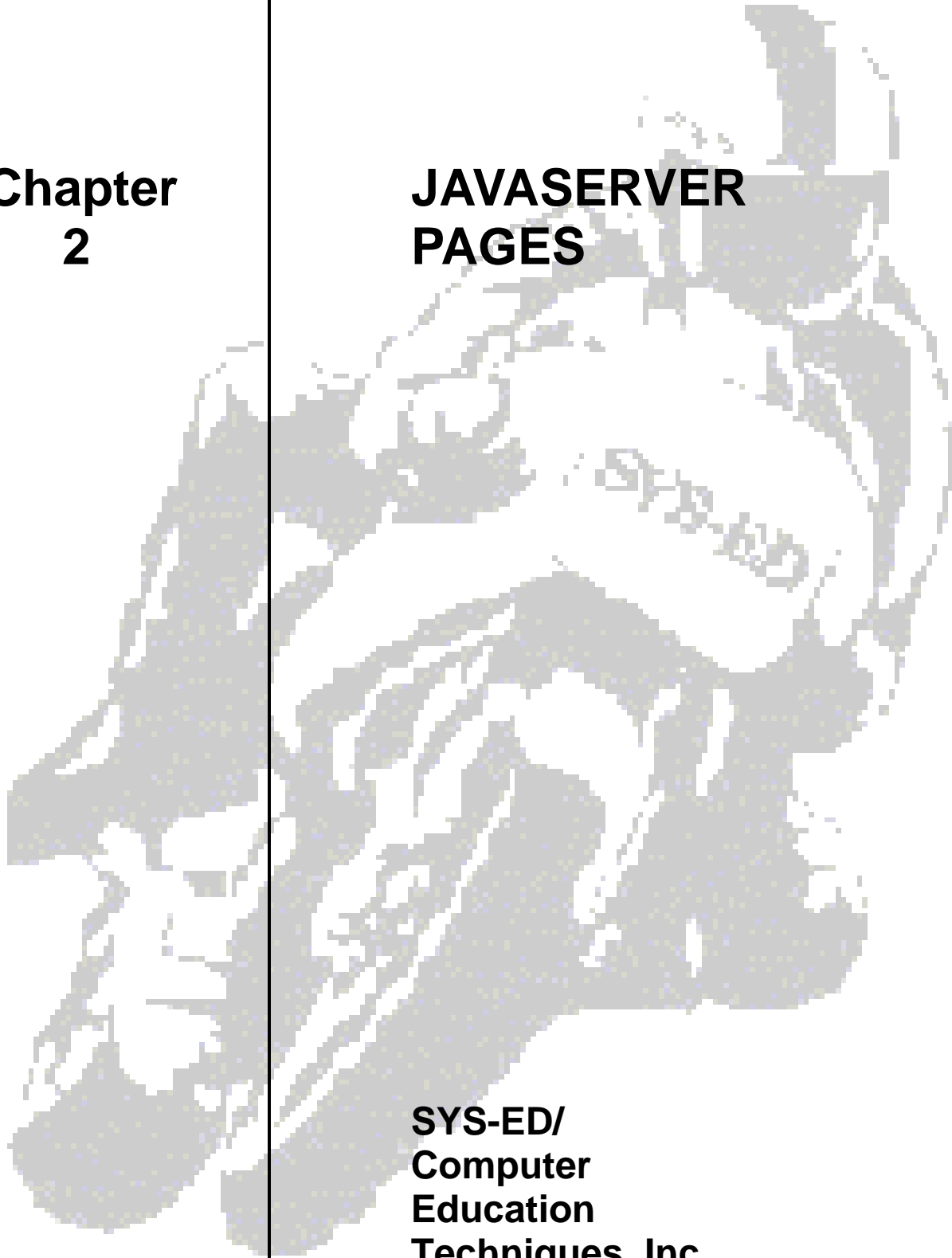


**Chapter
2**

**JAVASERVER
PAGES**



**SYS-ED/
Computer
Education
Techniques, Inc.**

Objectives

You will learn:

- JSP: purpose and function.
- Processing.
- Coding a JavaServer page.
- HTML and component-centric-and script-centered tags.
- JavaServer page components.
- Simple application.
- HTML form.
- Java Bean program.
- Property in JSP.
- GET and POST methods.
- Writing the bean.
- Scripting elements.
- Adding error pages.
- Core syntax commands.

