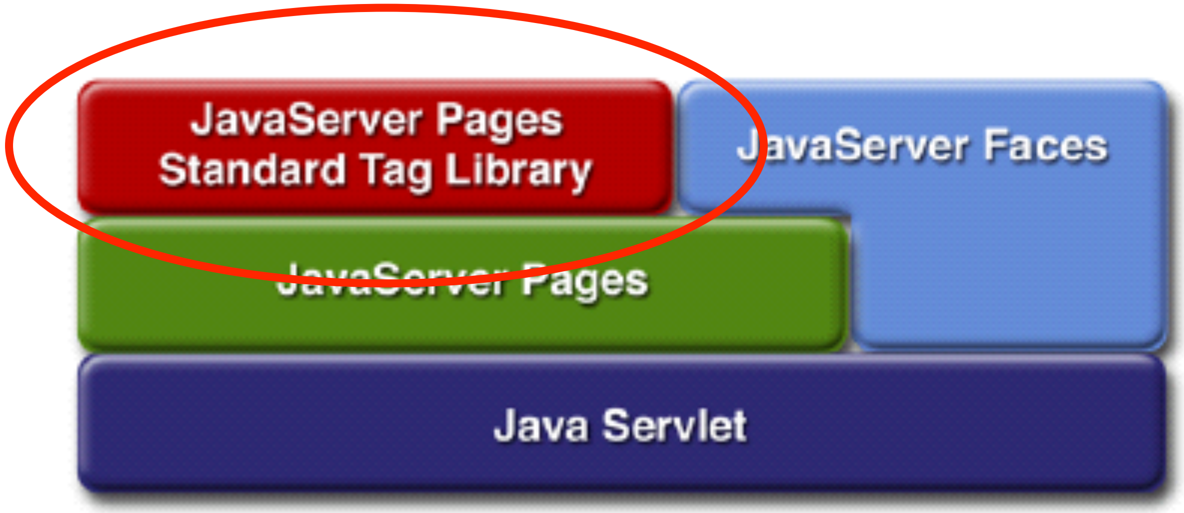
JSTL:
Java Standard Tag Library

Java Web Application Technologies



NOTE: all are based on Java Servlet

How to implement dynamic content in JSP?

- Three ways in JSP technology
  - Scriptlets
    - ***Easy, rapid*** for Java programmers
    - ***No reusability***
    - ***Difficult*** for Web designers: mixes HTML with Java code
  - JavaBeans / Entreprise Java Beans
    - ***Easy*** for Java programmers; object-oriented, component-oriented
    - ***Little re-usability***
    - ***Still Difficult*** for Web designers: better distinctions between HTML and Java, but still it can be scaring for non-programmers
  - JSP tag extensions

