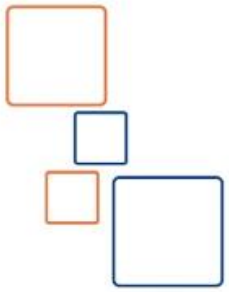
Building Dynamic Web Applications Using Servlets and JSP



Java™ Education and Technology Services

Invest In Yourself, Develop Your Career

Java™ Education and Technology Services

Chapter 8

JSP Basics
(Java Server Pages)

Ch8 : Outline

- Introduction to JSP
- JSP Vs. Servlets
- Advantages of JSP
- JSP Life Cycle
- JSP Packages
- JspPage & HttpJspPage Interfaces
- JSP Classes
- Strategies for Separating Presentation & Logic

Ch8 : Introduction to JSP

- Servlets :
  - Are easy to write and efficient to execute.
  - Make it simple to read request parameters.
  - Can easily make use of HTTP request headers.
  - Can flexibly manipulate HTTP response data.
  - Can customize their behavior based on cookies, track user-specific data with the session-tracking API.

What more do I need?

Ch8 : JSP Vs. Servlets

- Servlets:
  - Java programs with embedded HTML.
  - Generate dynamic content.
  - Do not separate static and dynamic content.

- Java Server Pages:
  - HTML pages with embedded Java code.
  - They can be pure XML-based files.
  - Generate dynamic content.
  - Separate static and dynamic content.

Ch8 : Advantages of JSP

- Easier to author (using website development tools).
- Separating the development team responsibilities.
- Business logic and presentation logic are separated.
- Simplify development with JSP tags, custom tags, and Java Beans.
- Recompile automatically when changes are made to the source file.