1 What is JSP

JSP or JavaServer Pages is a technology for creating and maintaining dynamic-content web pages.

Based on the Java programming language, JavaServer Pages offers proven portability, open standards, and a re-usable component model. The JavaServer Pages architecture enables the separation of content generation from content presentation. This separation serves to facilitate maintenance headaches.

1.1 CGI

An early solution to this problem was the CGI-BIN interface; developers wrote individual programs to this interface, and web-based applications called the programs through the web server. This solution had scalability problems; each new CGI request launches a new process on the server.

If multiple users access the program concurrently, these processes consume all of the web server's available resources and the performance degrades.

Individual web server vendors have tried to simplify web application development providing "plug-ins" and APIs for their servers.

These solutions are web-server specific, and don't address the problem across multiple vendor solutions.

1.2 Other Technologies

Microsoft's ASP - Active Server Pages technology provides the capability for creating dynamic content on a web page, but only works with Microsoft IIS or Personal Web Server.

A Java Servlet is a Java technology-based program that runs on the server; as opposed to an applet, which runs on the browser. Servlets can be written that take an HTTP request from the web browser, generate the response dynamically and then send a response containing an HTML or XML document to the browser. With this approach, the entire page must be composed in the Java Servlet.

If a developer or web master wanted to modify the appearance of the page, they would have to edit and recompile the Java Servlet, even if the logic is already working.

1.3 JSP Portability

JavaServer Pages files can be run on any web server or web-enabled application server that provides support for them. The JSP engine support involves recognition, translation, and management of the JavaServer Page lifecycle and its interactions with associated components.

If the server on which the JavaServer Pages is to be executed supports the same specification level as that to which the file was written, no changes will be necessary as files are moved from server to server.

JavaServer Pages architecture can include reusable Java components. The architecture also allows for the embedding of a scripting language directly into the JavaServer Pages file.

The components currently supported include JavaBeans and Servlets.

As the default scripting language, JavaServer Pages use the Java programming language.

2 Processing

A JavaServer Pages file is an HTML document with JSP scripting or tags. It may have associated components in the form of .class, .jar, or .ser files.

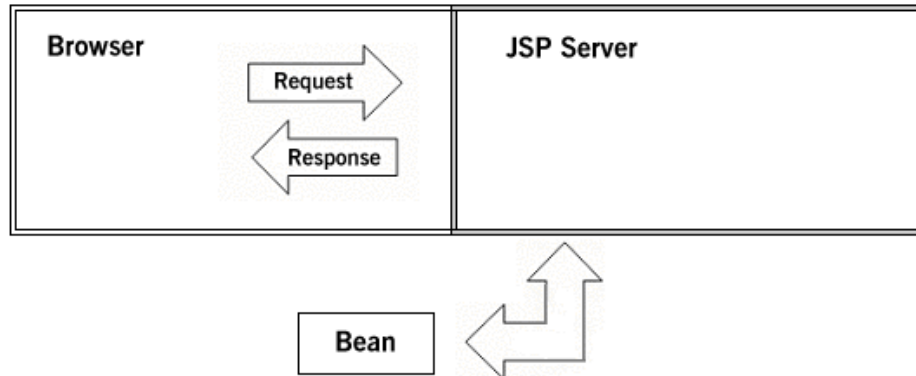
The .jsp extension is used to identify it to the server as a JavaServer Pages file. Before the page is served, the JavaServer Pages syntax is parsed and processed into a servlet on the server side.

