objects for scope

* If you have a naming conflict, use the *implicit EL objects* to specify scope.
* Because all of the implicit EL objects for specifying scope are maps, you can use the dot operator to specify a key when you want to return the object for that key.

\*\*\*\*\*\*\*\*\*\*\*Chapter 9\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Knowledge

Describe the use of these JSTL tags:

out,

forEach,

forToken,

if,

choose,

import,

url.

JSTL

* The *JSP Standard Tag Library* (*JSTL*) provides tags for common JSP tasks.
* Before you can use JSTL tags within an application, you must make the jstl-impl.jar and jstl-api.jar files available to the application. To do that with NetBeans, you can add the JSTL 1.2 class library to your project.
* Before you can use JSTL tags within a JSP, you must code a taglib directive that identifies the JSTL library and its prefix.

XSS attacks

* In a *cross-site scripting* (*XSS*) *attack*, an attacker attempts to inject Javascript into your page to trick your users into sending them personal data.
* To prevent an XSS attack, use the out tag to escape the output for your application. This tag replaces characters such as the left angle bracket (<) and right angle bracket (>) with their corresponding HTML entities.
* It’s generally considered a best practice to escape all output that’s sent to the browser, including other parts of the HTTP request such as cookies and other headers.

If and choose tags

* Use the if tag to perform conditional processing that’s similar to an if statement in Java.
* Use the choose tag to perform conditional processing similar to an if/else statement in Java. To do that, code multiple when tags and a single otherwise tag within the choose tag.
* If necessary, nest one if or choose tag within another.

url tag

* You can use the JSTL url tag to encode URLs within your web application that are relative to the application’s root directory. As a result, it can prevent your code from breaking if the application context changes.
* By default, the url tag automatically rewrites the URL to include the session ID whenever necessary. This can lead to session hijacking. To prevent this, add a tracking-mode element to the web.xml file to specify the application should only use cookies (not URL encoding) for tracking.
* Use the JSTL param tag if you want to automatically encode unsafe characters such as spaces with special characters such as plus signs.

\*\*\*\*\*\*\*\*\*\*\*\*\*Chapter 10\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Knowledge

1. Explain how a custom JSP tag gets associated with a tag handler class.

Custom Tags

* The *Tag Library Descriptor* (*TLD*) is an XML document that describes a *tag library* with custom tags.
* An application typically uses a single TLD to define all of its custom tags.
* There’s no limit to the number of TLDs an application can have.
* Within a tag element, you must use the name element to specify the name of the custom tag.
* Within a tag element, you must use the tagclass element to specify the *tag class* for the tag.
* Within a tag element, you can optionally use the info element to specify descriptive information about the tag.
* The elements that are required by a TLD may vary depending on the JSP engine.

A custom tag that doesn’t have a body

* The *tag class*, or *tag handler class*, is the Java class that defines the actions of the tag.
* A tag class must implement the Tag interface.
* For a tag that doesn’t have a body, implement the Tag interface by extending the TagSupport class and overriding the doStartTag method.
* To display text on the JSP, use the print method of the JspWriter object.
* To get a JspWriter object, use the getOut method of the pageContext object that’s defined in the TagSupport class.

A custom tag with a body

* A tag that has a body must have an opening tag, a body, and a closing tag.
* The body of the tag can contain any HTML or JSP elements.
* The tag class for a custom tag can control whether the body of the tag is displayed in the JSP.
* When you add a tag that has a body to a TLD, you must specify a value of “JSP” for the bodycontent element.
* To create a tag class for a tag that has a body, you extend the TagSupport class and override the doStartTag method.
* To display the body of the tag in the JSP, the tag class should return the EVAL\_BODY\_INCLUDE constant. Otherwise, the tag class should return the SKIP\_BODY constant.

A custom tag that has attributes

* In the TLD file, the tag element can include the definitions for one or more attributes.
* In the TLD file, each attribute should include at least the name and required elements.
* In the tag class, declare a private instance variable for each attribute.
* In the tag class, define a set method for each attribute with the standard naming conventions.

