**The Tag File Model**

1. What are **tag files**?
    a. A tag file is a JSP page or document designed to be used as a custom action. Tag files are a variant of the simple tags written in the form of xml files with '.tag' extension. They contain a mixture of template text and elements. They are translated into Java source code and then compiled into a simple tag class. You don't write tag entries in a TLD file. Tag files are self-contained, with their own deployment directives.
    b. A tag file is a good solution if you are using the same JSP content in many places.
2. How is the body of the tag for a tag file processed?
    a. The body of the tag is parsed into a new JspFragment instance and setJspBody() is called, unless the tag is declared to have body content of empty or no body is present. Using the <jsp:doBody /> action in the tag file causes the JspFragment for the body to be invoked.
3. How does the JSP Container **find the Tag File**?
    a. The Tag File should have an extension of .tag and it should be located in one of these places within your web application:
        i. /WEB-INF/tags
        ii. A subdirectory of /WEB-INF/tags
        iii. In a JAR file kept in /WEB-INF/ lib. The directory of the tag file within the JAR file must be /META-INF/tags—or a subdirectory of /META-INF/tags.
    b. The tag directory (/WEBINF/tags) is referenced in the XML name space declaration in the JSP page. The name before the file extension .tag corresponds to the name as used in the JSP document.
4. How are Tag files related to tag handler classes?
    a. Tag files are used as sources from which a SimpleTag class is generated when the tag file is first used.
5. How are Tag files similar to JSPs and servlets?

a. Both are used as sources from which a class file is generated. For tag files, a SimpleTag class is created when the tag file is first used and the use of a JSP triggers generation and compilation of a servlet.

6. How is it different when a tag file sits inside a JAR file and kept directly in /WEB-INF/tags (or a subdirectory)?
    a. When a tag file sits inside a JAR file, the tag file must have a declaration inside a TLD like:
        i. <taglib>
        ii. <tlib-version>1.0</tlib-version>
        iii. <short-name>webcert</short-name>
        **iv. <tag-file>**
        v. **<name>**mytag</name>
        vi. **<path>**/META-INF/tags/mytag.tag</path>
        **vii. </tag-file>**
        viii. </taglib>

        You can mix tag file declarations with regular tag declarations and EL functions, all in the same TLD.
    b. For tag files kept directly in /WEB-INF/tags (or a subdirectory), there is no need for a TLD entry, although it is permitted.
7. What is the difference in namespace declaration within a JSP page when a tag is written as a tag file than through a class implementing SimpleTag?
    a. In the tag file approach, there's no need to have a namespace referencing a tag library. Instead, you must reference a tag directory—a directory that acts as a repository for tag files.
    b. Here is the original declaration in the jsp for a class implementing SimpleTag:
        **i. <html xmlns:mytags="http://www.abc.com/taglibs/mytags" ... >**
            1. Here, the URL String "http://www.abc.com/taglibs/mytags" doesn't point to anything—it just has to match the corresponding <taglib-uri> setting in the web deployment descriptor.
    c. Here are the declarations when a tag file is used:
        **i. <html xmlns:mytags="urn:jsptagdir:/WEB-INF/tags/" >**

1. Here, the URN (of type JSP tag directory) designates a real location within the web application: the directory /WEB-INF/tags.

**ii. <html xmlns:mytags="urn:jsptld:/WEB-INF/tag1.tld" >**

1. Here tag1.tld is the name of an explicit TLD.

8. Give the traditional JSP declaration instead of the namespace syntax for importing a tag file in a JSP page?
    a. <% taglib prefix="mytags" tagdir="/WEB-INF/tags" %>
    b. You can provide an uri attribute if you have created an explicit TLD. The **tagdir** and **uri** can't coexist in the same <% **taglib** %> directive. The **prefix** is always present, and you can have either **tagdir** or **uri**—not both.
9. If the tagdir attribute on the taglib directive doesn't begin with /WEB-INF/tags or if it doesn't not point to a valid directory, what will happen?
    a. A translation error will occur.
10. What are the fallback mechanism the container uses if the URI used in a JSP taglib directive does not map to any entries in the taglib map?
    a. If it is an absolute url, a translationm error occur.
    b. If it is a relative URL, relative to the current location or the context, it is assumed to point to the location of the TLD.
    *c. It is a good practice not to rely on the fallback mechanism for URI resolution for TLDs.*
11. List down two **directives** that work identically in tag files and in JSP pages?
    a. <%@ taglib %>
    b. <%@ include %>
12. List down the **directives** that are only for tag files?
    a. **<%@ tag %>**, which shares several attributes in common with <%@ page %> for JSPs (language, import, pageEncoding, isELIgnored).
    b. **<%@ attribute %>**, like the <attribute> subelement of <tag> in a TLD
    c. **<%@ variable %>**, like the <variable> subelement of <tag> in a TLD
13. What are the allowed values for the body-content attribute for a tag file? What is the default value for it?
    a. Allowed values are scriptless, empty and tagdependent. Default is scriptless.