The servlet that is generated outputs real content in straight HTML for responding to the client. Since it is standard HTML, the dynamically generated response looks no different to the client browser than a static response.

2.1 Access Models: Client to JavaServer Page

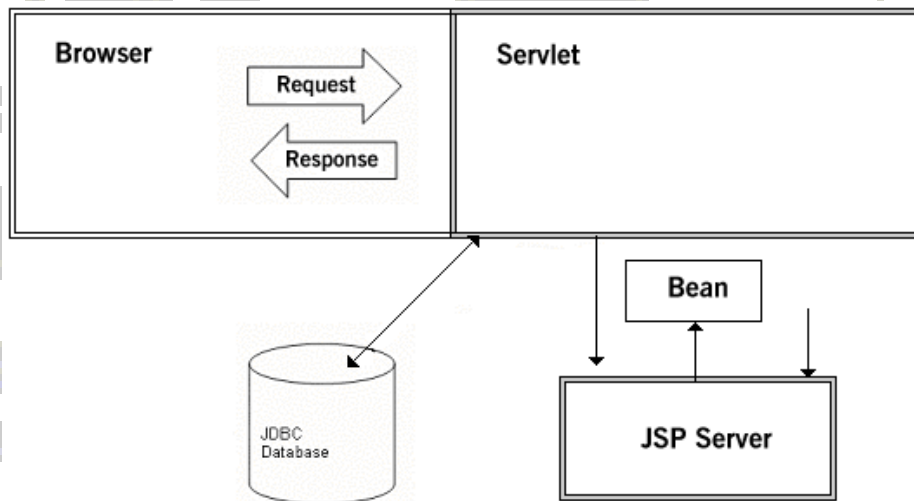
A JavaServer Pages file may be accessed in at least two different ways:

A client request comes directly into a JavaServer Page.



2.2 Access Models: Client to Servlet

A request comes through a servlet.



2.3 Access

The servlet generates the dynamic content. In order to handle the response to the client, the servlet creates and stores the dynamic content in the Bean. The servlet then invokes a JavaServer Page that will present the content along with the Bean generated from the servlet.

In both cases, the page could also contain any valid Java code.

2.4 Guidelines for Choosing Access Models

When evaluating whether a JavaServer Page should be used as the front-end to a servlet, back-end to a servlet; or whether only the servlet should be used, the following guidelines should be applied:

- When a graphical interface (GUI) is necessary to collect the request data, use a JavaServer Pages file.
- When the request and request parameters are otherwise available to the servlet, but the results of the servlet processing require a graphical interface to present them, use a JavaServer Pages file.
- When the presentation layout is minimal without many `println` lines in the servlet code and the presentation logic is not available to a customer or webpage designer, then a Servlet may be sufficient.

3 Coding a JavaServer Page

In the Java Web Server, JavaServer Pages are compiled to servlets. They have all the benefits associated with Java servlets:

- platform independence.
- standard API.
- network capabilities - RMI and JDBC
- faster than CGI.

JavaServer Pages have the following additional benefits:

- Separation of content generation from content presentation and leveraging of reusable components.
- Allows for the use of Java scriptlets and the use of JavaServer Pages tags to invoke Java components.

The possible permutations include:

- HTML only.
- HTML and Component-centric tags.

3.1 HTML

A file containing standard HTML code can be renamed with a filename extension of .jsp and it will need to meet the minimum requirements to be invoked as a JavaServer Page. This file would have to be parsed and compiled to a servlet. The servlet would still return a response when the page was invoked; its response to the client would contain the original HTML.

3.2 HTML and Component-centric Tags

A plain HTML file can have JavaServer Pages tags added in order to interact with Java components.

