

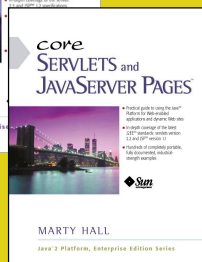
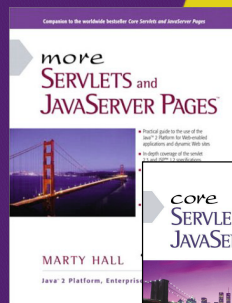


Exam Prep

Sun Certified Web Component Developer (SCWCD) for J2EE Platform

Core Servlets & JSP book: www.coreservlets.com
More Servlets & JSP book: www.moreservlets.com
Servlet and JSP Training Courses: courses.coreservlets.com

Slides © Marty Hall: <http://www.moreservlets.com/>, book © Sun Microsystems Press



For full SCWCD preparation, please see JSP and servlet training course at <http://courses.coreservlets.com/>.

Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this review. Available at public venues, or customized versions can be held on-site at your organization.

Slides © Marty Hall: <http://www.moreservlets.com/>, book © Sun Microsystems Press

Overview

- **Must have Java programmer certification first**
- **Emphasis on memorization**
 - Names of classes, methods, and packages
 - Names of JSP variables and associated classes
 - Servlet life cycle
 - Custom JSP tag libraries
 - web.xml elements
- **Official Web Site**
 - http://suned.sun.com/US/certification/java/exam_objectives.html
- **Format of following slides**
 - Topic from Sun objectives in black (quoted verbatim)
 - Answers and comments in red
 - Topics cross referenced to associated sections of *More Servlets and JavaServer Pages* (see <http://www.moreservlets.com>)

4

Java Web Component Certification

SCWCD Training: <http://courses.coreservlets.com/>

Section 1 – The Servlet Model

- **1.1 For each of the HTTP methods, GET, POST, and PUT, identify the corresponding method in the HttpServlet class.**
 - doGet
 - doPost
 - doPut
 - (doHead, doDelete, doTrace, doOptions)

Details: Section 2.3 of *More Servlets and JavaServer Pages*

5

Java Web Component Certification

SCWCD Training: <http://courses.coreservlets.com/>

Section 1 – The Servlet Model

- **1.2 For each of the HTTP methods, GET, POST, and HEAD, identify triggers that might cause a browser to use the method, and identify benefits or functionality of the method.**
 - GET
 - Type URL in address line or click on hypertext link
 - Submit form with no METHOD or METHOD="GET"
 - POST
 - Submit form with METHOD="POST"
 - HEAD
 - I assume question is getting at checking if cached documents are up to date, but all major browsers use conditional GET requests (i.e., with If-Modified-Since, not HEAD for this. HEAD returns HTTP headers but not document.)
- Details: Section 2.3 of *More Servlets and JavaServer Pages*

6

Java Web Component Certification

SCWCD Training: <http://courses.coreservlets.com/>

Section 1 – The Servlet Model

- **1.3 For each of the following operations, identify the interface and method name that should be used:**
 - Retrieve HTML form parameters from the request
 - `request.getParameter` (`HttpServletRequest`)
 - Retrieve a servlet initialization parameter
 - `ServletConfig.getInitParameter`
 - Retrieve HTTP request header information
 - `request.getHeader` (`HttpServletRequest`)
 - Set an HTTP response header; set the content type of the response
 - `response.setHeader`, `response.setContentType` (`HttpServletResponse`)
 - Acquire a stream for the response
 - `response.getWriter` (`HttpServletResponse`)
 - Acquire a binary stream for the response
 - `response.getOutputStream` (`HttpServletResponse`)
 - Redirect an HTTP request to another URL
 - `response.sendRedirect` (`HttpServletResponse`)

7

Java Web Component Certification

SCWCD Training: <http://courses.coreservlets.com/>

Section 1 – The Servlet Model

- **1.4 Identify the interface and method to access values and resources and to set object attributes within the following three Web scopes:**
 - Request
 - `request.getAttribute`, `request.setAttribute` (`HttpServletRequest`)
 - Session
 - `session.getAttribute`, `session.setAttribute` (`HttpSession`)
 - Obtain session via `request.getSession(true)`
 - Context
 - `context.getAttribute`, `context.setAttribute` (`ServletContext`)
 - Obtain context via `getServletContext()`

