

## 310-083 SCWCD

### SUN Sun Certified Web Component Developer for J2EE 5

**Practice Exam:** 310-083 Exams

**Exam Number/Code:** 310-083

**Exam Name:** Sun Certified Web Component Developer for J2EE 5

**Questions and Answers:** 276 Q&As

( [SCWCD](#) )



Exam : [310-083](#)

"Sun Certified Web Component Developer for J2EE 5", also known as 310-083 exam, is a SUN certification. With the complete collection of questions and answers, TestInside has assembled to take you through 276 Q&As to your 310-083 Exam preparation. In the 310-083 exam resources, you will cover every field and category in SUN Certification helping to ready you for your successful SUN Certification.

Quality and Value for the 310-083 Exam TestInside Practice Exams for SUN **SCWCD** Certification 310-083 are written to the highest standards of technical accuracy, using only certified subject matter experts and published authors for development.

#### ***TestInside provide the professional Q&A.***

1. We offer free update service for three month.

After you purchase our product, we will offer free update in time for three month.

2. High quality and Value for the 310-083 Exam.

310-083 simulation test questions, including the examination question and the answer, complete by our senior IT lecturers and the SCWCD product experts, included the current newest 310-083 examination questions.

3. 100% Guarantee to Pass Your SCWCD exam and get your SCWCD Certification.

If you do not pass the SUN Certification 310-083 exam (Sun Certified Web Component Developer for J2EE 5) on your first attempt using our TestInside testing engine and pdf file, we will give you a FULL REFUND of your purchasing fee.

#### ***use TestInside 310-083 Q&A ensure you pass the exam at your first try.***

TestInside professional provide SCWCD 310-083 the newest Q&A, completely covers 310-083 test original topic. With our complete SCWCD resources, you will minimize your SCWCD cost and be ready to pass your 310-083 tests on Your First Try, 100% Money Back Guarantee included!

[SUN 310-083](#) Test belongs to one of the SCWCD certified test, if needs to obtain the SCWCD certificate, you also need to participate in other related test, the details you may visit the [SCWCD](#) certified topic, in there, you will see all related SCWCD certified subject of examination.

#### ***TestInside Testing Engine Features***

Comprehensive questions and answers about 310-083 exam

310-083 exam questions accompanied by exhibits

Verified Answers Researched by Industry Experts and almost 100% correct

310-083 exam questions updated on regular basis



14. `<sl:item name="Eggs" />`

15. `</sl:shoppingList>`

The tag handler for `sl:shoppingList` is `ShoppingListTag` and the tag handler for `sl:item` is `ItemSimpleTag`.

`ShoppingListTag` extends `BodyTagSupport` and `ItemSimpleTag` extends `SimpleTagSupport`.

Which is true?

A. `ItemSimpleTag` can find the enclosing instance of `ShoppingListTag` by calling `getParent()` and casting the result to `ShoppingListTag`.

B. `ShoppingListTag` can find the child instances of `ItemSimpleTag` by calling `super.getChildren()` and casting each to an `ItemSimpleTag`.

C. It is impossible for `ItemSimpleTag` and `ShoppingListTag` to find each other in a tag hierarchy because one is a Simple tag and the other is a Classic tag.

D. `ShoppingListTag` can find the child instances of `ItemSimpleTag` by calling `getChildren()` on the `PageContext` and casting each to an `ItemSimpleTag`.

E. `ItemSimpleTag` can find the enclosing instance of `ShoppingListTag` by calling `findAncestorWithClass()` on the `PageContext` and casting the result to `ShoppingListTag`.

Answer: A

6. Given the JSP code:

```
<% request.setAttribute("foo", "bar"); %>
```

and the Classic tag handler code:

```
5. public int doStartTag() throws JspException {
```

```
6. // insert code here
```

```
7. // return int
```

```
8. }
```

Assume there are no other "foo" attributes in the web application.

Which invocation on the `pageContext` object, inserted at line 6, assigns "bar" to the variable `x`?

A. `String x = (String) pageContext.getAttribute("foo");`

B. `String x = (String) pageContext.getRequestScope("foo");`

C. It is NOT possible to access the `pageContext` object from within `doStartTag`.

D. `String x = (String)`

```
pageContext.getRequest().getAttribute("foo");
```

E. `String x = (String) pageContext.getAttribute("foo",`

```
PageContext.ANY_SCOPE);
```

Answer: D

7. You are building a web application that will be used throughout the European Union; therefore, it has significant internationalization requirements. You have been tasked to create a custom tag that generates a message using the `java.text.MessageFormat` class. The tag will take the `resourceKey` attribute and a variable number of argument attributes with the format, `arg<N>`. Here is an example use of this tag and its output:

```
<t:message resourceKey='diskFileMsg' arg0='MyDisk' arg1='1247' />
```

generates:

The disk "MyDisk" contains 1247 file(s).