The component could be as simple as a Bean which returns the current time. No Java scriptlets are required.

```
<html>
<head>
<title>HTML plus Bean JSP File</title>
</head>
<jsp:useBean id="clock" scope="page" class="sunexamples.beans.JspCalendar"
type="sunexamples.beans.JspCalendar" />
<body>
<h1>Hello World (HTML)</h1>
<p>Today is: <jsp:getProperty name="clock" property="date"/></p>
</body>
</html>
```

3.3 HTML and Script-centric Tags

This code uses a simple scriptlet containing raw Java code.

```
<HTML>
<HEAD>
<TITLE>HTML and scriptlet JSP File</TITLE>
</HEAD>
<BODY>
<H1>
Hello World !
<P>
<%
out.println("The time now is: " + new java.util.Date());
%>
</H1>
</BODY>
</HTML>
```

4 HTML and Component-centric Tag and Script-centered Tag

By checking for a name parameter in the incoming request, the page can return either a generic or a personalized greeting.

```
<html>
<head>
<title>HTML and Bean and scriptlet JSP File</title>
</head>
<jsp:useBean id="clock" scope="page" class="sunexamples.beans.JspCalendar"
type="sunexamples.beans.JspCalendar" />
<body>
<h1>
<%
if (request.getParameter("name") == null) {
    out.println("Hello World");
} else {
    out.println("Hello" + request.getParameter("name"));
}
%>
</h1>
<p>Today is: <jsp:getProperty name="clock" property="date"/></p>
</body>
</html>
```

4.1 Testing the JSP File

With the Java web server running, type in the URL to the file. This will be relative to the Java web server's document root.

```
http://server_name:doc port/path_to_jsp_file
```

If myfirst.jsp file was located in the default document root (public_html) and on the default document port (8080), the URL to be typed in would be:

```
http://schnauzer.eng:8080/myfirst.jsp
```

5 JavaServer Page Components

A JSP page looks like a standard HTML or XML page, with additional elements that the JSP engine processes and strips out. Typically, the JSP elements create text that is inserted into the results page.

The page includes the following components:

Component	Description
JSP directive	Passes information to the JSP engine. The first line indicates the location of the Java programming language extensions which are to be accessible from this page. Directives are enclosed in <code><%@</code> and <code>%></code> markers.
Fixed template data	Any tags that the JSP engine does not recognize it will pass on with the results page. These will be HTML or XML tags.
JSP actions or tags	Implemented as standard or customized tags, and have an XML tag syntax.
Expression	The JSP engine evaluates anything between <code><%=</code> and <code>%></code> markers. In the List Items above, the values of the Day and Year attributes of the Clock bean are returned as a string and inserted as output in the JSP file.
Scriptlet	Is a small script that performs functions not supported by tags or ties everything together. The native scripting language for JSP software is based on the Java programming language.

5.1 JSP Directives

JSP pages use JSP directives to pass instructions to the JSP engine.

JSP directives can include:

Page directives	Communicates page-specific information, such as buffer and thread information or error handling.
Language directives	Specifies the scripting language, along with any extensions.
Include directive	Used to include an external document in the page.
Taglib directive	Indicates a library of custom tags that the page can invoke.

5.2 JSP Tags

Most JSP processing will be implemented through JSP-specific XML-based tags. JSP 1.0 includes a number of standard tags, referred to as the core tags.

jsp:useBean	A tag which declares the usage of an instance of a JavaBeans component. If the Bean does not already exist, then the JavaBean component instantiates and registers the tag.
jsp:setProperty	Sets the value of a property in a Bean.
jsp:getProperty	Gets the value of a Bean instance property, converts it to a string, and puts it in the implicit object "out".
jsp:include	Inserts source code.
jsp:forward	Serves to transfer control.

Tags are easy to use and share between applications. The real power of a tag-based syntax comes with the development of custom tag libraries, in which tool vendors or others can create and distribute tags for specific purposes.

5.3 Scripting Elements

JSP pages can include scriptlets in a page. A scriptlet is a code fragment which is executed at request time processing. Scriptlets may be combined with static elements on the page to create a dynamically-generated page.