Details: Section 3.8 of *More Servlets and JavaServer Pages*

Section 1 – The Servlet Model

- **1.5 Given a life-cycle method: `init`, `service`, or `destroy`, identify correct statements about its purpose or about how and when it is invoked.**
 - `init`
 - Called when servlet is first loaded (servlet instance created). *Not* called for each request.
 - `service`
 - Called for each request. Calls `doGet`, `doPost`, etc.
 - `destroy`
 - Called when the server unloads a servlet (usually because servlet hasn't been accessed in a long time)

Details: Section 2.3 of *More Servlets and JavaServer Pages*

Section 1 – The Servlet Model

- **1.6 Use a RequestDispatcher to include or forward to a Web resource.**
 - Forwarding (request-based)

```
SomeClass value = new SomeClass(...);
request.setAttribute("key", value);
String url = "/some-dir/presentation1.jsp";
RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher(url);
dispatcher.forward();
```
 - For including, use `include` instead of `forward` and you can do more processing after the call.
 - There is also session-based and context-based sharing
 - Don't forget to synchronize for those
 - Note: JSP page can be in WEB-INF.

Details: Section 3.8 of *More Servlets and JavaServer Pages*

10

Java Web Component Certification

SCWCD Training: <http://courses.coreservlets.com/>

Section 2 – The Structure & Deployment of Modern Servlet Web Applications

- **2.1 Identify the structure of a Web Application and Web Archive file, the name of the WebApp deployment descriptor, and the name of the directories where you place the following:**
 - The WebApp deployment descriptor
 - WEB-INF/web.xml
 - The WebApp class files (assumedly they mean unjarred)
 - WEB-INF/classes (unpackaged)
 - WEB-INF/classes/directoryMatchingPackageName
 - Any auxiliary JAR files
 - WEB-INF/lib

Details: Chapter 4 of *More Servlets and JavaServer Pages*

11

Java Web Component Certification

SCWCD Training: <http://courses.coreservlets.com/>

Section 2 – The Structure & Deployment of Modern Servlet Web Applications

- **2.2 Match the name with a description of purpose or functionality, for each of the following deployment descriptor elements:**
 - Servlet instance (note: used for JSP pages also)
 - `servlet`
 - Servlet name
 - `servlet-name` (inside `servlet`)
 - Servlet class
 - `servlet-class` (inside `servlet`)
 - Initialization parameters
 - `init-param` (inside `servlet`)
 - URL to named servlet mapping
 - `servlet-mapping`

Details: Section 5.3 of *More Servlets and JavaServer Pages*

SCWCD Training: <http://courses.coreservlets.com/>

12

Java Web Component Certification

Section 3 – The Servlet Container Model

- **3.1 Identify the uses for and the interfaces (or classes) and methods to achieve [sic] the following features:**
 - Servlet context init. parameters
 - `ServletContext.getInitParameter`
 - Servlet context listener
 - `ServletContextListener`
 - Servlet context attribute listener
 - `ServletContextAttributeListener`
 - Session attribute listeners
 - `HttpSessionAttributeListener`
 - They didn't ask, but there is also `HttpSessionListener`

Details: Chapter 10 of *More Servlets and JavaServer Pages*

SCWCD Training: <http://courses.coreservlets.com/>

13

Java Web Component Certification

Section 3 – The Servlet Container Model

- **3.2 Identify the WebApp deployment descriptor element name that declares the following features:**
 - Servlet context init. parameters
 - *context-param*
 - Servlet context listener
 - *listener*
 - Servlet context attribute listener
 - *listener*
 - Session attribute listeners
 - *listener*

Details: Chapter 10 of [More Servlets and JavaServer Pages](#)

Section 3 – The Servlet Container Model

- **3.3 Distinguish the behavior of the following in a distributable:**
 - Servlet context init. parameters
 - Servlet context listener
 - Servlet context attribute listener
 - Session attribute listeners
- Presumably they mean "in a distributed Web app." The point is that the servlet context and sessions are guaranteed to be shared in a Web app running on a distributed (clustered) server. But, listener instances have no such guarantee, so they cannot store data in their instance variables (fields). Same goes for instance variables of servlets: not safe in distributed apps.