Ch8 : JSP Sample Page

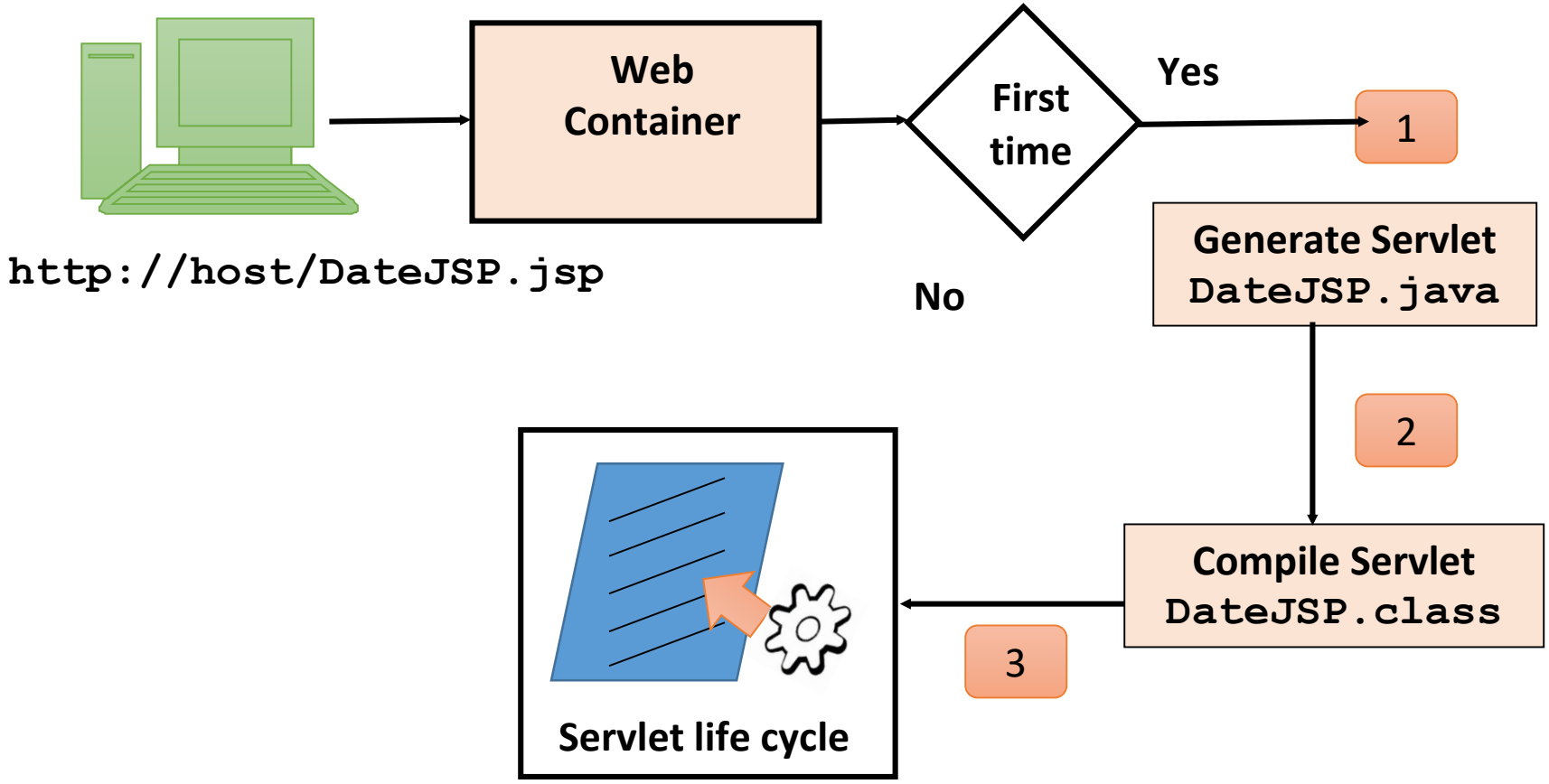
- The following is a sample web page that makes use of HTML tags and JSP tags:

```
<%@ page import="java.util.Enumeration" %>

<HTML>
<HEAD><TITLE> Using JSP </TITLE></HEAD>
<BODY BGCOLOR= #DADADA>
<%
        Enumeration parameters = request.getParameterNames();
        String param = null;
        while (parameters.hasMoreElements())
        {
            param = (String) parameters.nextElement();
            out.println(param + ":" + request.getParameter(param) + "<BR>");
        }
        out.close();
%>

</BODY>
</HTML>
```

Ch8 : JSP Life Cycle



Ch8 : Example

example.jsp

```
<HTML>
<HEAD>
</HEAD>
<BODY>


<% out.println ( "JSP is easy");%>


</BODY>
</HTML>
```

example_jsp.java

```
public void _jspService(.....)
{
      out.write("<HTML>\r\n");
    out.write("<HEAD>\r\n");
    out.write("</HEAD>\r\n");
    out.write("<BODY>\r\n");
                out.println("JSP is
easy");
    out.write("\r\n");
    out.write("</BODY>\r\n");
    out.write("</HTML>");
}
```

Program Files\Apache Software Foundation\Tomcat 5.5\work\Catalina\localhost\ MyApp

Ch8 : JSP Life Cycle

	JSP page translated into servlet	Servlet compiled	Servlet loaded into server's memory	jspInit called	_jspService called
Page first written					
Request1	Yes	Yes	Yes	Yes	Yes
Request2	No	No	No	No	Yes
Server restarted					
Request3	No	No	Yes	Yes	Yes
Request4	No	No	No	No	Yes
Page modified					
Request5	Yes	Yes	Yes	Yes	Yes
Request6	No	No	No	No	Yes

Ch 8: Are Servlets Still Needed?

- Yes, some tasks are better accomplished by servlets rather than by JSP (when the servlet is mostly intended for performing sophisticated business logic with little or no presentation).

- Remember: JSP pages are translated into servlets.

- In your project, you would typically use a combination of a few servlets and many JSP.