A custom tag that reiterates its body

* To pass data to the tag class, store that data as a session attribute.
* To access a tag that has a body, the tag class must implement the BodyTag interface. The easiest way to do this is to extend the BodyTagSupport class.
* If the doStartTag method returns the EVAL\_BODY\_BUFFERED constant, the body of the tag is evaluated by calling the doInitBody method and the doAfterBody method.
* The doInitBody method sets the initial values for the first row of the body.
* If the doAfterBody method returns the EVAL\_BODY\_AGAIN constant, the doAfterBody method is called again.
* You can use the setAttribute method of the PageContext object to set any attributes that you need to access from the JSP tag.
* You can use the getEnclosingWriter and writeOut methods of the bodyContent object to write the body to the JSP.

\*\*\*\*\*\*\*\*\*\*\*Chapter 5\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Chapter 5, page 128

In the second to last paragraph, the first line should say “doPost”, not “doGet”.

Chapter 5, page 130 In the second paragraph, the third line should say “that’s stored in the murach.email class”, not “that’s shown in the previous figure”.

Chapter 5, page 152 On the second to last line, it should say “@WebInitParam”, not “@InitParam”.

Chapter 5, page 153 In the second code example, it should say “@WebInitParam”, not “@InitParam”.

Chapter 5, page 161 All five common methods of a servlet should begin with “protected”, not “public”.

Chapter 5, page 170 In the sixth summary bullet, it should say “getParameterValues”, not “getParameters”.

\*\*\*\*\*\*\*\*\*\*\*\*Chapter 6\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

175 : Java Bean is a special type of object that provides standard way to access its properties POJO

Code for User bean class

177: JSP Expression Language (EL) makes it easy to access attributes and Java Bean Properties from a request object

When you use the dot operator with a Java Bean, the code to the left of the operator specifies the Java Bean and the code to the right of the operator specifies the property of the java bean

When you use this syntax, EL looks up the attribute starting with the smallest scope(page scope) and moves towards largest scope (application) (page,request,session,application)

Attributes with Application scope are not thread safe

How to display an attribute: ${attribute}

Servlet code Gregorian calendar

GregorianCalendar currentDate = new GregorianCalendar();

Int currentYear = currentDat.get(calendar(YEAR);

Request.setAttribute(“currentYear”, currentYear);

JSP code: <p>The current year is ${currentYear}</p>

How to display the property of an attribute : ${attribute.property}

Servlet code

User user = new User(firstName, lastName, email);

Request.setAttribute(“user”, user);

JSP code: <p>Hello ${user.firstName}</p>

179: The JSP Standard Tag library (JSTL) provides tags for common JSP tasks

Before you can use JSTL tags within an app, you must make the jstl-impl.jar and jstl-api.jar files available to the application

To add the JSTL library to a NetBeans project, switch to Projects, right click on Libraries, select add libraries, select JSTL Library

Before you can use JSTL tags within a JSP, you must code a taglib directive that identifies the JSTL library and its prefix

You can use the if tag to perform conditional processing that’s similar to an if statement in Java. Within the if tag, you can use the test attribute to specify the Boolean condition for the if statement

JSTL provides many other tags .

Taglib directive that specifies the JSTL core library

JSTL if tag that displays or hides a validation message

JSTL if tag that tests for a string value

181: To import classes in a JSP, you use the import attribute of the page directive

To get the values of attributes or parameters that are passed to a JSP, you can use the getAttribute or getParameter method of the implicit request object named request. These methods work the same as the methods that are available from the request object that’s available to the doGet and doPost methods of a servlet

5 types of JSP Tags: JSP directive, scriptlet, expression, comment, declaration

Code that uses a directive, scrptlet and expression

JSP tags that hide or display a validation message

183: HTML comment in a JSP <!--

JSP comment <%--

Java comments in a JSP scriptlet: <% // /\* \*/ %>

HTML: When you code HTML comments, the comments are compiled and executed, but the browser doesn’t display them

JSP comments aren’t compiled or executed

Java comments within a scriptlet aren’t compiled or executed

185: Code that uses JSP tags to access the User bean

Same code using standard JSP tags

Same code using el

Advantages of standard JSP tags (creates Javabean if dne, provide a way to set properties)

Advantages of EL (elegant & compact syntax, allows access to nested properties. Better with null values, more functionality)

Standard JSP tags make it easier for non-programmers to use beans because they look more like HTML tags and don’t require Java code

EL makes it easier for non-programmers to use beans because they are shorter and easier to code than standard JSP tags

187: The useBean tag

Scope Values: page request session application

The useBean tag is used to access a bean and, if necessary, create a bean from the java bean class

The 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

The JSP tags for working with beans use XML syntax. As a result, these tags are case-sensitive, a front slash indicates the end of the opening tag or the start of a closing tag and all attributes must be enclosed by single or double quotes

189: The getProperty tag

The setProperty tag

Escape sequences within attributes

How to use an escape sequence

How to avoid an escape sequence

The name attribute for the getProperty and setProperty tags must match the Id attribute in the useBean tag

To code special characters within an attribute, use escape sequences. However, if you enclose an attribute in double quotes, you don’t need to use an escape sequence for single quotes. If you enclose in single quotes you don’t need escape sequences for double quotes.