Section 4 – Designing and Developing Servlets to Handle Server-side Exceptions

- **4.1 For each of the following cases, identify correctly constructed code for handling business logic exceptions, and match that code with correct statements about the code's behavior:**
 - Return an HTTP error using the `sendError` response method (Section 2.7 of *More Servlets and JSP*)
 - `response.sendError(response.SC_NOT_FOUND, "Error Msg in HTML");`
 - The server wraps message in HTML page. Remember problems with IE 5 (`SC_NOT_FOUND` is 404).
 - Return an HTTP error using the `setStatus` method. (Section 2.7 of *More Servlets and JSP*)
 - `response.setStatus(someInt)`
 - Use constants instead of explicit ints. What other "error" code would servlets commonly send other than 404 and 302?

Section 4 – Designing and Developing Servlets to Handle Server-side Exceptions

- **4.2 Given a set of business logic exceptions, identify the following:**
 - The configuration that the deployment descriptor uses to handle each exception
 - How does this differ from third item (below)?
 - How to use a `RequestDispatcher` to forward the request to an error page (Section 3.8 of *More Servlets & JSP*)

```
SomeError value = new SomeError(...);
request.setAttribute("key", value);
String url = "/some-dir/error.jsp";
RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher(url);
dispatcher.forward();
```
 - Specify the handling declaratively in the deployment descriptor (Section 5.8 of *More Servlets & JSP*)
 - Use error-page with exception-type

Section 4 – Designing and Developing Servlets to Handle Server-side Exceptions

- **4.3 Identify the method used for the following:**
 - Write a message to the WebApp log
 - `log("Message");`
 - The above is a method of `GenericServlet`. You can also call `log` on the `ServletContext` – same result.
 - Write a message and an exception to the WebApp log.
 - `log("Message", someThrowable);`
 - The above is a method of `GenericServlet`. You can also call `log` on the `ServletContext` – same result.

Section 5 – Designing and Developing Servlets Using Session Management

- **5.1 Identify the interface and method for each of the following:**
 - Retrieve a session object across multiple requests to the same or different servlets within the same WebApp
 - `request.getSession(true);`
 - Store objects into a session object
 - `session.setAttribute("key", value);`
 - Retrieve objects from a session object
 - `(SomeClass)session.getAttribute("key");`
 - Respond to the event when a particular object is added to a session
 - Use `HttpSessionAttributeListener`
 - Respond to the event when a session is created and destroyed
 - Use `HttpSessionListener`
 - Expunge a session object
 - `session.invalidate();`

Section 5 – Designing and Developing Servlets Using Session Management

- **5.2 Given a scenario, state whether a session object will be invalidated.**
 - This is a fuzzy question. The normal cases when session objects are invalidated is when user code calls `invalidate` or the session times out.
- **5.3 Given that URL-rewriting must be used for session management, identify the design requirement on session-related HTML pages.**
 - Call `response.encodeURL` on all URLs output to client.
 - Call `response.encodeRedirectURL` on all URLs that are sent in Location header with 301/302 responses (as with `response.sendRedirect`).

Details: Section 2.10 of *More Servlets & JavaServer Pages*

SCWCD Training: <http://courses.coreservlets.com/>

Section 6 - Designing and Developing Secure Web Applications

- **6.1 Identify correct descriptions or statements about the security issues: (Section 7.1 of *More Servlets & JSP*)**
 - Authentication, authorization
 - Authentication is making sure a user is who they claim to be.
 - Authorization is making sure that the authenticated user is allowed to do what they are trying to do.
 - Data integrity
 - Making sure the information hasn't been tampered with. In this context, SSL is they key point. (Actually, making sure the information hasn't been *seen* is equally important.)
 - Auditing
 - Logging what is going on to trace attacks later.
 - Malicious code
 - This is usually discussed in the context of attacks from Web sites vs. to Web sites. Eg site sends your browser an ActiveX control or something else that attacks you.
 - Web site attacks
 - General category of security breaches vs. a Web site.