Ch 8: JSP Packages

- When working with JSP, you deal with two main packages:
  - **`jakarta.servlet.jsp`**
  - **`jakarta.servlet.jsp.tagext`** (Advanced)

- Common Interfaces:
  - **`JspPage`**
  - **`HttpJspPage`**

Ch 8: JspPage Interface

- The **`JspPage`** is the interface that must be implemented by all JSP servlet classes (directly or indirectly).
- It has two methods:
  - **`public void jspInit()`**
  - **`public void jspDestroy()`**
- The two methods can be overriden from within the JSP page, as in the following example:

```
<%!
    public void jspInit() {
    System.out.println("Init");
    }

    public void jspDestroy() {
    System.out.println("Destroy");
    }
%>
```

Ch 8: HttpJspPage Interface

- The HttpJspPage extends the JspPage interface.

- It has one method only:
  - `public void _jspService(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException`

- It is automatically implemented by the JSP container for the generated servlet. It holds the content of the JSP page.

- You are not allowed to override this method in your JSP page, because this method represents the page content itself .

Ch 8: JSP Classes

- The following section will go into the details of the following classes:
  - The **`JspFactory`** class.
  - The **`JspEngineInfo`** class.
  - The **`PageContext`** class.
  - The **`JspWriter`** class.

Ch 8: JSP Classes

Ch 8: JSP Classes

- The set of JSP classes are automatically used by the JSP container when generating the corresponding servlet of a JSP page.
- All the implicit objects are automatically created by the container in the implementation code of the **`_jspService(..)`** method.
- You do not need to go through any sophisticated code, you just use those ready objects in your JSP page.

Ch 8: Strategies for Separating Presentation & Logic

1. Call Java code directly from within your JSP (using scriptlets).

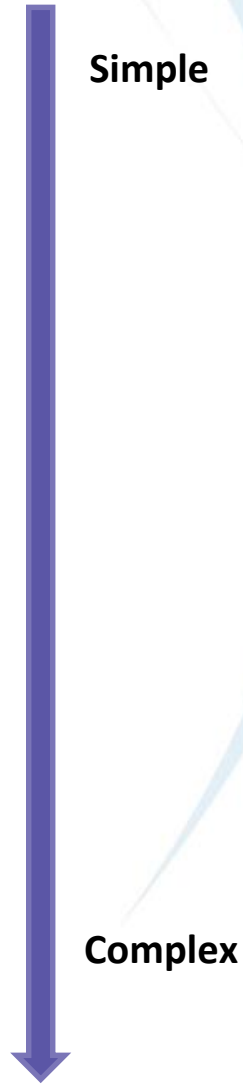
1. Develop separate utility classes, and only insert the Java code needed to invoke methods from the utility classes in your JSP.

1. Develop separate utility classes structured as beans.

1. Use the JSP expression language (EL) and JSTL.

1. Use custom tags.

1. MVC architecture: a controller servlet responds to requests, looks up data via model classes, stores the data in beans, and then forwards to a JSP page to that reads the beans to present the results.



Ch 8: The Need for Separation

- **Easier Development:** it is easier to write and maintain Java code in Java classes using IDEs.
- **Easier Compilation:** it is easier to detect code syntax errors when compiling a class regularly, rather than waiting for the server to run the generated corresponding servlet class.
- **Easier Debugging:** debugging is easy using special tools in IDEs.
- **Team Division:** you can have a team for web design and another team dedicated for business logic.
- **Easier Testing:** it is much easier to perform unit testing on methods than on JSP scriptlets.
- **Better Reuse:** Separating the business logic in classes helps towards better reuse of code components.