Which Simple tag class definition accomplishes this goal of handling a variable number of tag attributes?

A. `public class MessageTag extends SimpleTagSupport`

```
implements VariableAttributes {
```

```
private Map attributes = new HashMap();
```

```
public void setVariableAttribute(String uri,
```

```
String name, Object value) {
```

```
this.attributes.put(name, value);
```

```
}
```

```
// more tag handler methods
```

```
}
```

B. The Simple tag model does NOT support a variable number of attributes.

```

C. public class MessageTag extends SimpleTagSupport
implements DynamicAttributes {
private Map attributes = new HashMap();
public void putAttribute(String name, Object value) {
this.attributes.put(name, value);
}
// more tag handler methods
}

```

```

D. public class MessageTag extends SimpleTagSupport
implements VariableAttributes {
private Map attributes = new HashMap();
public void putAttribute(String name, Object value) {
this.attributes.put(name, value);
}
// more tag handler methods
}

```

```

E. public class MessageTag extends SimpleTagSupport
implements DynamicAttributes {
private Map attributes = new HashMap();
public void setDynamicAttribute(String uri, String name,
Object value) {
this.attributes.put(name, value);
}
// more tag handler methods
}

```

Answer: E

8. Click the Exhibit button.

The attribute "name" has a value of "Foo,"

What is the result if this tag handler's tag is invoked?

- A. Foo
- B. done
- C. Foodone
- D. An exception is thrown at runtime.
- E. No output is produced from this code.
- F. Compilation fails because of an error in this code.

Answer: A

9. <sl:item name="Bread" />

13. <sl:item name="Milk" />

14. <sl:item name="Eggs" />

15. </sl:shoppingList>

The tag handler for sl:shoppingList is ShoppingListTag and the tag handler for sl:item is ItemSimpleTag.

ShoppingListTag extends BodyTagSupport and ItemSimpleTag extends SimpleTagSupport.

Which is true?

- A. ItemSimpleTag can find the enclosing instance of ShoppingListTag by calling getParent() and casting the result to ShoppingListTag.
- B. ShoppingListTag can find the child instances of ItemSimpleTag by calling super.getChildren() and casting each to an ItemSimpleTag.
- C. It is impossible for ItemSimpleTag and ShoppingListTag to find each other in a tag hierarchy because one is a Simple tag and the other is a Classic tag.
- D. ShoppingListTag can find the child instances of ItemSimpleTag by calling getChildren() on the PageContext and casting each to an ItemSimpleTag.

E. ItemSimpleTag can find the enclosing instance of ShoppingListTag by calling findAncestorWithClass() on the PageContext and casting the result to ShoppingListTag.

Answer: A

10. To take advantage of the capabilities of modern browsers that use web standards, such as XHTML and CSS, your web application is being converted from simple JSP pages to JSP Document format. However, one of your JSPs, /scripts/screenFunctions.jsp, generates a JavaScript file. This file is included in several web forms to create screen-specific validation functions and are included in these pages with the following statement:

10. <head>

11. <script src='/scripts/screenFunctions.jsp'

12. language='javascript'

13. type='application/javascript'> </script>

14. </head>

15. <!-- body of the web form -->

Which JSP code snippet declares that this JSP Document is a JavaScript file?

A. <%@ page contentType='application/javascript' %>

B. <jsp:page contentType='application/javascript' />

C. <jsp:document contentType='application/javascript' />

D. <jsp:directive.page contentType='application/javascript' />

E. No declaration is needed because the web form XHTML page already declares the MIME type of the /scripts/screenFunctions.jsp file in the <script> tag.

Answer: D

11. Which implicit object is used in a JSP page to retrieve values associated with <context-param> entries in the deployment descriptor?

A. config

B. request

C. session

D. application

Answer: D

12. You have created a JSP that includes instance variables and a great deal of scriptlet code. Unfortunately, after extensive load testing, you have discovered several race conditions in your JSP scriptlet code. To fix these problems would require significant recoding, but you are already behind schedule. Which JSP code snippet can you use to resolve these concurrency problems?