14. Can we write tag files in pure XML, as JSP documents?
    a. Yes. There are custom actions equivalents, such as <jsp:directive.tag ...>.
15. Can we have the <%@ page %> directive in a tag file?
    a. No.
16. List down the attributes of the **tag directive**?
    a. Tag attributes are:
        i. display-name
        ii. body-content
        iii. dynamic-attributes
        iv. small-icon
        v. large-icon
        vi. descripttion
        vii. example
        viii. language
        ix. import
        x. pageEncoding
        xi. isELIgnored
17. List down the attributes of the **attribute directive**?
    a. Attributes of the attribute directive are:
        i. name
        ii. required
        iii. fragment
        iv. rtexprvalue
        v. type
        vi. description
18. What is a **JspContextWrapper**?
    a. The JSP Context Wrapper is a JspContext created and maintained by a tag handler implementation. It wraps the Invoking JSP Context, that is, the JspContext instance passed to the tag handler by the invoking page via setJspContext().
    b. The container passes the tag file the JspContext instance it passes to every tag handler using the setJspContext(). The tag file keeps a copy of the original

JspContext, but creates a wrapper of the context called Jsp Context Wrapper which is of type JspContextWrapper. The getJspContext() method always return this wrapper. Most of the methods of the wrapper delegates to the original invoking Jsp Context, except those relating to page-scoped attributes. The wrapper provides local storage for page-scoped attributes, distinct from those in the invoking JSP context.

19. List down the attributes of the **variable directive**?
    a. Attributes are:
        i. **name-given –** the name of both the invoking JSP context's page scoped attribute and the JSP wrapper context's page scoped attribute local to the tag file whose value is to be synchronized.
        ii. **name-from-attribute** – declares the name of a scoped attribute to be defined in the JSP currently invoking this tag. The alias must also be declared with this attribute. Name used as a value of name-from-attribute attribute of the variable directive should already be declared during translation time and must also be declared using the attribute directive which must also have the following attribute set:
            1. required=true
            2. rtexprvalue=false
            3. type="java.lang.String"
        iii. **alias** – name of a locally scoped attribute to hold the value of this variable. Container will synchronize this value with the variable whose name is given by name-from-attribute. Should be used only along with name-from-attribute.
        iv. **variable-class** – fully qualified name of class for this variable. Default is java.lang.String.
        v. **declare** – if true container will create this variable in the calling JSP, else the variable should already be existing.
        vi. **scope** – AT_BEGIN, AT_END or NESTED.
        **vii. Description**

b. All attributes of the variable directive are optional except that either name-given or both the name-form-attribute and alias attributes must be declared.

20. What are the alternatives of variable and attribute directive in an explicit TLD for configuring tag files?
    a. Configuration of variables and attributes can only be declared in the tag file, using the variable and attribute directives respectively. There are no <variable> or <attribute> elements in an explicit TLD for configuring tag files.
21. List down the **implicit objects available for a tag file**?
    a. The implicit objects available are:
        i. request
        ii. response
        iii. jspContext
        iv. session
        v. application
        vi. out
        vii. config
    b. The JSP implicit objects page and exception are not available. jspContext replaces the servlet environment specific pageContext. jspContext is more generic and is the JSP context wrapper instance.
22. Give the use of **<jsp:doBody/>** standard action?
    a. Used to invoke the body and write its contents to the out stream. Same as calling invoke(null) on the JspFragment.
    b. To override the default behaviour of writing to the out stream we can use **either** the **varReader** attribute to get the evaluated contents in a Reader or the **var** attribute to get the contents as a String.
    c. We can also specify an optional attribute scope to define the scope of the attribute defined by **varReader** or **var**.
        i. <jsp:doBody var=”myVar” scope=”request”/>
23. Give the use of <jsp:invoke/> action with an example?
    a. <jsp:invoke/> is used to invoke attribute fragments and write to out stream like:
        i. <jsp:invoke fragment=”attributeName”/>

b. To override the default behaviour of writing to the out stream we can use **either** the **varReader** attribute to get the evaluated contents in a Reader or the **var** attribute to get the contents as a String.

c. We can also specify an optional attribute scope to define the scope of the attribute defined by **varReader** or **var**.