Scripts are delineated within `<%` and `%>` markers. Anything within those markers will be evaluated by the scripting language engine. The JSP specification supports all of the usual script elements, including expressions and declarations.

6 Simple Application

This code is a simple HTML application.

```
<table border="0" width="400" cellspacing="0" cellpadding="0">
<tr>
<td height="150" width="150"> &nbsp; </td>
<td width="250"> &nbsp; </td>
</tr>
<tr>
<td width="150"> &nbsp; </td>
<td align="right" width="250">
 </td>
</tr>
</table>
<br>
```

6.1 Page Directive

The page directive is a JSP tag that will be used in JSP source files.

Example:

```
<%@ page info="a hello world example" %>
```

The page directive provides instructions to the JSP engine that applies to the entire JSP source file.

Page can specify:

- The scripting language used in the JSP source file packages would import the error page called if an error or exception occurs.
- The page directive can be used anywhere in the JSP file; proper coding style would have it placed at the top of the file.

7 HTML Form

```
<HTML>
<HEAD>
<TITLE>Simple JSP Example</TITLE>
</HEAD>
<BODY>
<P>How many times?</P>
<FORM METHOD="GET" ACTION="SimpleJSP.jsp">
<INPUT TYPE="TEXT" SIZE="2" NAME="numtimes">
<INPUT TYPE="SUBMIT">
</FORM>
</BODY>
</HTML>
```

7.1 Scriptlets Coding

```
<%@ page language="java" %>
<HTML>
<HEAD>
<TITLE>Simple JSP Example</TITLE>
</HEAD>
<BODY>
<P>
  <% int numTimes = Integer.parseInt(request.getParameter("numtimes"));
     for (int i = 0; i < numTimes; i++) {
       %>
         Hello, world!<BR>
       <% }
     %>
</P>
</BODY>
</HTML>
```

7.2 Date Object

```
<!--
  This is some comment in the second JSP
-->
<%@ page import="java.util.Date"%>
<%= "The current date is " +new Date() %>
```

7.3 Handling HTML Forms

One of the most common parts of an electronic commerce application is an HTML form in which a user enters some information. The information the user enters in the form is stored in the request object, which is sent from the client to the JSP engine. The JSP engine then sends the request object to whatever server-side component the JSP file specifies.

The component handles the request, retrieving data from a database or other data store, and passes a response object back to the JSP engine.

The JSP engine and web server then send the revised JSP page back to the client; where the user can view the results in the web browser.

7.4 Form Creation

An HTML form is typically defined in a JSP source file, using JSP tags to pass data between the form and some type of server-side object.

The following items need to be performed in the JSP application:

1. Writing a JSP source file, creating an HTML form, and giving each form element a name.
2. Writing the Bean in a .java file, defining properties, and get and set methods that correspond to the form element names.
3. Returning to the JSP source file, adding a `<jsp:useBean>` tag for creating or locating an instance of the Bean.
4. Adding a `<jsp:setProperty>` tag for setting properties in the Bean from the HTML form; the Bean will need a matching set method.
5. Adding a `<jsp:getProperty>` tag for retrieving the data from the Bean; the Bean will need a matching get method.
6. If more processing needs to be performed on the user data, use the request object from within a scriptlet.

10 GET and POST Methods

The HTTP GET and POST methods send data to the server. In a JSP application, GET and POST send data to the Bean, servlet, or other server-side component that is handling the form data.

GET is used for getting data from the server and POST is for sending data there. GET appends the form data to a query string to an URL, in the form of key/value pairs from the HTML form.

In the query string, key/value pairs are separated by & characters, spaces are converted to + characters, and special characters are converted to their hexadecimal equivalents. Since the query string is in the URL, the page can be bookmarked or sent as e-mail with its query string.

The query string is usually limited to a relatively small number of characters.