A. <%@ page isThreadSafe='false' %>

B. <%@ implements SingleThreadModel %>

C. <%! implements SingleThreadModel %>

D. <%@ page useSingleThreadModel='true' %>

E. <%@ page implements='SingleThreadModel' %>

Answer: A

13. <sl:item name="Eggs" />

15. </sl:shoppingList>

The tag handler for sl:shoppingList is ShoppingListTag and the tag handler for sl:item is ItemSimpleTag.

ShoppingListTag extends BodyTagSupport and ItemSimpleTag extends SimpleTagSupport.

Which is true?

A. ItemSimpleTag can find the enclosing instance of ShoppingListTag by calling getParent() and casting the result to ShoppingListTag.

B. ShoppingListTag can find the child instances of ItemSimpleTag by calling super.getChildren() and casting each to an ItemSimpleTag.

C. It is impossible for ItemSimpleTag and ShoppingListTag to find each other in a tag hierarchy because one is a Simple tag and the other is a Classic tag.

D. ShoppingListTag can find the child instances of ItemSimpleTag by calling getChildren() on the PageContext and casting each to an ItemSimpleTag.

E. ItemSimpleTag can find the enclosing instance of ShoppingListTag by calling findAncestorWithClass() on the PageContext and casting the result to ShoppingListTag.

Answer: A

14. Given the definition of MyServlet:

```
11. public class MyServlet extends HttpServlet {  
12.     public void service(HttpServletRequest request,  
13.         HttpServletResponse response)  
14.         throws ServletException, IOException {  
15.         HttpSession session = request.getSession();  
16.         session.setAttribute("myAttribute", "myAttributeValue");  
17.         session.invalidate();  
18.         response.getWriter().println("value=" +  
19.             session.getAttribute("myAttribute"));  
20.     }  
21. }
```

What is the result when a request is sent to MyServlet?

- A. An IllegalStateException is thrown at runtime.
- B. An InvalidSessionException is thrown at runtime.
- C. The string "value=null" appears in the response stream.
- D. The string "value=myAttributeValue" appears in the response stream.

Answer: A

15. </sl:shoppingList>

The tag handler for sl:shoppingList is ShoppingListTag and the tag handler for sl:item is ItemSimpleTag.

ShoppingListTag extends BodyTagSupport and ItemSimpleTag extends SimpleTagSupport.

Which is true?

- A. ItemSimpleTag can find the enclosing instance of ShoppingListTag by calling getParent() and casting the result to ShoppingListTag.
- B. ShoppingListTag can find the child instances of ItemSimpleTag by calling super.getChildren() and casting each to an ItemSimpleTag.
- C. It is impossible for ItemSimpleTag and ShoppingListTag to find each other in a tag hierarchy because one is a Simple tag and the other is a Classic tag.
- D. ShoppingListTag can find the child instances of ItemSimpleTag by calling getChildren() on the PageContext and casting each to an ItemSimpleTag.
- E. ItemSimpleTag can find the enclosing instance of ShoppingListTag by calling findAncestorWithClass() on the PageContext and casting the result to ShoppingListTag.

Answer: A

```
16. session.getAttribute("myAttribute");
```

```
20. }
```

```
21. }
```

What is the result when a request is sent to MyServlet?

- A. An IllegalStateException is thrown at runtime.
- B. An InvalidSessionException is thrown at runtime.
- C. The string "value=null" appears in the response stream.
- D. The string "value=myAttributeValue" appears in the response stream.

Answer: A

17. A developer wants to make a name attribute available to all servlets associated with a particular user, across multiple requests from that user, from the same browser instance.

Which two provide this capability from within a tag handler? (Choose two.)

- A. `pageContext.setAttribute("name", theValue);`
- B. `pageContext.setAttribute("name", getSession());`
- C. `pageContext.getRequest().setAttribute("name", theValue);`
- D. `pageContext.getSession().setAttribute("name", theValue);`
- E. `pageContext.setAttribute("name", theValue, PageContext.PAGE_SCOPE);`
- F. `pageContext.setAttribute("name", theValue, PageContext.SESSION_SCOPE);`

Answer: DF

18. The `sl:shoppingList` and `sl:item` tags output a shopping list to the response and are used as follows:

- 11. `<sl:shoppingList>`
- 12. `<sl:item name="Bread" />`
- 13. `<sl:item name="Milk" />`
- 14. `<sl:item name="Eggs" />`
- 15. `</sl:shoppingList>`

The tag handler for `sl:shoppingList` is `ShoppingListTag` and the tag handler for `sl:item` is `ItemSimpleTag`.

`ShoppingListTag` extends `BodyTagSupport` and `ItemSimpleTag` extends `SimpleTagSupport`.

Which is true?