24. What is a **taglib map**?

a. Each TLD is associated with a unique URI. This mapping is either found in the TLD itself or in the DD. When the container loads the application it gethers together all the TLD files it can find and maintain a mapping between TLD files and URIs. This map is known as the taglib map.

25. How does the container generates the taglib map?

a. The container generates the taglib map through the following rules in order:

i. The taglib map in the web.xml is examined for TLD-to-URI mappings which are made using the <taglib> elements in the web.xml.

ii. All TLDs are examined for implicit map entries which are made using the <uri> element of the TLD. The container searches both packged jars under /META-INF/ and all TLDs under /WEB-INF/ in the application.

iii. Implicit map entries from the container are added to the taglib map. These are installed by JavaEE server and reference common libraries like JSTL.

26. What is **Variable synchronization** for tag files?

a. Variable synchronization for tag files is the mechanism through which container copies variables from JSP context wrapper's page scope to original invoking JSP context's page scope. ie the container updates the calling pages attributes with the local attribute in the tag file. If it does not exist in the tag file, it is removed from the calling page's context. Variable synchronization for tag files copies one page scope to another.

b. Synchronization always occur before any fragment is executed and never after a fragments execution.

c. The container guarantees that all synchronization variables of scope AT_BEGIN and NESTED will be synchronized just before the invocation of <jsp:doBody/>

and <jsp:invoke/>. This allows variables to be used in the body of the action and its attribute values.

27. How is variable synchronization of tag files different from that of classic tags and raw SimpleTag implementations?
    a. Variable synchronization for tag files copies one page scope to another whereas synchronization for classic tags involves updating JSP scripting variables.
    b. Raw SimpleTag implementations only ever access one page scope and cannot contain JSP scripting elements. Hence variable synchronization is unavailable for them.
28. What is the condition for web container to can generate an implicit TLD file for a tag library?
    a. All tag files with .tag or .tagx extension and deployed in the /WEB-INF/tags directory are assigned an implicit TLD by the container. An implicit TLD is created by the container for each subdirectory, and each subdirectory therefore represents a distinct tag library.
    b. The web container can generate an implicit TLD file for a tag library comprised of only tag files.
29. Can the web container generate an implicit TLD file for a tag library comprised of both simple tag handlers and tag files?
    a. No
30. For tag files kept directly in /WEB-INF/tags (or a subdirectory) can you have a TLD? Is it necessary? Give any reason you might do so.
    a. For tag files kept directly in /WEB-INF/tags (or a subdirectory) you don't have to provide a TLD as the web container will generate an implicit TLD file. However you can have a TLD entry if (for some reason) you wanted a tag name that is different from the tag file name.
31. Because there is no TLD entry to refer to, what all extra information a tag file has to provide?
    a. Because there is no TLD entry to refer to, the tag file has to assume its own responsibility for providing some of the information.

i. We should declare attributes using the **<%@ attribute %>** directive. The only mandatory attribute for this directive is name. This is entirely equivalent to a TLD file containing: **<attribute>** <name>begin</name> **</attribute>.** <% attribute %> has other attributes, and all of them (required, rtexprvalue, description, type, and fragment) are equivalent to the TLD subelements of <attribute>.

ii. The tag file needs to find its own way of substituting the Java logic otherwise placed in the doTag() method of a genuine simple tag handler class. Actual Java logic is disallowed here and you normally use EL, other custom actions or actions from the JSTL.

iii. Example:

```
1. <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2. <%@ taglib prefix="mytags" uri="http://www.abc.com/taglibs/mytags" %>
3. <%@ attribute name="begin" %>
4. <%@ attribute name="end" %>
5. <table border="1">
6. <c:forEach begin="${begin}" end="${end}" step="1" varStatus="counter">
7. <tr>
8. <td>${counter.index}</td>
9. <td>${mytags:unicodeConverter(counter.index)}</td>
10. </tr>
11. </c:forEach>
12. </table>
```

32. How do we retrieve attribute values in a tag file?
    a. No setter methods are available for attributes in the tag file approach compared to the SimpleTag approach.
    b. In the tag file, we retrieve the attribute value for an arbitrary attribute using:
        i. getJspContext().getAttribute("myAtr")
33. Which all directives used in normal JSPs can be used in the tag files?

a. Aside from the page directive all directives used in normal JSPs (include and taglib ) can be used in the tag files.