Java™ Education and Technology Services

Chapter 9

JSP Syntax

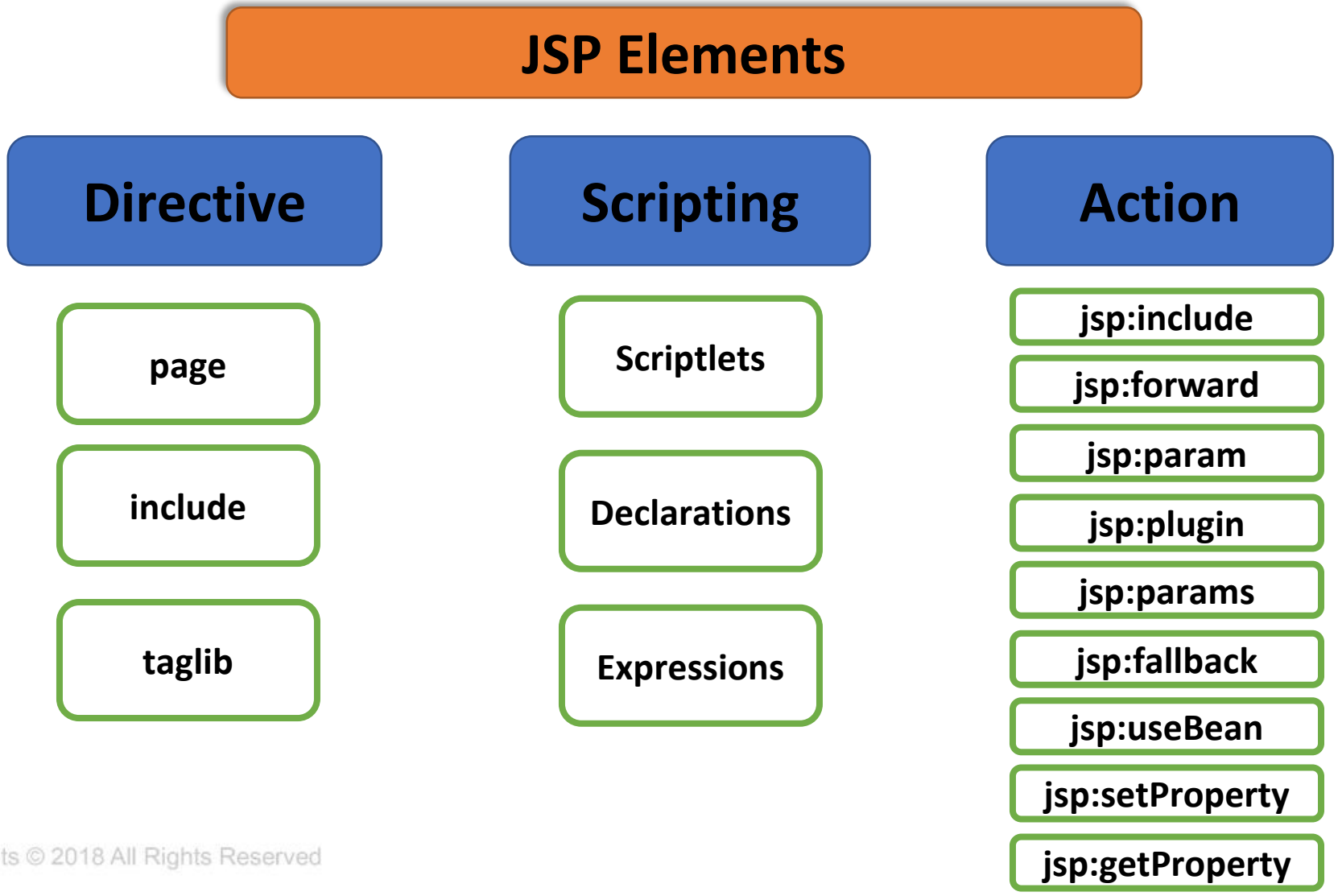
Ch 9: Outline

- ✓ JSP Syntax
- ✓ Directive Elements
- ✓ Scripting Elements
- ✓ Action Elements
- ✓ XML- Based JSP Tags

Ch 9: JSP Syntax

- A JSP page consists of two main parts:
  - **JSP Tags:** make up the syntax and semantics of JSP.
  - **Other Data :** anything else that is not intended for the JSP container (e.g. HTML, CSS, JavaScript tags).
- There are three broad categories of JSP tags:
  - Directive elements.
  - Scripting elements.
  - Action elements.