The getProperty tag can’t be used to get a null value or an empty string, and the setProperty tag can’t be used to set a property to a null value or an empty string. However, you can use the constructor of a bean or the code in a servlet to set a property to a null value or an empty string

191: header file named header.html

Footer file named footer.jsp

A JSP file that uses both include files

You can use a JSTL import tag to include HTML or JSP files within a JSP

In this example, the header and footer files are both stored in a directory named includes

193: web page displayed in browser with header & footer

195: How to include a file at compile time with an include directive

How to include a file at runtime with include action

How to include a file at runtime with JSTL import tag

To include a file in a JSP at compile time, use the include directive

When you use the include directive, the 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. However, some of the newer web servers automatically detect changes to include files and automatically regenerate and recompile the servlets for the JSPs that need to be updates

To include a file in a JSP at rntime, you use the include action or the JSTL import tag

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 included files from other applications and web servers

197: How to fix common JSP errors

HTTP Status 404 – File not found error

HTTP Status 500 – Internal Server Error

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 info about the error

Chapter 6, page 180 In the second to last paragraph, the fourth line should say “first example”, not “second example”.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Chapter 7: Sessions & Cookies\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

203: why session tracking is difficult with HTTP

How Java keeps tracks of sessions

HTTP is a stateless protocol. Once a browser makes a request , it dorps the connection to the server. So to maintain state, a web application must use session tracking

By default, the servlet API uses a cookie to store a session ID in each browser. Then, the browser passes the cookie to the server with each request

To store the data for each session, the server creates a session object

To provide session tracking when cookies are disabled in the browser, you can use URL encoding to store the session ID in the URl for each page of an application. However, there are several problems with this technique, and it is not considered a best practice

There are two types of cookies: persistent cookies are stored on the user’s PC and per-session cookies deleted when the session ends

205: Index page & cart page

Without session tracking, the cart would not be able to have more than one item

207: A method of the request object: getSession()

3 methods of the session object: setAttribute, getAttribute, removeAttribute

Examples

Code that gets a session object

Code that sets a String object as an attribute

Code that sets a user-defined object as an attribute

Code that gets a String object

Code that gets a user-difined object

Code that removes an object

A session object is created when a browser makes the first request to a site. It is destroyed when the session ends

A session ends when a specified amount of time elapses wthout another request or when the user exits the browser

209: More methods of the session object: getAttributeNames(), getId(0, isNew(), setMaxInactiveInterval(int seconds), invalidate()

Examples

A method that gets all the names of the attributes for a session

A method that gets the ID for a session

A method that sets the inactive interval for a session

A method that sets the inactive interval for a session

A method that invalidates the session and unbinds any objects

If the session object has been explicitely or implicitly invalidates, all methods of the session object throw an IllegalState Exception

For more information about these and other methods of the session object, you can look up the HttpSession interface in the javax.servlet.httpp package in the cdocumentation for the Servlet and JavaServer Pages API

211: An example that synchronizes access to the session object

Another example that synchronizes access to the session object

A web browser with 3 tabs accessing same sessin object

Each servlet creates one session object that exists for multiple requests that come from a single client

If the client as one browser window open access to the session object is thread safe

If the client has multiple browser windows open, its possible (thought unlikely) that two threads form the same client could access the session object at the same time. As a result, the session object isn’t thread safe.