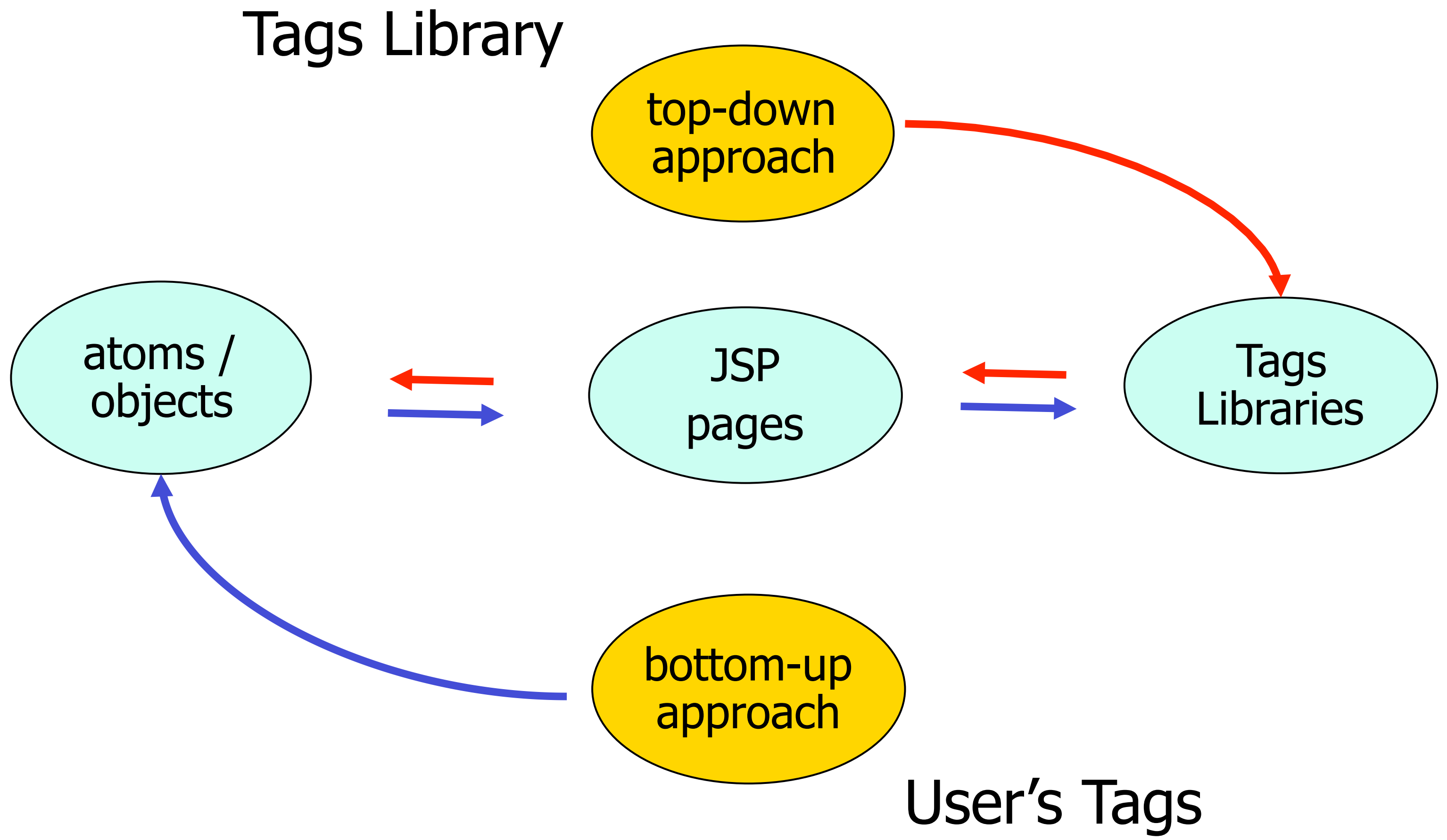
JSP tag extensions

- if you do not want Java in your JSPs but you still want to make presentation decisions based on some logic, you need a technology as easily comprehensible as HTML that is still capable of invoking the power of Java where necessary.
- This is why *custom tags* were created. Custom tags are HTML-like tags that you develop using Java and that are meant to provide some additional functionality.
  - If we take HTML tags as an example, to create a new paragraph we use the <p> tag that HTML provides [<p>...</p>]. The reason that the text is displayed in a new paragraph is that the browser sees and understands these tags.
  - Similarly, if we were to use the <blue>...</blue> tag to display all numeric content within it in blue, would it work? Of course not, as the browser has no idea what the <blue> tag stands for.
  - But we can provide the <blue> tag implementation we want ourselves, using the power of Java.
  - To take this example a step further, what we can do is tell the Web designers responsible for the presentation that whenever they encounter a case in which they want all numeric data in the text to be displayed in blue, they can use the <blue> tag [<blue></blue>].
  - If we weren't able to use this tag, we would have had to embed Java right in the JSP page.

Advantages of using custom tags

- **Simplicity and ease of use** — Custom tags are very simple for even a nonprogrammer to use and using them makes the JSP page a lot easier to comprehend
- **Code reusability** — Once you have the functionality in place as a custom tag, that code never has to repeat. The same piece of code can be reused over and over
- **Ease of maintenance** — Any change you make to the core tag logic is reflected across all pages in which the tag is used. This can be a lifesaver. You do not have to touch any of the individual JSPs in which that particular tag was used.