Ch 9: JSP Tags



Ch 9: Directive Elements

- Directive elements are messages to the JSP container. They hold information about how the container is to translate the JSP page into a corresponding servlet.
- They take the form of instructions, not Java code.
- Types of Directives:
  - Page directives.
  - Include directives.
  - Tag library directives (Advanced, used with custom tags).

Ch 9: Directive Elements

- The following is the general syntax of Directive Elements:
  - **<%@** directive (attribute="value")* **%>**

- The following samples demonstrate the use of Directives:
  - <%@ page buffer="16384" session="false" %>
  - <%@ include file="footer.html" %>
  - <%@ taglib uri="…." prefix ="c"%>

Ch 9: Directive Elements: Page Directive

Attribute	Value Type	Default Value
import	Fully qualified class name	None
contentType	MIME type, character set	"text/html;charset=ISO-8859-1"
session	boolean	"true"
buffer	Buffer size in Kb or false	8192
autoFlush	boolean	"true"
info	String	Depends on the container
errorPage	URL	None
isErrorPage	boolean	"false"
isThreadSafe	boolean	"true"
extends	Class name	None
language	Scripting language name	"java"

Ch 9: Directive Elements: Page Directive

- It is not allowed to repeat values for attributes except for the import attribute.

- The following are examples of different attributes of the Page Directive:
  - <%@ page import="java.util.Enumeration, java.util.Date" %>

  Or <%@ page import="java.util.Enumeration”>

  <%@ page import="ava.util.Date”>

  - <%@ page session="false" %>
  - <%@ page buffer="16384" %>
  - <%@ page autoFlush="false" %>
  - <%@ page errorPage="ErrorPage.jsp" %>

Ch 9: Directive Elements: Page Directive

- Utility classes must always be in packages, and the JSP page should use the import attribute.

- Utility classes must be placed in /WEB-INF/classes/ or in /WEB-INF/lib as jar files.

- By default, the generated servlet imports java.lang.*, jakarta.servlet.*, jakarta.servlet.jsp.*, jakarta.servlet.http.*

Ch 9: Directive Elements: Include Directive

- The Include Directive enables JSP authors to include the contents of other files in the current JSP page.
- We can include static pages like HTML or another JSP page.
- Include Syntax:
  - **<%@** include file="relativeURL" **%>**
- Example:
  - <%@ include file=“header.html" %>
- The whole code of the included page[HTML , JSP] is copied into the generated servlet of the current page.

Ch 9: Scripting Elements

- Scripting Elements allows the insertion of Java code in JSP pages.

- Types of Scripting Elements:
  - Scriptlets.
  - Declarations.
  - Expressions.

Ch 9: Scripting Elements: Scriptlets

- Scriptlets are used to directly write Java code inside JSP.
- Whatever code you write in scriptlets will be placed inside the **`_jspService()`** method of the generated servlet.
- Syntax of Scriptlets:
  - **<%** // place your java code here **%>**
- Example:

```
<%
  out.println("Current Time: " + getSystemTime()) ;
%>
```

Ch 9: Scripting Elements: Declarations

- Declarations are used to declare new methods or variables in JSP.
- Whatever code you write in Declaration tags will be inserted in the body of the generated servlet class (outside of the **`_jspService()`** method).
- It is not possible to access the implicit objects of the **`_jspService()`** method from within declaration elements.

Ch 9: Scripting Elements: Expressions

- Expression Elements are used to evaluate expressions and print them out to the user.

- Expressions are used as a short hand for evaluating an expression or variable then using **`out.println()`** to print its value.

- Syntax of Expression Element:
  - **<%=** expression **%>**
- Example:

```
<%= java.util.Calendar.getInstance().getTime() %>
```

has the same effect of :

```
<% out.print( java.util.Calendar.getInstance().getTime());%>
```

Ch 9: Action Elements

- The Standard Action Elements are a set of well-known tags for performing useful operations without the need to write Java code.