Since the servlet specification doesn’t guarantee that it will always return the same session object, you can’t make the session object thread safe by synchronizing it. Instead you can synchronize the session id string for the session object

213: **How cookies work**

* + A cookie is a name/value pair that is stored in a browser
  + On the server, a web application creates a cookie and sends it to the browser. On the client, the browser saves the cookie and sends it back to the server every time it accesses a page from that server
  + Cookies can be set to persist within the users browser up to 3 years
  + Some users disable cookies in their browsers. As a result, you can’t count on all cookies enabled
  + Browsers generally accept only 20 cookies form each site and 300 cookies total. They can limit to each cookie 4kilobytes
  + A cookie can be associate with one or more subdomain names

**Typical uses for a cookie**

* + To allow users to skip logi and registration forms that gather data like user name, password, address, or credit card data
  + To customize pages that display information like weather reports, sports scores, and stock quotations
  + To focus advertising like banner ads that target the users interests

A per session cookie that holds the session ID is automatically created for each session. That cookie is used to relate the browser to the session object

You can also create and send other cookies to a user’s browser. You can use these cookies to access user-specific data that’s stored in a file or database

215: Constructor of the cookie class

Methods of the cookie class: setMaxAge, setPath, getName, getValue

Method of a response object: addCookie

Method of the request object: getCookies(0

Code that creates a cookie and sets it in a response

Code that gets a cookie value from a request

217: A JSP that shows all cookies for the current server

JSP code that displays the names and values for all cookies

Servlet code that deletes all persistent cookies

JSP code that displays the value for a cookie named emailCookie

To display all the cookies for a user’s browser, use JSTL to loop through all cookies stored in the built-in map named cookie. Then code the key property to get the name of each cookie and code the value property to get the value of each cookie

To delete a persistent cookie from a browser, set the age of the cookie to 0

219: 4 methods of the cookie class: setPath, setdomain, setSecure, setVersion

All of these set methods have corresponding get methods

For more information about these methods: javax.servlet.http package doc for Servlet & JavaServer Pages APi

221: A utility class that gets the value of a cookie

Code that uses the CookieUtil class to get the value of a cookie

To make it easier to get the value of a cookie, you can create a utility class that contains a method that accepts an array of Cookie objects and the name of the cookie and then returns the value of the cookie

223: How to work with URL rewriting and hidden files

How to use URL rewriting to pass parameters

The syntax for URL rewriting

A link that adds a product code to a URL

The link displayed in a browser

The URL that’s displayed when you click on the link

A form tag that calls a JSP

A link that uses EL for the product code value

A link that includes two parameters

Two limitations of URL rewriting

1. Most browsers limit the number of characters that can be passed by an URL to 2000 characters
2. Its difficult to include spaces and special characters such as the ? and & characters in parameter values

Two security risks with URL rewriting

1. Parameter values can leak to third-party sites such as Google & FaceBook
2. Parameter values are stored in the browser history

You can use URL rewriting to pass parameters to a servlet or JSP. To do that, you add the parameters to the end of the URL

225: A form tag that uses a hidden text field and a button

A form tag that uses EL to set hidden field values

One limitation of hidden fields : Because hidden fields are displayed in the source code for the page that’s returned to the browser, anyone can view the parameters. As a result hidden fields aren’t appropriate for secure data like passwords

You can use hidden fields to pass parameters to a servlet or JSP by coding hidden fields within a form tag

227: Download application & user interface

229: file structure & web.xml file

231: code for JSPs

233: code for the 8601\_download.jsp

This is one of the four JSPs for downloading songs

When the web browser receives the URL for a sound file, it downloads and plays it

This JSP gets the product code from the session object and uses it in the URLs for sound files. This isn’t necessary, though because the URLs could be hard-coded

Another way to handle the downloads is to write one JSP that works for all the albums. To implement that, you can store the data for the downloadable songs in one file for each album. Then the download JSP can get the product code from the session object, read the related file, and load its data into the table

235: Code for the Download servlet class

237

239

\*\*\*\*\*\*\*\*\*\*\*\*Chapter 8: EL \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

245: Code in a JSP that accesses a User object that’s stored in the session object

EL

Standard JSP tags

Advantages of EL

1. EL has more elegant and compact syntac than standard JSP tags
2. EL lets you access nested properties
3. EL lets you access collections such as maps arrays and lists
4. EL does a better job of handling null values
5. EL provides more functionality

Disadvantages of EL

1. EL doesn’t create a Java Bean if it doesn’t already exist
2. EL doesn’t provide a way to set properties

The JSP Expression Language(EL) provides a compact syntax that lets you get data from JavaBeans, maps, arrays, and lists that have been stored as attributes of a web application

247:Example accessing current date

Example that accesses the firstName property of the user attribute

Sequence of scopes Java searches to find the attributes (page, request, session, application)

A JavaBean is a special type of object that provides a standard way to access its properties

A map is a special tyoe of collection that’s used to store key/value pairs: HashMap is a collection map