Approaches

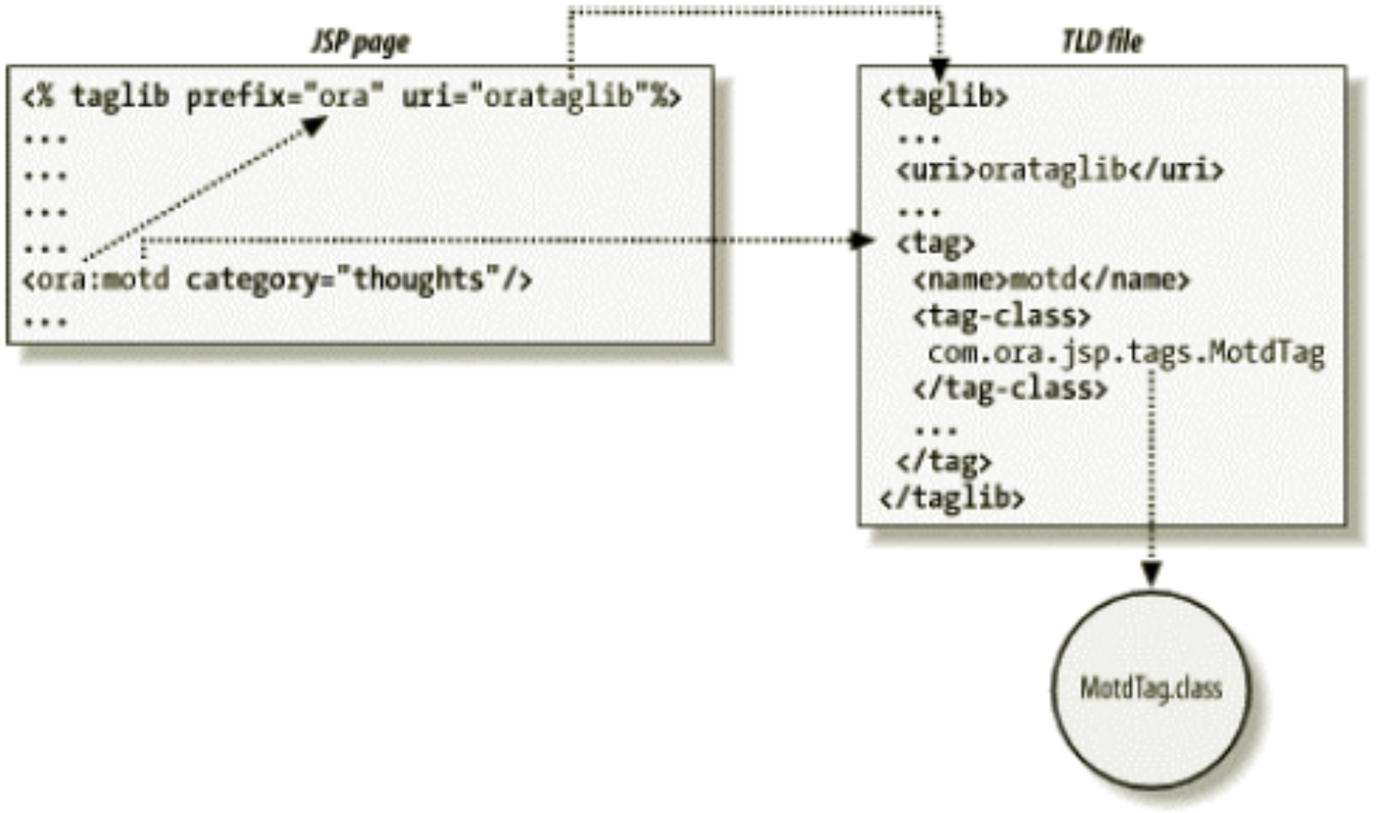

Tags Library
top-down approach
atoms / objects
JSP pages
Tags Libraries
bottom-up approach
User's Tags