Section 6 - Designing and Developing Secure Web Applications

- **6.2 Identify the deployment descriptor element names, and their structure, that declare the following:**
 - A security constraint
 - **security-constraint**
 - A Web resource
 - **web-resource-collection** (within security-constraint)
 - The login configuration
 - **login-config** (contains auth-method)
 - A security role
 - **role-name** (within auth-constraint, which is within security-constraint)

Details: Section 7.1 of *More Servlets & JavaServer Pages*

Section 6 - Designing and Developing Secure Web Applications

- **6.3 Given an authentication type: BASIC, DIGEST, FORM, and CLIENT-CERT, identify the correct definition of its mechanism.**
 - **BASIC:** Ask for username in dialog box. Track users with HTML headers. Password not encrypted.
 - **DIGEST:** Like BASIC except password is encrypted. Unsupported by most browsers.
 - **FORM:** Ask for username with regular HTML form. Track users with modified session tracking.
 - **CLIENT-CERT:** Get username from X509 certificate. Track users with modified session tracking.

Details: Chapter 7 of *More Servlets & JavaServer Pages*

Section 7 – Designing and Developing Thread-safe Servlets

- **7.1 Identify which attribute scopes are thread-safe:**
 - Local variables
 - Yes.
 - Instance variables
 - No, unless servlet implements `SingleThreadModel`.
(Note: `SingleThreadModel` deprecated in JSP 2.0/servlets 2.4)
 - Class variables
 - No, not even if servlet implements `SingleThreadModel`.
 - Request attributes
 - Yes.
 - Session attributes
 - Technically, no. Same client could access multiple times.
 - Context attributes
 - No.

24

Java Web Component Certification

SCWCD Training: <http://courses.coreservlets.com/>

Section 7 – Designing and Developing Thread-safe Servlets

- **7.2 Identify correct statements about differences between the multi-threaded and single-threaded servlet models.**
 - When using `SingleThreadModel`, no concurrent access is allowed for a single servlet instance. But servers are allowed to use a pool of instances and allow concurrent access to the pool.
- **7.3 Identify the interface used to declare that a servlet must use the single thread model.**
 - `public class xxxx extends HttpServlet`
`implements SingleThreadModel`
(Note: deprecated in JSP 2.0/servlets 2.4)
 - See Section 2.3 of *More Servlets and JavaServer Pages*
 - For JSP, use `<%@ page isThreadSafe="false" %>`
 - See Section 3.4 of *More Servlets and JavaServer Pages*

25

Java Web Component Certification

SCWCD Training: <http://courses.coreservlets.com/>

Section 8 - The Java Server Pages (JSP) Technology Model

- **8.1 Write the opening and closing tags for the following JSP tag types:**

Details: Section 3.3 of *More Servlets and JavaServer Pages*

- Directive
 - `<%@ ... %>`
 - `<jsp:directive.xxxx>...</jsp:directive.xxxx>`
- Declaration
 - `<%! ... %>`
 - `<jsp:declaration>...</jsp:declaration>`
- Scriptlet
 - `<% ... %>`
 - `<jsp:scriptlet>...</jsp:scriptlet>`
- Expression
 - `<%= ... %>`
 - `<jsp:expression>...</jsp:expression>`

Section 8 - The Java Server Pages (JSP) Technology Model

- **8.2 Given a type of JSP tag, identify correct statements about its purpose or use.**

- **Declaration:** code (method or field) that is part of servlet class that results from JSP document. Use to override `jspInit` or define fields (for persistence), usually.
- **Scriptlet:** code that is part of `_jspService` method. Use for side effects.
- **Expression:** code that is inside a print statement in `_jspService`. Use to output values.
- **Directives:** more global control of resulting code.

- **8.3 Given a JSP tag type, identify the equivalent XML-based tags.**

- See previous slide.