When you use the dot operator, the code to the left of the operator must specify a JavaBean or a map, and the code to the right of the operator must specify a JavaBean or map key

When you use this syntax, EL looks up the attribute starting with the smallest scope(page ) to largest scope (application)

249: implicit EL objects for specifying scope

1. pageScope
2. requestScope
3. sessionScope
4. applicationScope

Example specifies request scope

Example specifies session scope

If you have a naming conflict, you can use the implicit eL objects to specify scope

All of the implicit EL objects for specyfing scope are maps. As a result, you can use the dot operator to specify a key when you want to return the object for that key

251: The syntax for the [] operator

An example that works with a JavaBean property

An example that works with an array

AN example that works with a list

A list is a special type of collection such as an ArrayList<> that uses an index to retrieve an object that’s stored in the collection

Although the [] operator can be used to work with JavaBeans and maps, it is commonly used to work with arrays and lists.

With EL, the quotation marks are required for specifying a property in a JavaBean or a key in a map, but the quotation marks are optional when specyfing an index of an array or list

253: How to use the dot operator to access nested properties

Example

If a JavaBean has a property that retrns another JavaBean, You can use the dot operator to access nested properties

There is no limit to the number of nested properties that you can access with the dot operator

255:How to use the [ ] operator to access attributes

If the expression within [ ] isn’t enclosed within quotes, El evaluates the expression. To start, EL checks to see if the expression is an attribute, Then it attempts to evaluate the expression

If multiple [] operators exist, the expression is evalusted from the innermost [ ] to the outermost. You can nest as many as necessary

257: Other implicit objects you can use: param, paramValues, header, headerValues, cookie, initParam, pageContext

How to get param values from the request

How to get an HTTP header

How to work with cookies

259: How to get a context initialization parameter

How to use the pageContext object

4 implicit EL objects specifying scope

All implicit objects used by EL are maps, except for the pageContext object, which is a JavaBean

261: Arithmetic & Relational EL operators

263: Logical operators

Other operators : 1.) empty x 2.) x ? y : z

Keywords : null true false

For arithmetic expressions, you can use parenthesis to control or clarify the order

In arithmetic expressions, null is zero

In logical expressions, null is fallse

265:How to disable EL

How to disable scripting

\*\*\*\*\*\*\*\*\*\*\*\*Chapter 9: JSTL \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

271: JSP Standard Tag Library provides tags for common tasks that need to be performed in JSPs

Primary JSTL Libraries:

Core c

Formatting fmt

SQL sql

XML x

Functions fn

The taglib directive that specifies the JSTL core library

Example that uses JSTL to escape the output

Before you can use JSTL tags within an application, you must make the jstl-impl.jar and jstl-api.jar files available to the application.

Before you can use JSTL tags within a JS, you must code a taglib directive that identifies the JSTL library and its prefix

273: How to view the documentation for a library

275: How to use the out tag

Example that escapes output entered by the user

Output for an XSS attack

Example that escapes output that displays cookie names and values

Display default value

Using single quotes

277: forEach Tag

Example that uses JSTL to loop through a collection

279: forTokens tag

Example using JSTL to loop through comma-delimited string

Example to parse a string

281: 4 more attributes for looping ( begin, end, step, varStatus)

283: if tag

285: choose tag for if/else

287: url tag

289: other tags: set, remove, catch, redirect, param

291: cart app

293: code for business classes : product class

294: LineItem class

295: Cart class

296: code for servlets & Jsps: index.jsp

299: Cartservlet class

303: cart.jsp

\*\*\*\*\*\*\*\*\*\*\*\*Chapter 10: Custom JSP tag\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

