Simple Custom Tags

1. Explain **the life cycle** of a "Simple" Custom Tag Event Model?
    a. Various tag events while processing an occurrence of the simple tag are:
        i. **Construction -** The JSP container makes a new instance of the tag handler class for every occurrence of the simple tag. The container calls the zero-argument constructor.
        ii. **setJspContext (JspContext pc)** - Saves the JspContext object for later access.
        iii. **setParent (JspTag tag) -** Saves the immediate parent as a JspTag. This method is called only if the custom action has a custom action as a parent.
        iv. **set<AttributeName>(<Type> attributeValue) -** The JSP container calls any "set" methods for attribute names defined in the TLD.
        v. **setJspBody(JspFragment jspBody)** - The JSP container calls this method only if the custom action has a body and passes the **JspFragment** which we can save to process the body later. The JSP container doesn't process that for us automatically.
        vi. **doTag()** - Within this method, you can do whatever you like:
            1. Process the body (more than once if required)
            2. Write directly to page output
        vii. **Variable Synchronization** - The variable synchronization process creates attributes accessible in your page after the end of your tag.
        viii. **Garbage Collection** - There is no release() method for simple tags. So any cleanup must happen in doTag() or a method called from doTag().
2. In a "Simple" Custom Tag Event Model any tag handler class needs to implement the …….. **interface**.
    a. javax.servlet.jsp.tagext.**SimpleTag**
3. Are there any classes that already **implement the SimpleTag** interface?
    a. SimpleTagSupport. We can extend and use it.
4. Give two methods that are present in the SimpleTagSupport which are not present in the SimpleTag interface?
    a. getJspBody()

b. getJspContext()

5. What does the **getJspBody()** of SimpleTagSupport class do?
    a. It returns the body fragment of type JspFragment.
6. What is the effect of calling invoke() on JspFragment defining the body of the custom action?
    a. The fragments are not directly written to the response, but to a JspFragment instance, which acts like a buffer.
    b. The effect of calling invoke() on JspFragment defining the body of the custom action, is to process this piece of JSP page source. Template text is written directly to page output. Any EL expressions (such as ${variable}) are evaluated before being sent to page output.
7. What will happen if we call **invoke(null)** on the JspFragment object returned by the getJspBody()?
    a. We can call invoke(java.io.Writer out) on the JspFragment object and can define our own Writer and divert the JspFragment output there.
    b. But by supplying null as the parameter value, we are writing to the JspWriter associated with the JspContext. In other words, getJspBody().invoke(null) is shorthand for: getJspBody().invoke(getJspContext().getOut()).
8. What does the below code snippet mean in a tag handler code:
        i. JspFragment fragment = getJspBody();
        ii. fragment.invoke(null);
    a. The code gets the JspFragment defining the body of the custom action.
    b. The effect of calling invoke() is to process this piece of JSP page source. The parameter passed into the invoke method determines the writer to be used. By supplying null as the parameter value, you are writing to the JspWriter associated with the JspContext.
9. Write code snippet to set an attribute named number in page scope from within the tag handler doTag() method?
    a. Code snippet:
        i. JspContext ctx = getJspContext();
        ii. ctx.setAttribute("number", new Integer(123));

10. Give the equivalents for PageContext and BodyContent in the simple tag model?
    a. JspContext and JspFragment
11. How are javax.servlet.jsp.**JspContext** and javax.servlet.jsp.**PageContext** related?
    a. PageContext extends from JspContext.
12. Give few important methods added by PageContext in addition to those inherited from JspContext?
    a. PageContext adds methods to do with
        i. Accessing implicit objects in a servlet environment (e.g.,getRequest(), getResponse(), getServletContext()).
        ii. Redirection (forward(), include()).
13. What will happen with the following code:

    HttpServletRequest request = myJspContext.getRequest();

    where myJspContext is an object of type JspContext?
    a. Won't compile. getRequest() is not part of JspContext, but PageContext.
14. Give the differences between javax.servlet.jsp.tagext.**BodyContent** and javax.servlet.jsp.tagext.**JspFragment**?
    a. **BodyContent** inherits from javax.servlet.jsp.**JspWriter**, which is a java.io.**Writer**. **JspFragment** isn't a Writer; it inherits directly from java.lang.Object.
    b. **BodyContent** has some content in it already when your classic custom tag handler code gets hold of it. This is the result of the JSP container evaluating the body of the tag. **JspFragment** content constitutes the body before any evaluation has taken place. In your simple tag handler code, you control when to do the evaluation by calling the invoke() method on the JspFragment object.
    c. There is no concept of buffering with **JspFragment**, as there is with **BodyContent**. Nothing of the body is buffered because nothing has been output until you decide.
15. After SimpleTag.doTag() method, how can you skip the remaining of the page?
    a. We need to throw javax.servlet.jsp.**SkipPageException**, a subclass of javax.servlet.jsp.**JspException**.

16. Why are Simple tag handlers are forbidden to have JSP scripting element code? Give any reason.
    a. Simple tag handlers are forbidden to have JSP scripting element code, mainly because using scripting elements in bodies require synchronization between the scripting elements in the current JSP and the body of the current action being invoked.
17. Can Simple tag handlers accept JSP scripting elements as attribute values?
    a. Simple tag handlers accept JSP scripting elements as attribute values, provided that the attribute accepts a runtime expression and is not of data type JspFragment ie, <**rtexprvalue>** in TLD is true and <fragment> has a value of false or is omitted from the TLD.
18. What are the allowed values for the <body-content> attribute for a simple tag file declaration within the TLD?
    a. 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. The fourth value—JSP—may get through XML schema validation, but the JSP container will give you a translation error.
    b. Simple tags are not allowed the full range of JSP syntax. Java language syntax within scriptlets, declarations, or expressions is disallowed.
19. In a "Simple" Tag Model, ……. is the last method called by the container before garbage collection.
    a. **doTag**()
20. List down the important methods of JspContext?
    a. JspContext contains all the methods (inherited by PageContext as well) to do with:
        i. Attribute access (e.g., getAttribute(), setAttribute())
        ii. Writer access (getOut())
        iii. Programmatic access to the EL evaluator (getExpressionEvaluator(), getVariableResolver())

21. When using simple tags, the container may translate certain parts of the document into fragments. What is a fragment and what is the type of a fragment? Which all parts of a simple tag are converted to fragments?
    a. When using simple tags, the container may translate certain parts of the document into fragments. Each of these pieces extends the javax.servlet.jsp.tagext.JspFragment class.
    b. A fragment represents the unevaluated form of contents. It is illegal to declare a fragment attribute using normal attribute syntax in the opening tag of the action; all fragment attributes must be declared as <jsp:attribute>s. Each fragment may contain only template text, EL expression and nested actions, but never JSP scripting elements.
    c. The following parts of a document are converted into JspFragments:
        i. The bodies of simple tags
        ii. Attributes declared using <jsp:attribute> and configured in the TLD with a <fragment> having value true.
22. List down the abstract methods of JspFragment abstract class?
    a. The JspFragment abstract class has two abstract methods:
    b. JspContext getJspContext()
    c. Void invoke(Writer w)
23. …….. object in a simple tag can be used for access to attributes in all scopes and the JspWriter currently associated with the page.
    a. JspContext
24. **More Notes on simple tags**
    a. In the **SimpleTagSupport**, doTag() is a do-nothing implementation.
    b. Both PageContext and JspContext are classes, not interfaces.