- Action Elements follow the XML syntax. They are:
  - jsp:include
  - jsp:forward
  - jsp:param
  - jsp:plugin
  - jsp:params
  - jsp:fallback
  - jsp:useBean
  - jsp:setProperty
  - jsp:getProperty

Ch 9: Action Elements: Include, Forward, Param

- They have same exact effect of the RequestDispatcher's **`include(..)`** and **`forward(..)`** methods.

- Syntax:

```
<jsp:include page="relativeURL“>
( <jsp:param . . . /> )*
</jsp:include>
```

```
<jsp:forward page="relativeURL">
( <jsp:param . . . /> )*
</ jsp:forward >
```

Ch 9: Action Elements: Include, Forward, Param

- The included pages can be HTML files, plain text files, JSP pages, or Servlets.
- When including JSP pages or servlets, only the output of the page is included, not the actual code.

- Do not use complete HTML documents for inclusion.
- The included page uses the same request object as the originally requested page and has access to the same request parameters.
- The main page can add to or replace the request parameters using jsp:param.

Ch 9: Action Elements: Include, Forward, Param

- The jsp:forward Action forwards the control to the destination page.
- The main page must not attempt to write any output to the response afterforwarding to the destination page.
- The main page can add to or replace the request parameters using jsp:param.

Ch 9: jsp:include Action Vs. @include Directive

<jsp:include page=“ url “ />

- Includes the ***output*** of a page at request time
- No change needed in the main page when the included pages change.
- The included pages cannot use any JSP constructs that affect the main page as a whole.

<%@ include file=“ url “%>

- Insert JSP code into the main page before that main page is translated into a servlet.
- The main page must be updated whenever any of the included pages change.
- The included code can contain JSP constructs such as field definitions and content-type settings that affect the main page as a whole.

Ch 9: Action Elements: useBean, setProperty, getProperty

- The jsp:useBean tag is used create an obtain a reference to a bean object and make use of it.
- Typically the bean is used to store and read data, to be passed from page to page.

- The jsp:setProperty and jsp:getProperty tags are used instead of using the Java code of the setters and getters.

- The jsp:useBean tag will be explored in details in the next chapter.

Ch 9: Action Elements: plugin & Fallback

- The **plugin** action is used to insert Java components into a JSP page. It determines the type of browser and inserts the <object> or <embed> tags as needed.
- If the needed plugin is not present, it downloads the plugin and then executes the Java component.
- The Java component can be either an Applet or a JavaBean.

```
<jsp:plugin type="applet" codebase="dirname" code="MyApplet.class" width="60" height="80">
        <jsp:param name="fontcolor" value="red" />
        <jsp:param name="background" value="black" />
        <jsp:fallback> Unable to initialize Java Plugin </jsp:fallback>
</jsp:plugin>
```

Ch 9: Comments in JSP

- HTML comments: **<!--** HTML Comment **-->**

- JSP comments: **<%--** JSP Comment **--%>**
  - **<% //** java comment **%>**

Ch 9: Converting to XML Syntax

- All the previous Directive and Scripting elements have an alternative XML-based tag.

- Benefits of using XML-based tags:
  - The JSP page can be edited and manipulated using an XML editing tool.
  - The XML syntax errors can be detected by the XML editing tool.
  - The semantic content of the JSP page can be validated against a set of descriptions (DTD files: Document Type Definition).

- Remember: the Standard Action Elements are already XML-based tags.

Ch 9: Converting to XML Syntax

- Syntax of the XML-based tags:
  - <jsp:directive.directiveName attribute_list />
  - <jsp:declaration> declaration code </jsp:declaration>
  - <jsp:scriptlet> scriptlet code </jsp:scriptlet>
  - <jsp:expression> expression </jsp:expression>

- Examples:

```
<jsp:directive.page attr="value" />

<jsp:scriptlet>
        String s; s = request.getParameter("user");
</jsp:scriptlet>
```

Ch 9: Converting to XML Syntax

- XML-compatible version of all standard JSP elements start with the jsp prefix (or namespace).
- XML tags are case sensitive.
- XML tags must be explicitly closed.
- To write a single quote ' within an attribute's value use \‘ and for double quotes " use \“ and for \ use \\ and for %> use %\> and for <% use <\%.