34. What will be the contents of the page scope of tag files when invoked?
    a. Tag files will always see an empty page scope when invoked unless attributes are declared.
35. How are the attributes of tags for tag files treated when invoked?
    a. For each attribute declared in the actions opening tag, a new page scoped attribute is created, with the same name as the declared attributes, and with the attributes evaluated value. These attributes are never visible outside the tag file as these attributes are added to the JSP context wrapper and not to the original invoking JSP context.
    b. For attributes declared as <jsp:attribute> child elements, these attributes are added to the tag-local page scope. If the type of the attribute is JspFragment, the container parses the body of the <jsp:attribute> element into a JspFragment instance. These fragments can be invoked in the tag file using <jsp:invoke/>
36. From a tag file how can we get access to the calling JSPs page scope contents?
    a. It is not possible. From a tag file we can gain direct access to any scopes other than page through the JSP context wrapper. The page scope is completely isolated from the calling JSP or the parent action and it is impossible to gain access to the calling JSPs page scope contents. This avoids naming conflicts.
37. Describe the use of <jsp:body> element inside tag files?
    a. The container evaluates the body of the action, either as a normal body content or as the content of a <jsp:body> element. Normally, the body of a standard or custom action invocation is defined implicitly as the body of the XML element used to represent the invocation.
    b. The body is defined explicitly using the jsp:body standard action if one or more jsp:attribute elements appear in the body of the tag.
        i. <jsp:element name="${content.headerName}"
        ii. xmlns:jsp="http://java.sun.com/JSP/Page">
        iii. <jsp:attribute name="lang">${content.lang}</jsp:attribute>
        iv. <jsp:body>${content.body}</jsp:body>

v. </jsp:element>

c. It is legal to use the jsp:body standard action to supply bodies to any standard actions that accepts a body (except for jsp:body, jsp:attribute, jsp:scriptlet, jsp:expression, and jsp:declaration).
d. The body standard action accepts no attributes.
e. If one or more jsp:attribute elements appear in the body of a tag invocation but no jsp:body element appears or an empty jsp:body element appears, it is the equivalent of the tag having an empty body.

38. Can a tag file make use of other tags in its implementation?
    a. A tag file may make use of other tags in its implementation.
    b. Classic tags implementing Tag relies on the PageContext object which is part of the servlet model, while SimpleTag uses the JspContext which is servlet independent. If the SimpleTag is running on a non servlet platform, then the classic tags won't be available for use and will throw an exception. However most of the JSP web servers are JavaEE compliant and hence most JSP platforms will be servlet based and support classic tags.
39. What all locations are allowed for a tag file inside a web application?
    a. he container will automatically find a tag file if it has an extension of .tag and that it is located in one of these places within your web application:
        i. /WEB-INF/tags
        ii. A subdirectory of /WEB-INF/tags
        iii. In a JAR file kept in /WEB-INF/ lib. The directory of the tag file within the JAR file must be /META-INF/tags—or a subdirectory of /META-INF/tags.
40. Can we have a TLD for a tag file? Describe the use of TLD for a tag file?
    a. When tag files are deployed directly into the WAR file in /WEB-INF/tags (or a subdirectory), no TLD is required, although it is permitted. A TLD must accompany all tag handler classes and tag files in the JAR. Any tags or tag files which are not declared in the TLD will not be recognized or loaded by the container.

b. You can mix tag file declarations with regular tag declarations and EL functions, all in the same TLD. The <name> component must be unique across all the different types.
c. In the simple tag file declaration within the TLD, the <body-content> is restricted to three allowed values: empty, tag dependent, and scriptless instead of four for custom model which also includes JSP.
d. Compared to SimpleTag model, you don't write tag entries in a tag library descriptor (TLD). Tag files are self-contained, with their own deployment directives.

41. How is Tag directive similar to page directive?
    a. Tag directive is similar to page directive that more than one tag directive may be declared, but only one occurrence of any given attribute may be declared, unless each occurrence of that attribute has the same value. The import and pageEncoding attributes are exceptions to this rule and may occur multiple times.

**42. More Notes**

**a.** When your JSP page source invokes the tag from a **tag file:**
    i. Your JSP page source invokes the tag, which maps to a tag file in a tag directory.
    ii. If this is its first use, the container generates a Java source code file that implements SimpleTag.
    iii. This source is kept and compiled in a location known to the server. Thereafter, the server keeps an internal mapping between invocations of the tag file and the actual compiled class.
    iv. Beyond that, everything works just as the simple tag declared in a tag handler.

b. Within the JSP document, in the tag file approach, elements for presentation and data have migrated to the tag file and it no longer has a body with EL variables.