10.1 POST

The POST method, however, passes data of unlimited length as an HTTP request body to the server.

The user working in the client web browser cannot see the data that is being sent; therefore, POST requests will be well suited for sending confidential data or large amounts of data to the server.

11 Writing the Bean

When using a `<jsp:getProperty>` tag in a JSP source file, a corresponding get method will need to be included in the Bean.

When using a `<jsp:setProperty>` tag in a JSP source file, one or more corresponding set methods in the Bean will be needed.

11.1 Getting Data from the Form to the Bean

Setting properties in a Bean from an HTML form is a two-part task:

1. Creating or locating the Bean instance with `<jsp:useBean>`.
2. Setting property values in the Bean with `<jsp:setProperty>`.

In a JSP source file, the `<jsp:useBean>` tag must appear above the `<jsp:setProperty>` tag. The `<jsp:useBean>` tag looks for a Bean instance with the name that has been specified. If one is not found then the Bean, instantiates one. This allows a Bean to be created in one JSP file and used in another, as long as the Bean has a large enough scope.

The second step sets property values in the Bean with a `<jsp:setProperty>` tag. This is done by using `<jsp:setProperty>` for defining properties in the Bean with names that match the names of the form elements. It would also be necessary to define set methods for each property.

11.2 Checking the Request Object

The data the user enters is stored in the request object, which usually implements `javax.servlet.HttpServletRequest`. The request object can be accessed directly within a scriptlet.

Scriptlets are fragments of code written in a scripting language and placed within `<%` and `%>` characters.

The JSP engine always uses the request object behind the scenes, even if it is not called explicitly from a JSP file.

11.3 Getting Data from the Bean to the JSP Page

Once the user's data has been sent to the Bean, it can be used for retrieving the data and displaying it in the JSP page.

The `<jsp:getProperty>` tag is used to give it the Bean name and property name:

```
<h1>Hello, <jsp:getProperty name="mybean" property="username"/>!
```

The Bean names used on the `<jsp:useBean>`, `<jsp:setProperty>`, and `<jsp:getProperty>` tags must match:

hellouser.jsp:

```
<jsp:useBean id="mybean" scope="session" class="hello.NameHandler" />
<jsp:setProperty name="mybean" property="*" />
```

response.jsp:

```
<h1>Hello, <jsp:getProperty name="mybean" property="username"/>!
```

The tags are in two files, but the Bean names still must match. When they do not match, JSP throws an error.

11.4 Useful Methods

<code>getRequest javax.servlet.jsp.PageContext</code>	Returns the current request object.
<code>getParameterNames javax.servlet.ServletRequest</code>	Returns the names of the parameters request currently contains.
<code>getParameterValues javax.servlet.ServletRequest</code>	Returns the values of the parameters request currently contains.
<code>GetParameter javax.servlet.ServletRequest</code>	Returns the value of a parameter when the name.JSP. has been provided.

12 Scripting Elements

Programmatic implementation can be added to JSP files. The JSP tags can be used for encapsulating tasks that would be difficult or time-consuming to program.

Scripting language fragments can be used to supplement the JSP tags.

The scripting languages which are available depend on the JSP engine being used.

12.1 General Rules for Scripting

The following rules need to be adhered to when adding scripting elements to a JSP source file:

1. Use a page directive to define the scripting language used in the JSP page.
2. The declaration syntax `<%! .. %>` declares variables or methods.
3. The expression syntax `<%= .. %>` defines a scripting language expression and casts the result as a String.
4. The scriptlet syntax `<% .. %>` can handle declarations, expressions, or any other type of code fragment valid in the page scripting language.
5. When writing a scriptlet, end the scriptlet with `%>` before switching to HTML, text, or another JSP tag.

12.2 Difference Between `<%`, `<%=`, and `<%`

Declarations, expressions, and scriptlets have similar syntax and usage, but also some important differences.