Custom-Tag Implementation Framework

JSTL components:

- **Tag library**— A collection of tags. Most tags tend to be packaged as part of libraries containing many tags that might work independently or in coordination with each other. Being reusable components, tag libraries are generally distributed as JAR files.
- **Tag-library descriptor**—This is an XML file with .tld extension. This file contains the syntax of the tags and the instructions that the container requires in order to manage those tags.
- **Tag handler**—This is the Java class that contains the logic to perform the expected action for the tag. It is a container-managed object. The class has to implement one of the specified Java interfaces if it is to be treated as a valid tag handler by the container.

Relations between tag components



```
<% taglib prefix="ora" uri="orataglib"%>
...
...
...
...
<ora:motd category="thoughts"/>
...
```

```
<taglib>
 ...
 <uri>orataglib</uri>
 ...
 <tag>
  <name>motd</name>
  <tag-class>
   com.ora.jsp.tags.MotdTag
  </tag-class>
  ...
 </tag>
</taglib>
```

Working with the JSTL

- When custom tags first arrived on the scene they were heralded as an important step in simplifying JSP development.
  - However, most developers found developing custom tags somewhat difficult. JSPs' support for scriptlets also led many developers to take the easy way out and go back to using scriptlets and embedding Java code in their JSPs. As a result, developing custom tags for specific project requirements never really caught on as much as it was expected to.
  - However, custom tags that were meant to achieve basic functionality that was common across companies and projects were rapidly adopted. Look around the Web and you can find freely available tags that do specific tasks very well. It was a wise move to come up with a new specification for these common tags
  - JSTL is just that. It is a specification for developing custom tags that perform common tasks like formatting dates, parsing XML, and iterating through collections.

Example of JSTL tag: NumberFormat.jsp

taglib directive that tells the container, which unique (URI) identifies this tag and the prefix that you would be using in the page

