Tag Libraries

- ❑ Tags are used in a JSP page using the syntax `<prefix:tagname attribute="value">`any body content goes here`</prefix:tagname>`.
- ❑ The prefix used must match the value of the prefix attribute in the `taglib` directive.
- ❑ A `taglib` directive has the following syntax: `<%@ taglib prefix="prefix" uri="anyStringYouLike" %>`.

- ❑ The URI declared in the `taglib` directive (or namespace value) should match the value of a `<taglib-uri>` element in the deployment descriptor.
- ❑ The `<taglib-uri>` element is paired with a `<taglib-location>` element, which points to the location within the web application where the tag library descriptor file is located.
- ❑ `<taglib-uri>` and `<taglib-location>` are both subelements of `<taglib>`, which is a subelement of `<jsp-config>`, which is a subelement of the root element `<web-app>`.
- ❑ Apart from `<taglib>`, `<jsp-config>` can have one other type of subelement: `<jsp-property-group>`.
- ❑ Within `<jsp-property-group>`, you can define a group of JSP files with the `<url-pattern>` element.
- ❑ Also within `<jsp-property-group>`, EL can be turned off for the defined group of JSP files by setting the `<el-ignored>` element to a value of "true."
- ❑ Again within `<jsp-property-group>`, scripting can be turned off for the defined group of JSP files by setting the `<scripting-invalid>` element to a value of "true."
- ❑ A tag library descriptor file (TLD) has a root element of `<taglib>`.

- A TLD must have elements `<tlib-version>` and `<short-name>` defined.
- A TLD can have any number of `<tag>` elements.
- The `<name>` subelement of `<tag>` defines the (unique) name of the tag, as it is called in the JSP page.
- The `<tag-class>` subelement of `<tag>` defines the fully qualified name of the Java tag handler class.
- The `<body-content>` subelement of `<tag>` defines what is allowed to appear between the opening and closing tags:
  - empty: The tag mustn't have a body.
  - tagdependent: The tag can have a body, but the contents (scriptlets, EL, etc.) are completely ignored and treated like template text.
  - scriptless: The tag body can contain EL or template text, but no Java language scripting constructs (expressions, scriptlets, declarations)—if these are present, a translation error results.
  - JSP: The tag body can contain anything: Java language scripting, EL, or template text.
- Another subelement of `<tag>` is `<attribute>`: This can appear zero, one, or many times.
- The `<attribute>` element is used to define attributes on a tag and has three subelements:
  - `<name>`—a name for the attribute (unique within the tag)
  - `<required>`—true or false—whether the attribute must be present or is optional.
  - `<rtexprvalue>`—true or false—whether the attribute value can be provided by an expression (Java language or EL).

JSTL

- There are five JavaServer Page Standard Tag Libraries (JSTL): core, relational database access (SQL), Formatting with Internationalization, XML Processing, and EL standard functions. The exam focuses only on the core library.
- The actions (tags) within the libraries represent a standard way of performing frequently required functionality within web applications.

- ❑ The following tag directive is used for access to core library actions in your pages:

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

- ❑ The "c" value for `prefix` is usual, but optional.
- ❑ The value for `uri` must be just as shown above.
- ❑ There are fourteen core library actions, divided into four groups: general purpose, conditional, iteration, and URL-related.
- ❑ There are four actions in the general purpose group: `<c:out>`, `<c:set>`, `<c:remove>`, and `<c:catch>`.
- ❑ `<c:out>` is for directing output to the JSPWriter. Its attributes are *value* (the output), *default* (the output if *value* is **null**), and *escapeXml* (for converting XML-unfriendly characters).
- ❑ `<c:set>` is for setting attributes in any scope. Its main attributes are*value* (contents of attribute), *var* (name of attribute), and *scope* (scope of attribute).
- ❑ `<c:set>` also has *target* and *property* attributes for setting properties on beans.
- ❑ `<c:remove>` is for removing attributes in any scope. It has only the attributes *var* (name of attribute to remove) and *scope* (scope of attribute).
- ❑ `<c:catch>` catches Throwable objects thrown from the statements it contains. It has only the one optional attribute, *var*, to store the Throwable object for later use.
- ❑ The conditional group in the JSTL library has four actions: `<c:if>, <c:choose>, <c:when>,` and `<c:otherwise>`.
- ❑ `<c:if>` is used to conditionally execute some JSP statements if a test proves true. Its attributes are *test* (expression for the test), *var* (optional attribute variable to hold the result of the test), and *scope* (scope of the optional attribute).
- ❑ `<c:choose>` is used to contain mutually exclusive tests, held in `<c:when>` actions.
- ❑ `<c:choose>` can only contain `<c:when>` and `<c:otherwise>` actions (and white space).
- ❑ `<c:when>` has only one attribute: *test*.
- ❑ Only the statements bounded by the first `<c:when>`...`</c:when>` action whose test is true will be executed.