309: Custom tag without a body

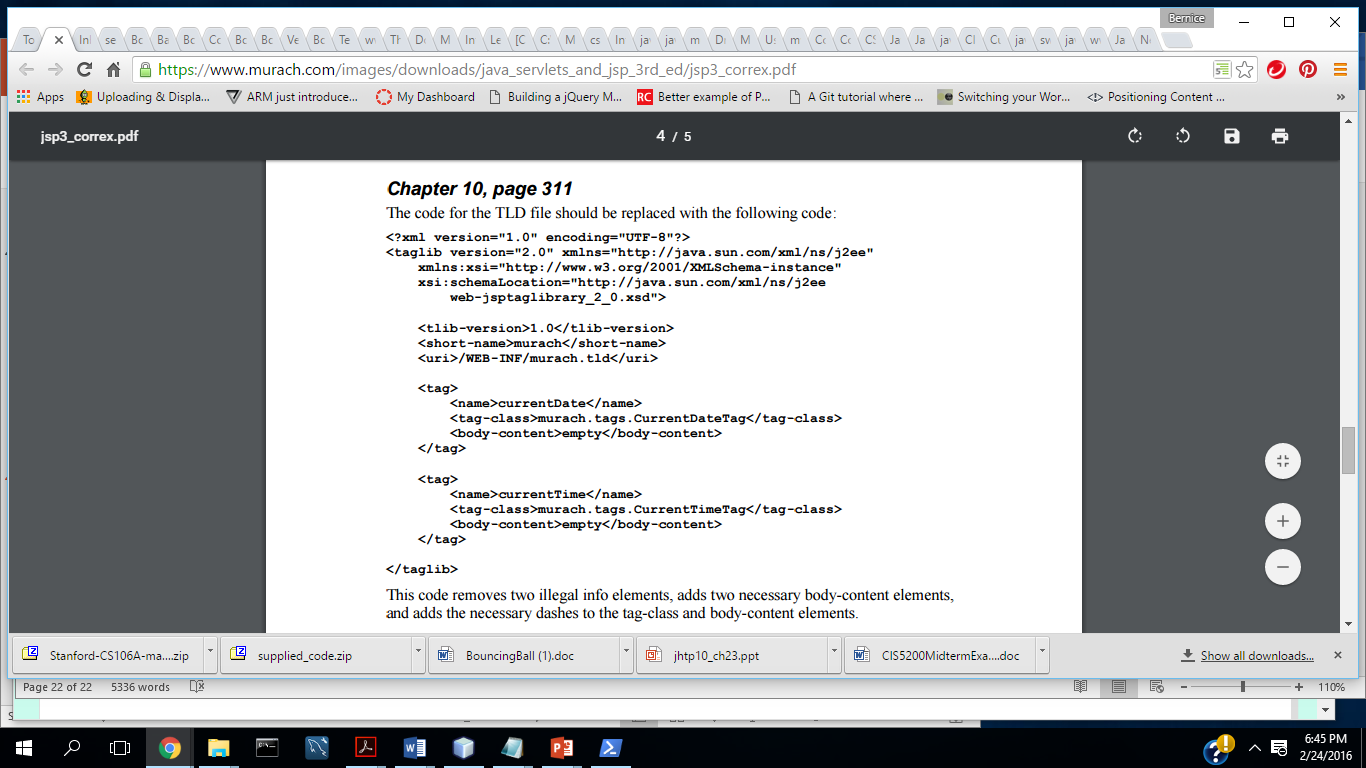
Taglib directive

311: tag element

Tag Library descriptor ( TLD) is an XML document that descripes a tag library that contains custom tags that can be used tin an app. No limit to TLDs

Example: TLD with 2 tag elements

Chapter 10, page 311 The code for the TLD file should be replaced with the following code: 1.0 murach /WEB-INF/murach.tld currentDate murach.tags.CurrentDateTag empty currentTime murach.tags.CurrentTimeTag empty This code removes two illegal info elements, adds two necessary body-content elements, and adds the necessary dashes to the tag-class and body-content elements.



313: tag class for a tag without a body

315: JsP code that uses a custom tag with a body

317 tag element in the TLD file

Tag class tha conditionally displays the body of a tag

319: JSP code that uses a custom tag that has attributes

321: syntax for the attribute element in a tag element

Tag element with 2 attributes

An attribute element that uses the integer data type

The attribute child elements: name, required, rtexprvalue, type

323: tag class that uses two attributes

How to code a tag handler that uses attributes

325: how to code a custom tag that reiterates its body

Servlet code that sets the cart attribute

JSP code that displays all items in the cart (using JSP expressions)

JSP code that displays all items in the cart (using El)

327: Tag element for TLD

Tag class that reiterates the body of the tag

329:

331: How to work with scripting variables

Custom JSP tag without scripting variables

Custom JSP tag wit scripting variables

Code in tag class that adds the scripting variable to the pageContext object

The tag element in the TLD

How to create a scripting variable

333: TEI class for 4 scripting variables

335: common methods and fields of the TagSupport class

337: methods and fields of the PageContext class

339: Methods and fields of the BodyTagSupport class

341: Methods and fields of the BodyContent class