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

<html>
<body>
<fmt:formatNumber value="00099765.4355"
                  type="currency" currencySymbol="$"
                  maxFractionDigits="2"/>
</body>
</html>
```

NumberFormat.jsp on execution displays $99,765.44 as the output.

whenever the container finds in this JSP a tag with the prefix 'fmt', it will look for the associated tag-library descriptor (URI) to check the exact usage of the tag and the tag handler responsible for actually processing the tag and performing the required action.

JSTL common prefixes

- **Core library**—This library consists of the tags that are used most often. These are tags that iterate through collections, perform if-logic, write output to the page, and so on.
- **XML library**—This library provides tags to parse XML, perform XSL transformations, and perform similar actions on your XML.
- **Formatting library**—This library is most useful for internationalization and performing date and number transformation.
- **SQL library**—This library consists of tags to connect to the database, fire SQL queries, and perform other database-related activities.

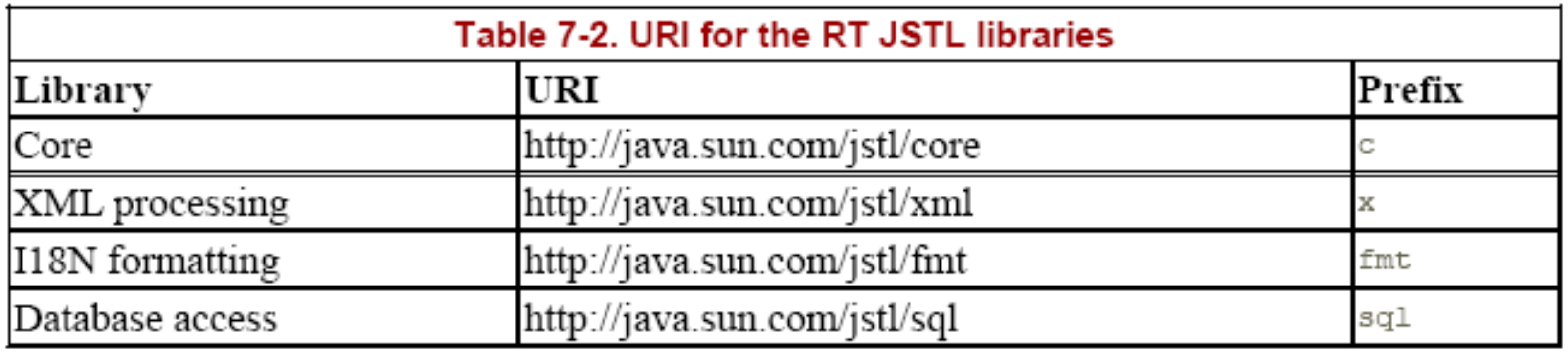


Table 7-2. URI for the RT JSTL libraries

Library	URI	Prefix
Core	http://java.sun.com/jstl/core	c
XML processing	http://java.sun.com/jstl/xml	x
I18N formatting	http://java.sun.com/jstl/fmt	fmt
Database access	http://java.sun.com/jstl/sql	sql

Links

- If you want to read the JSTL specification, check out http://java.sun.com/products/jstl.

- Apache Taglibs is the reference implementation for this specification and can be downloaded at http://jakarta.apache.org.

- Here you can also find tag libraries for string manipulation, working with dates, and so on that are not part of the specification but that could come in handy.

Importing a tag library

The JSP standard syntax

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

- All tags for this library that are used in this JSP page will be in the format **`<fmt:XXX>`**
- Restriction on special names: jsp, jsp, java, javax, servlet,sun, sunw

The XML JSP syntax

- While writing a JSP page using the XML syntax, taglib does not have a corresponding tag form, unlike other directives, such as page and include.
- To achieve the same purpose served by the previous taglib we would place the code in this format:

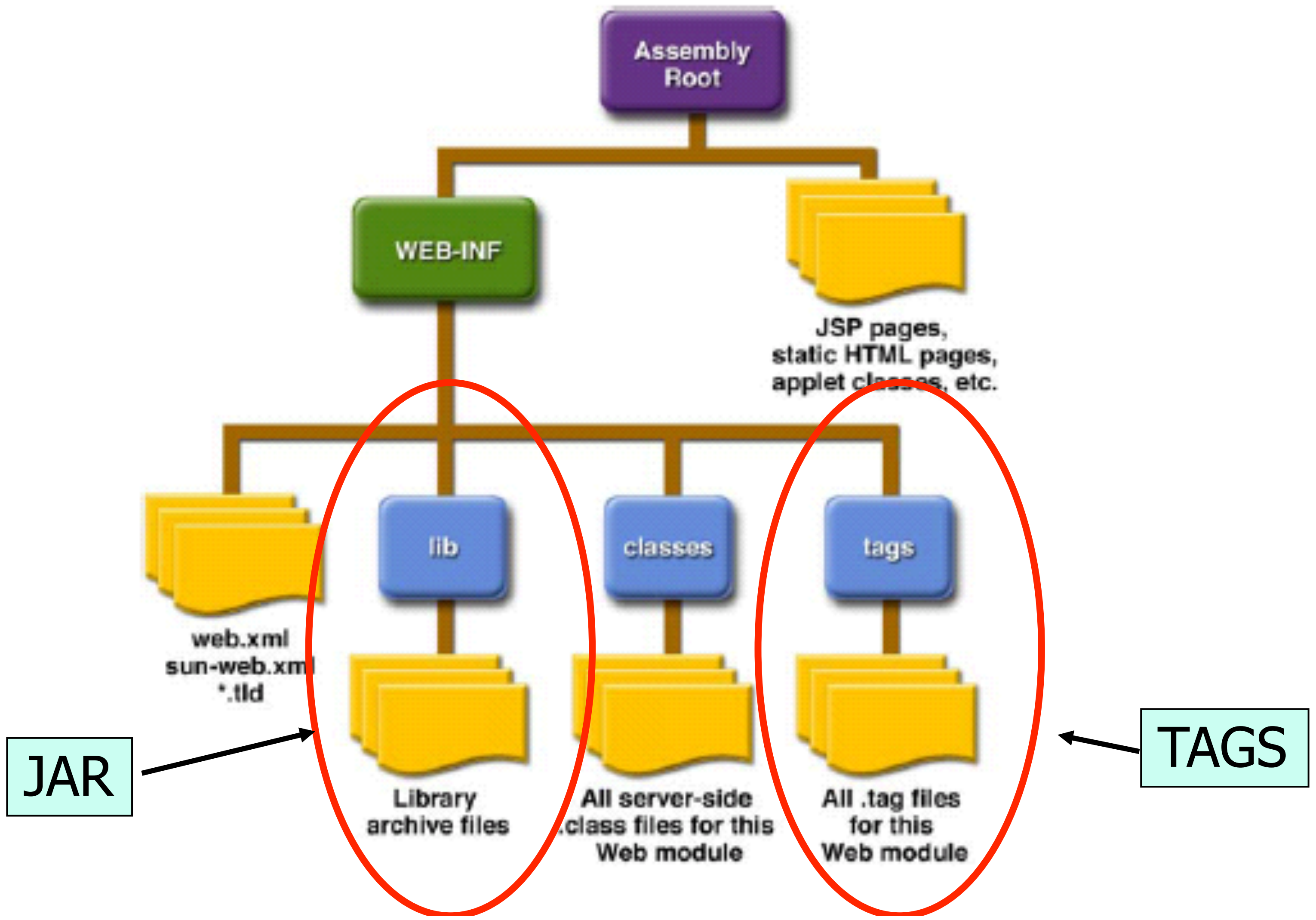
```
<jsp:root xmlns:fmt=http://java.sun.com/jstl/fmt
        version="2.0">
```