Details: Section 3.3 of *More Servlets and JavaServer Pages*

Section 8 - The Java Server Pages (JSP) Technology Model

- **8.4 Identify the page directive attribute, and its values, that:**
 - Import a Java class into the JSP page
 - `<%@ page import="..." %>`
 - Declare that a JSP page exists within a session
 - **None. This is the default.**
 - Use `<%@ page session="false" %>` to disable that.
 - Declare that a JSP page uses an error page
 - `<%@ page errorPage="..." %>`
 - Declare that a JSP page is an error page
 - `<%@ page isErrorPage="true" %>`

Details: Section 3.4 of *More Servlets and JavaServer Pages*

Section 8 - The Java Server Pages (JSP) Technology Model

- **Identify and put in sequence the following elements of the JSP page lifecycle:**
 - Page translation
 - When the JSP page is translated into a servlet. Happens the *very* first time anyone requests the JSP page.
 - JSP page compilation
 - When the servlet that resulted from page translation is compiled. Happens immediately after page translation.
 - Load class
 - When the class of the servlet (the one that resulted from the JSP page) is loaded into the server's memory. Normally happens the first time the JSP page is requested since the server has been restarted (or Web app reloaded), but can happen at server startup if load-on-startup is specified.
 - Create instance
 - When an object of the servlet's class is created. Normally happens once immediately after the class is loaded, but might happen multiple times if `isThreadSafe="false"` is specified.

Section 8 - The Java Server Pages (JSP) Technology Model

- **Identify and put in sequence the following elements of the JSP page lifecycle:**
(Continued)
 - Call `jspInit`
 - Happens immediately after the servlet instance is created.
 - Call `_jspService`
 - Happens for each request. This is the method that really handles the request and is analogous to `service` in a regular servlet (ie like `doGet` or `doPost` but both GET and POST are sent to `_jspService`).
 - Call `jspDestroy`
 - If the server unloads a servlet instance (e.g., if it hasn't been accessed in a long time), then this method is called.

Section 8 - The Java Server Pages (JSP) Technology Model

- **8.6 Match correct descriptions about purpose, function, or use with any of the following implicit objects:**
 - request
 - The `HttpServletRequest` passed to `_jspService`
 - response
 - The `HttpServletResponse` passed to `_jspService`
 - out
 - The `JspWriter` that sends data to client
 - session
 - The current `HttpSession`. This variable doesn't exist if you do `<%@ page session="false" %>`.
 - config
 - The `ServletConfig` with which you can read init params.

Section 8 - The Java Server Pages (JSP) Technology Model

- **8.6 Match correct descriptions about purpose, function, or use with any of the following implicit objects: (Continued)**
 - application
 - The `ServletContext`.
 - page
 - Synonymous with "this". Is reference to current servlet instance.
 - `pageContext`
 - The current `PageContext` object -- a one-stop shopping location for servlet objects (it has methods to get the request, response, servlet context, etc.)
 - exception
 - The current exception. This variable is *only* bound in error pages, and only when the system transfers to them automatically.