- A. `ItemSimpleTag` can find the enclosing instance of `ShoppingListTag` by calling `getParent()` and casting the result to `ShoppingListTag`.
- B. `ShoppingListTag` can find the child instances of `ItemSimpleTag` by calling `super.getChildren()` and casting each to an `ItemSimpleTag`.
- C. It is impossible for `ItemSimpleTag` and `ShoppingListTag` to find each other in a tag hierarchy because one is a Simple tag and the other is a Classic tag.
- D. `ShoppingListTag` can find the child instances of `ItemSimpleTag` by calling `getChildren()` on the `PageContext` and casting each to an `ItemSimpleTag`.
- E. `ItemSimpleTag` can find the enclosing instance of `ShoppingListTag` by calling `findAncestorWithClass()` on the `PageContext` and casting the result to `ShoppingListTag`.

Answer: A

19. Click the Exhibit button.

The resource requested by the `RequestDispatcher` is available and implemented by the `DestinationServlet`.

What is the result?

- A. An exception is thrown at runtime by `SourceServlet`.
- B. An exception is thrown at runtime by `DestinationServlet`.
- C. Only "hello from dest" appears in the response output stream.
- D. Both "hello from source" and "hello from dest" appear in the response output stream.

Answer: A

- 20. `<sl:shoppingList>`
- 12. `<sl:item name="Bread" />`
- 13. `<sl:item name="Milk" />`
- 14. `<sl:item name="Eggs" />`
- 15. `</sl:shoppingList>`

The tag handler for `sl:shoppingList` is `ShoppingListTag` and the tag handler for `sl:item` is `ItemSimpleTag`.

`ShoppingListTag` extends `BodyTagSupport` and `ItemSimpleTag` extends `SimpleTagSupport`.

Which is true?

- A. `ItemSimpleTag` can find the enclosing instance of `ShoppingListTag` by calling `getParent()` and casting the result to `ShoppingListTag`.
- B. `ShoppingListTag` can find the child instances of `ItemSimpleTag` by calling `super.getChildren()` and casting each to an `ItemSimpleTag`.

- C. It is impossible for ItemSimpleTag and ShoppingListTag to find each other in a tag hierarchy because one is a Simple tag and the other is a Classic tag.
- D. ShoppingListTag can find the child instances of ItemSimpleTag by calling getChildren() on the PageContext and casting each to an ItemSimpleTag.
- E. ItemSimpleTag can find the enclosing instance of ShoppingListTag by calling findAncestorWithClass() on the PageContext and casting the result to ShoppingListTag.

Answer: A

21. You have built a collection of custom tags for your web application. The TLD file is located in the file: /WEB-INF/myTags.xml. You refer to these tags in your JSPs using the symbolic name: myTags. Which deployment descriptor element must you use to make this link between the symbolic name and the TLD file name?

A. <taglib>

```
<name>myTags</name>
<location>/WEB-INF/myTags.xml</location>
</taglib>
```

B. <tags>

```
<name>myTags</name>
<location>/WEB-INF/myTags.xml</location>
</tags>
```

C. <tags>

```
<tags-uri>myTags</tags-uri>
<tags-location>/WEB-INF/myTags.xml</tags-location>
</tags>
```

D. <taglib>

```
<taglib-uri>myTags</taglib-uri>
<taglib-location>/WEB-INF/myTags.xml</taglib-location>
</taglib>
```

Answer: D

**[More 310-083 Information](#)**

**Related 310-083 Exams**

[310-083](#) Sun Certified Web Component Developer for J2EE 5

[310-081](#) Sun Certified Web Component Developer for J2EE 1.4

[310-084](#) Sun Certified Web Component Developer for Java. EE 5 Upgrade

[310-082](#) Sun Certified Web Component Developer for J2EE 1.4. Upgrade

**Other SUN Exams**

|                         |                         |                         |                         |                         |                         |                         |                         |
|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|
| <a href="#">310-066</a> | <a href="#">310-220</a> | <a href="#">310-014</a> | <a href="#">310-610</a> | <a href="#">310-345</a> | <a href="#">310-302</a> | <a href="#">310-052</a> | <a href="#">310-053</a> |
| <a href="#">310-400</a> | <a href="#">310-025</a> | <a href="#">310-016</a> | <a href="#">310-091</a> | <a href="#">310-044</a> | <a href="#">310-231</a> | <a href="#">212-811</a> | <a href="#">310-100</a> |
| <a href="#">310-202</a> | <a href="#">310-051</a> | <a href="#">310-013</a> | <a href="#">310-061</a> |                         |                         |                         |                         |