- Until JSP 1.2 it was mandatory to have **`<jsp:root>`** as the top element of a JSP written using the XML syntax. With JSP version 2.0, using **`<jsp:root>`** is no longer mandatory.

The tag-library-descriptor location

- The TLD file can be maintained as an independent file in the application-directory structure, or it can be part of the .jar file that contains the tag library.
- **Independent files**
  - With JSP 2.0, in the case of an independent TLD file the file must be present in the **WEB-INF directory or some subdirectory of it**. This TLD file also should not be placed in the WEB-INF/classes or WEB-INF/lib directories.
  - While developing your own tags, keep all your TLD files in a separate directory such as **WEB-INF/tags.** This makes editing the files as you work on them a lot easier.
  - Once you're done developing the tags you can package these TLDs into JAR files.
- **TLDs within JARs**
  - The easiest way to distribute tag libraries is by packaging them into JAR files.
  - Using a tag library packaged as a JAR file can be as simple as dumping that JAR file into the **WEB-INF/lib** directory of the application in which you intend to use the library.
  - For a tag library packaged as a JAR file the TLD files must reside in the META-INF directory **within** the JAR file or any of its subdirectories.
  - A tag library packaged as a JAR file can have one or more tag-library descriptors.

Tag-library-descriptor location


Assembly Root
WEB-INF
JSP pages, static HTML pages, applet classes, etc.
web.xml
sun-web.xml
*.tld
lib
classes
tags
Library archive files
All server-side .class files for this Web module
All .tag files for this Web module
JAR
TAGS

Conclusions

- In this lesson we looked at the usage and relevance of custom tags.
- With JSP 2.0, custom tags are a far easier and more viable option then they ever were before.
  - Although Classic tag handlers do confer the advantage of backward compatibility up to JSP version 1.1,
  - simple tag handlers should be used wherever possible.
- With an ever-growing number of freely available tag libraries on the Web, try and avoid developing new tags for tasks for which a tried and tested tag library might already exist.
- The JSP Standard Tag Libraries (JSTL) is another recent development that you must explore before taking up any tag development.
- If you find that you have too many complex situations to be tackled in your JSP, it is likely that a change of design is what is really required rather than creating new tags to handle complexities.