Section 8 - The Java Server Pages (JSP) Technology Model

- **8.7 Distinguish correct and incorrect scriptlet code for:**
 - Well, there are any number of "incorrect code" examples. I'll give two examples of each that are technically legal, but perhaps what they are getting at is that they think the first is better style.
 - A conditional statement
 - ```
<% if(someCondition) { %>
<H2>Some HTML Result</H2>
<% } else { %>
<H2>A Different HTML Result</H2>
<% } %>
```
    - ```
<% if(someCondition) {
    out.println(" <H2>Some HTML Result</H2>");
} else {
    out.println("<H2>A Different HTML Result");
} %>
```

Section 8 - The Java Server Pages (JSP) Technology Model

- **8.7 Distinguish correct and incorrect scriptlet code for:**
 - An iteration statement
 - `<TABLE ...>`
`<% for(...) { %>`
`<TR>...`
`<% } %>`
`</TABLE>`
 - `<TABLE ...>`
`<% for(...) {`
`out.println("<TR>...");`
`} %>`
`</TABLE>`

Section 9 - Designing and Developing Reusable Web Components

- **9.1 Given a description of required functionality, identify the JSP page directive or standard tag in the correct format with the correct attributes required to specify the inclusion of a Web component into the JSP page. (See *More Servlets & JSP Section 3.5*)**
 - `<jsp:include page="/foo/bar.jsp"/>`
 - *Output of bar.jsp included at request time.* This is normally best option due to easier maintenance.
 - `<%@ include file="/foo/bar.jsp" %>`
 - *The actual text of bar.jsp is included at page translation time.* Use this when bar.jsp defines methods or fields that you need to use in the main page (i.e., when just the output of bar.jsp is not enough). Some people also cite efficiency advantages of this; I'm skeptical, but be aware of this issue in case they ask it on exam.

Section 10 - Designing and Developing JSP pages Using JavaBean Components

- **10.1** For any of the following tag functions, match the correctly constructed tag, with attributes and values as appropriate, with the corresponding description of the tag's functionality: (**See *More Servlets & JSP Section 3.6***)
 - Declare the use of a JavaBean component within the page.
 - `<jsp:useBean id="..." class="..." scope="..."/>`
 - Note that scope is optional (defaults to page)
 - Specify, for `jsp:useBean` or `jsp:getProperty` tags, the name of an attribute.
 - `<jsp:useBean id="..." .../>`
 - `<jsp:getProperty name="..." ...>`

Section 10 - Designing and Developing JSP pages Using JavaBean Components

- **10.1** For any of the following tag functions, match the correctly constructed tag, with attributes and values as appropriate, with the corresponding description of the tag's functionality: (**Continued**)
 - Specify, for a `jsp:useBean` tag, the class of the attribute.
 - `<jsp:useBean id="..." class="..." scope="..."/>`
 - Note that you use the fully qualified class (package.Class), regardless of whether you have `<%@ page import="..." %>`
 - Specify, for a `jsp:useBean` tag, the scope of the attribute.
 - `<jsp:useBean id="..." class="..." scope="..."/>`

Details: Section 3.6 of *More Servlets and JavaServer Pages*

Section 10 - Designing and Developing JSP pages Using JavaBean Components

- **10.1** For any of the following tag functions, match the correctly constructed tag, with attributes and values as appropriate, with the corresponding description of the tag's functionality: **(Continued)**
 - Access or mutate a property from a declared JavaBean.
 - Access: `<jsp:getProperty name="..." property="..."/>`
 - Mutate: `<jsp:setProperty name="..." property="..." value="..."/>`
 - Specify, for a `jsp:getProperty` tag, the property of the attribute.
 - `<jsp:getProperty name="..." property="..."/>`
 - Specify, for a `jsp:setProperty` tag, the property of the attribute to mutate, and the new value.
 - `<jsp:setProperty name="..." property="..." value="..."/>`

Section 10 - Designing and Developing JSP pages Using JavaBean Components

- **10.2** Given JSP page attribute scopes: request, session, application, identify the equivalent servlet code. **(More Servlets & JSP Section 3.8)**
 - Assume `<jsp:useBean id="foo" class="bar.Baz" .../>`
 - Assume in servlet code that `someBar` is of type `bar.Baz`.