Declarations (between `<%!` and `%>` tags) contain one or more variable or method declarations that end or are separated by semicolons:

```
<%! int i = 0; %>
<%! int a, b; double c; %>
<%! Circle a = new Circle(2.0); %>.
```

A variable or method must be declared in a JSP page before using it in the page. The scope of a declaration is a JSP file, but if the JSP file includes other files with the include directive, the scope expands to cover the included files as well.

Expressions (between `<%=` and `%>` tags) can contain any language expression that is valid in the page scripting language, but without a semicolon:

```
<%= Math.sqrt(2) %>
<%= items[i] %>
<%= a + b + c %>
<%= new java.util.Date() %>
```

The definition of a valid expression is up to the scripting language.

The parts of the expression are evaluated in left-to-right order.

One key difference between expressions and scriptlets is that a semicolon is not allowed within expression tags, even if the same expression requires a semicolon when you use it within scriptlet tags.

Scriptlets (between `<%` and `%>` tags) provide the capability for allowing a variety of valid scripting language statements to be written.

```
<%
String name = null;
if (request.getParameter("name") == null) {
%>
```

A scriptlet must end a language statement with a semicolon if the language requires it.

When writing a scriptlet, any of the JSP implicit objects or classes imported by the page directive, declared in a declaration, or named in a `<jsp:useBean>` tag can be used.

12.3 Counter Scriptlet

```
<html>
<body>
<h1> This is a counter example </h1>

<%! int i=0 ; %>

<%
i++;
%>
Hello World ! <%= "This JSP has been accessed " +i +" times" %>
</body>
</html>
```

13 Number Guess Game JSP

The Number Guess game utilizes scriptlets, expressions, and HTML forms.

```
<%@ page import = "num.NumberGuessBean" %>
<jsp:useBean id="numguess" class="num.
NumberGuessBean" scope="session" />
<jsp:setProperty name="numguess" property="*" />
<html>
<head><title>Number Guess</title></head>
<body bgcolor="white">
<font size=4>
<% if (numguess.getSuccess() ) { %>
Congratulations! You got it.
And after just <%= numguess.getNumGuesses() %>
tries.<p>
<% numguess.reset(); %>
Care to <a href="numguess.jsp">try again</a>?
<% } else if (numguess.getNumGuesses() == 0) { %>
Welcome to the Number Guess game.<p>
I'm thinking of a number between 1 and 100.<p>
<form method=get>
What's your guess? <input type=text name=gues>
<input type=submit value="Submit">
</form>
<% } else { %>
Good guess, but nope. Try <b><%= numguess.
getHint() %></b>.
You have made <%= numguess.getNumGuesses() %>
guesses.<p>.
I'm thinking of a number between 1 and 100.<p>
<form method=get>
What's your guess? <input type=text name=gues>
<input type=submit value="Submit">
</form>
<%}%>
</font>
</body>
</html>.
```

14 Java Program

```
package num;
import java.util.*;
public class NumberGuessBean {
    int answer;
    boolean success;
    String hint;
    int numGuesses;
    public NumberGuessBean() {
        reset();
    }
    public void setGuess(String guess) {
        numGuesses++;
        int g;
        try {
            g = Integer.parseInt(guess);
        }
        catch (NumberFormatException e) {
            g = -1;
        }
        if (g == answer) {
            success = true;
        }
        else if (g == -1) {
            hint = "a number next time";
        }
        else if (g < answer) {
            hint = "higher";
        }
        else if (g > answer) {
            hint = "lower";
        }
    }
    public boolean getSuccess() {
        return success;
    }
}
```

14.1 Scripting Elements in a JSP File

Scriptlets are not required to write scriptlets which are mingled with HTML and JSP tags. Between the `<%` and `%>` tags, lines of scripting language code can be written. Doing less processing in scriptlets and more in components like servlets or Beans makes an application code more reusable and portable.

15 Adding Error Pages

Error pages are specialized JSP pages which display information about exceptions.