- ❑ One `<c:otherwise>` can be included after any `<c:when>` actions.
- ❑ The statements within `<c:otherwise>` are executed only if all the preceding `<c:when>` tests prove false.
- ❑ There are two actions in the iterator group of the JSTL core library: `<c:forEach>` and `<c:forTokens>`.
- ❑ `<c:forEach>` is used to iterate through a series of items in a collection, or simply to loop for a set number of times.
- ❑ `items` can hold Arrays, Strings, and most collection types in java.util: Collections, Maps, Iterators, and Enumerations.
- ❑ When iterating through collections, `<c:forEach>` uses the attributes *items* (for the collection object) and *var* (to represent each item in the collection on each circuit of the loop).
- ❑ When looping a set number of times, `<c:forEach>` uses the attributes *begin* (the number to begin at), *end* (the number to end at), and *step* (the amount to step by when working through from the begin number to the end number).
- ❑ In either case, a special variable called *varStatus* can be used to obtain properties about the current iteration.
- ❑ All the above attributes can be combined in a hybrid syntax—for example, to step through every second item in a collection.
- ❑ `<c:forTokens>` works similarly to `<c:forEach>`—but is specialized for breaking up (tokenizing) Strings.
- ❑ It has the same six attributes as `<c:forEach>`, and an additional seventh of its own.
- ❑ The *items* attribute will accept only a String as input.
- ❑ The additional seventh parameter is *delims*, which holds the characters used to denote where to break up the String.
- ❑ There are four actions in the URL-related group of the JSTL core library: `<c:import>`, `<c:url>`, `<c:redirect>`, and `<c:param>`.
- ❑ `<c:import>` is used to include a URL resource within the current page at run time.
- ❑ `<c:import>` has six attributes. The main one is *url* (the expression representing the URL resource to import).
- ❑ The *context* attribute can be used to specify a different context housed in the same application server.

❑ The *var* and *scope* attributes can be used to place the contents of the URL resource in a scoped attribute (as a String).

❑ Alternatively, *varReader* can be used to keep the contents of the URL resource in a Reader object.

❑ `<c:url>` is used to compose URL strings (for use as links in documents, for example).

❑ `<c:url>` has four attributes: *value* (expression for the URL string), *context* (optional alternative context on the same web application server), *var* (optional String attribute to hold the result of the URL String expression), and *scope* (scope for *var,* if used).

❑ `<c:redirect>` is used to instruct the web client to point to an alternative resource.

❑ `<c:redirect>` has two attributes: *url* (the URL for the client to point to) and *context* (optional alternative context if the URL is in a different web application on the same server).

❑ `<c:param>` can be nested within `<c:url>`, `<c:import>`, or `<c:redirect>`.

❑ `<c:param>` is used to attach additional parameters to the requests made or implied by the other URL-related actions.

❑ `<c:param>` has two attributes: *name* (the name of the request parameter) and *value* (the value of the request parameter).

EL Functions

❑ Any **public static** method in any Java class can be exposed as an EL function.

❑ EL functions are defined in `<function>` elements in a tag library descriptor (TLD).

❑ Within the `<function>` element, an EL function must have three subelements defined: `<name>`, `<function-class>`, and `<function-signature>`.

❑ `<name>` is a unique name for the function.

❑ `<name>` must be unique not only within functions in the TLD but also within other elements that might be defined in the TLD, such as custom tags and tag files.

❑ `<function-class>` gives the fully qualified name of the Java class containing the method backing the EL function.

- `<function-signature>` reflects the signature of the method backing the EL function.
- `<function-signature>` must always use fully qualified names for Java classes returned or passed in to the function (even String must be expressed as java.lang.String).
- Parameter names are omitted from the function signature (only types are defined).
- The method name in the function signature must match the method name in the Java class.
- Qualifiers (such as **public** and **static**) are omitted in the function signature. So the Java method with signature . . .

```
public static String getDefinition(String word, int timeForSearch)
```

- . . . would yield the following `<function-signature>`:

```
<function-signature>java.lang.String getDefinition(java.lang.String,
int)</function-signature>
```

- This function might be called in the JSP page with the following EL syntax:

```
${myfunctions:getDefinition(wordHeldInAttribute, 30)}
```