 - **request**
 - Store: `request.setAttribute("foo", someBar);`
 - Access: `(bar.Baz)request.getAttribute("foo");`
 - **session**
 - First: `HttpSession session = request.getSession(true);`
 - Store: `session.setAttribute("foo", someBar);`
 - Access: `(bar.Baz)session.getAttribute("foo");`
 - **application**
 - Store: `getServletContext().setAttribute("foo", someBar);`
 - Access: `(bar.Baz)getServletContext().getAttribute("foo");`

Section 10 - Designing and Developing JSP pages Using JavaBean Components

- **10.3 Identify techniques that access a declared JavaBean component.**
 - In servlet: see previous page.
 - In JSP:
 - `<jsp:useBean id="..." class="..." scope="request"/>`
 - `<jsp:useBean id="..." class="..." scope="session"/>`
 - `<jsp:useBean id="..." class="..." scope="application"/>`
 - SCWCD currently does not cover JSP 2.0. If future versions do, then `#{beanName}` is also an answer.

Details: Section 3.6 of *More Servlets and JavaServer Pages*

Section 11 - Designing and Developing JSP pages Using Custom Tags

- **11.1 Identify properly formatted tag library declarations in the Web application deployment descriptor. (**web.xml**)**
 - `<taglib>`
 - `<taglib-uri>someName</taglib-uri>`
 - `<taglib-location>/WEB-INF/blah.tld</taglib-location>`
 - `</taglib>`

Details: Section 3.7 of *More Servlets and JavaServer Pages*

Section 11 - Designing and Developing JSP pages Using Custom Tags

- **11.2 Identify properly formatted taglib directives in a JSP page.**
 - `<%@ taglib uri="..." prefix="..." %>`

Details: Section 3.7 of *More Servlets and JavaServer Pages*

Section 11 - Designing and Developing JSP pages Using Custom Tags

- **11.3 Given a custom tag library, identify properly formatted custom tag usage in a JSP page. Uses include: (*MSAJSP Section 3.7*)**
 - An empty custom tag
 - `<prefix:tagName/>` (don't forget that slash at end)
 - A custom tag with attributes
 - `<prefix:tagName someAttribute="blah" />`
 - A custom tag that surrounds other JSP code
 - `<prefix:tagName>Content</prefix:tagName>`
 - Nested custom tags
 - `<prefix:tagName1>`
 - `<prefix:tagName2>Content</prefix:tagName2>`
 - `<prefix:tagName3/>`
 - Content
 - `</prefix:tagName1>`

Section 12 - Designing and Developing a Custom Tag Library

- **12.1 Identify the tag library descriptor element names that declare the following:**

- The name of the tag
- The class of the tag handler
- The type of content that the tag accepts
- Any attributes of the tag

`<taglib>...`

`<tag>`

`<name>...</name>`

`<tagclass>...</tagclass>`

`<bodycontent>...</bodycontent>`

`<attribute>...</attribute>`

`</tag>...</taglib>`

- Note: Use tag-class and body-content in code specific to JSP 1.2
Details: Section 3.7 of *More Servlets and JavaServer Pages*

Section 12 - Designing and Developing a Custom Tag Library

- **12.2 Identify the tag library descriptor element names that declare the following:**

- The name of a tag attribute
- Whether a tag attribute is required
- Whether or not the attribute's value can be dynamically specified

• `<attribute>`

`<name>...</name>`

`<required>...</required>`

`<rtexprvalue>...</rtexprvalue>`

`</attribute>`

- Note the above goes inside tag element, after bodycontent and info.

Details: Section 3.7 of *More Servlets and JavaServer Pages*

Section 12 - Designing and Developing a Custom Tag Library

- **12.3 Given a custom tag, identify the necessary value for the bodycontent TLD element for any of the following tag types:**
 - Empty-tag
 - `<bodycontent>empty</bodycontent>`
 - Custom tag that surrounds other JSP code
 - `<bodycontent>JSP</bodycontent>`
 - Custom tag that surrounds content that is used only by the tag handler
 - `<bodycontent>tagdependent</bodycontent>`
 - Remember, bodycontent renamed body-content in JSP 1.2. You only need to do it the new way if you use the JSP 1.2 DTD for your TLD file.

Section 12 - Designing and Developing a Custom Tag Library

- **12.4 Given a tag event method (doStartTag, doAfterBody, and doEndTag), identify the correct description of the methods trigger.**
 - `<prefix:tagName>Blah blah</prefix:tagName>`
 - `<prefix:tagName>`
 - *doStartTag runs when parser sees this.*