Java™ Education and Technology Services

Lab Exercise

Lab Exercise

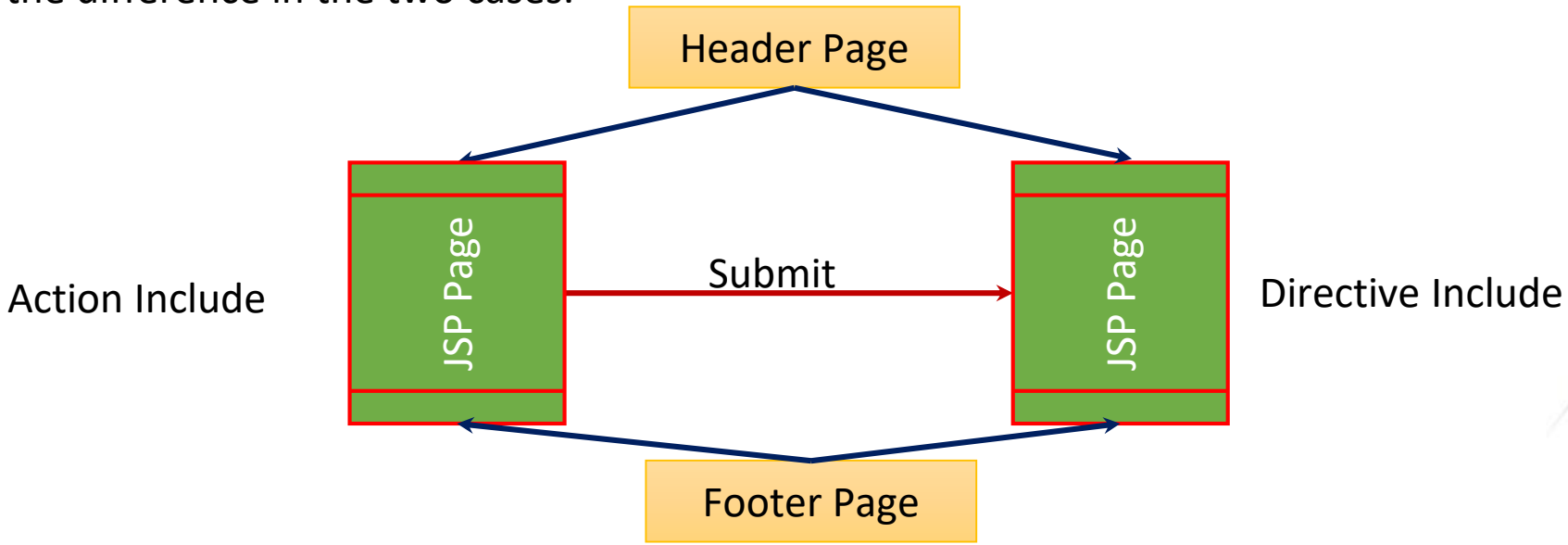
1.Form to Submit to a JSP

- Create an HTML page with an UserName field and a submit button.
- The submit button should take the user to a JSP page that displays the user's userName.

Lab Exercise

2.Including Header and Footer

- Create an HTML or JSP page that shows a header.
- Create another HTML or JSP page that shows a footer.
- Subsequently, use both jsp:include and include directive to include the header and footer to your page, and note the difference in the two cases.



Lab Exercise

3.Random Number Utility Classes

- Write a utility class that has a method that generates a random number within a given range.

- Invoke methods of this utility class from a jsp page using scripting elements.
- Form with one input field [range] post to it self.
- Import java.util.Random in JSP page directive.
- In scripting element :
  - *Check if request range not null.*
  - *Make an object form Utility class.*
  - *Call Generate random number method from this object and pass the range parameter to it.*
  - *Print the out to the response in specific div.*