 - Blah blah
 - *doAfterBody runs when parser is done seeing this – i.e., just before starting to handle the closing tag.*
 - `</prefix:tagName>`
 - *doEndTag runs when parser sees this*

Details: Section 3.7 of *More Servlets and JavaServer Pages*

Section 12 - Designing and Developing a Custom Tag Library

- **12.5 Identify valid return values for the following methods:**
 - doStartTag
 - SKIP_BODY: empty (standalone) tag – ignore any tag body
 - EVAL_BODY_INCLUDE: include tag body
 - doAfterBody
 - EVAL_BODY_TAG: repeat processing of body (renamed to EVAL_BODY_AGAIN in JSP 1.2)
 - SKIP_BODY: done
 - doEndTag
 - EVAL_PAGE: carry on
 - SKIP_PAGE: skip rest of page
 - PageContext.getOut [sic]
 - Returns the JspWriter

Section 12 - Designing and Developing a Custom Tag Library

- **12.6 Given a "BODY" or "PAGE" constant, identify a correct description of the constant's use in the following methods:**
 - doStartTag
 - SKIP_BODY: empty (standalone) tag – ignore any tag body
 - EVAL_BODY_INCLUDE: include tag body
 - doAfterBody
 - EVAL_BODY_TAG: repeat processing of body (renamed to EVAL_BODY_AGAIN in JSP 1.2)
 - SKIP_BODY: done
 - doEndTag
 - EVAL_PAGE: carry on
 - SKIP_PAGE: skip rest of page

Details: Section 3.7 of *More Servlets and JavaServer Pages*

Section 12 - Designing and Developing a Custom Tag Library

- **12.7 Identify the method in the custom tag handler that accesses:**
 - A given JSP page's implicit variable
 - `pageContext`
 - The JSP page's attributes
 - `pageContext.getOut()`
 - `pageContext.getRequest()`
 - `pageContext.getResponse()`
 - `pageContext.getSession()`
 - `pageContext.getServletContext()`

Section 12 - Designing and Developing a Custom Tag Library

- **12.8 Identify methods that return an outer tag handler from within an inner tag handler.**
 - `getParent`: most immediate enclosing tag
 - `findAncestorWithClass`: enclosing tag of given type
 - Might return null
 - Used more commonly than `getParent`

Details: Section 3.7 of *More Servlets and JavaServer Pages*

Section 13

- **13.1 Given a scenario description with a list of issues, select the design pattern (Value Objects, MVC, Data Access Object, or Business Delegate) that would best solve those issues.**
 - It is almost always useful to have separate classes to do logic that is repeated multiple times.
 - This is especially true in JSP, where removing the business logic from the page lets you divide up your development team more cleanly
 - MVC particularly useful when the same submission can yield multiple substantially different results
- Note: except for MVC (*MSAJSP* Section 3.8), these patterns are not mentioned by name in *MSAJSP*. Definitions on next slide are taken verbatim from *Core J2EE Patterns*.

52

Java Web Component Certification

SCWCD Training: <http://courses.coreservlets.com/>

Section 13

- **13.2 Match design patterns with statements describing potential benefits that accrue from the use of the pattern, for any of the following patterns:**
 - Value Objects
 - Facilitates data exchange between tiers by reducing network chattiness.
 - MVC
 - Lets servlet do what it is good at: creating and processing data. Lets JSP do what it is good at: presentation.
 - Data Access Object
 - Abstracts data sources; provides transparent access to data
 - Business Delegate
 - Decouples presentation and service tiers, and provides a façade and proxy interface to the services.

53

Java Web Component Certification

SCWCD Training: <http://courses.coreservlets.com/>



Questions?

Core Servlets & JSP book: www.coreservlets.com
More Servlets & JSP book: www.moreservlets.com
Servlet and JSP Training Courses: courses.coreservlets.com

Slides © Marty Hall: <http://www.moreservlets.com/>, book © Sun Microsystems Press

More Information

- **Source code for examples**
 - <http://www.moreservlets.com>
- **More Servlets & JSP**
 - <http://www.moreservlets.com>
 - Site includes info on servlet and JSP training courses
- **Core Servlets & JSP**
 - Prequel to *More Servlets & JSP*
 - <http://www.coreservlets.com>
- **Servlet home page**
 - <http://java.sun.com/products/servlet/>
- **JavaServer Pages home page**
 - <http://java.sun.com/products/jsp/>