The following steps need to be performed in order to add error pages that display exception information to a web application:

1. Write a bean or enterprise bean, servlet, or other component in order that it throws certain exceptions under certain conditions.
2. Use a simple tracking mechanism in a component to help gather information about what a user was doing when the exception was thrown.
3. In the JSP file, use a page directive with `errorPage` set to the name of a JSP file that will display a message to the user when an exception occurs.
4. Write an error page file, using a page directive with `isErrorPage="true"`.
5. In the error page file, use the exception object to get information about the exception.
6. Use informative messages, either in the error page file or included from other files, in order to present a user with an informative message relevant to what he or she was doing when the exception was thrown.

16 Core Syntax Commands

Syntax Command	Description	Example
HTML Comment	Generates a comment that is sent to the client.	<code><!-- comment [<%= expression %>] --></code>
Hidden Comment	Documents the JSP page; but is not sent to the client.	<code><%-- comment --%></code>
Declaration	Declares a variable or method valid in the scripting language used in the JSP page.	<code><%! declaration; [declaration;]+ ... %></code>
Expression	Contains an expression valid in the scripting language used in the JSP page.	<code><%= expression %></code>
Scriptlet	Contains a code fragment valid in the page scripting language.	<code><% code fragment %></code>
Include Directive	Includes a static file in a JSP file, parsing the file's JSP elements.	<code><%@ include file="relativeURL" %></code>
Page Directive	Defines attributes that apply to an entire JSP page.	<pre> <%@ page [language="java"] [extends="package .class"] [import="{package .class package.*}, ..."] [session="true false"] [buffer="none 8kb sizekb"] [autoFlush="true false"] [isThreadSafe="true false"] [info="text"] [errorPage="relativeURL"] [contentType="mimeType [; charset=characterSet]"] "text/html ; charset=ISO-8859-1"] [isErrorPage="true false"] %> </pre>

Syntax Command	Description	Example
Taglib Directive	Defines a tag library and prefix for the custom tags used in the JSP page.	<pre><%@ taglib uri="URIToTagLibrary" prefix="tagPrefix" %></pre>
<jsp:forward>	Forwards a client request to an HTML file, JSP file, or servlet for processing.	<pre><jsp:forward page="{relativeURL <%= expression %>}" /> or <jsp:forward page="{relativeURL <%= expression %>}" > <jsp:param name="parameterName" value="{parameterValue <%= expression %>}" />+ </jsp:forward></pre>
<jsp:getProperty>	Gets the value of a Bean property in order that it can be displayed in a result page.	<pre><jsp:getProperty name="beanInstanceName" property="propertyName" /></pre>
<jsp:include>	Includes a static file or sends a request to a dynamic file.	<pre><jsp:include page="{relativeURL <%= expression %>}" flush="true" /> or <jsp:include page="{relativeURL <%= expression %>}" flush="true" > <jsp:param name="parameterName" value="{parameterValue <%= expression %>}" />+ </jsp:include></pre>

Syntax Command	Description	Example
<jsp:plugin>	Executes an applet or Bean and, if necessary, downloads a Java plug-in to execute it.	<pre> <jsp:plugin type="bean applet" code="classFileName" codebase="classFileDirectoryName" [name="instanceName"] [archive="URIToArchive, ..."] [align="bottom top middle left right"] [height="displayPixels"] [width="displayPixels"] [hspace="leftRightPixels"] [vspace="topBottomPixels"] [jreversion="JREVersionNumber 1.1"] [nspluginurl="URLToPlugin"] [iepluginurl="URLToPlugin"] > [<jsp:params> [<jsp:param name="parameterName" value="{parameterValue <%= expression %>" /> }" />]+ </jsp:params>] [<jsp:fallback> text message for user </jsp:fallback>] </jsp:plugin> </pre>

Syntax Command	Description	Example
<jsp:setProperty>	Sets a property value or values in a Bean.	<pre> <jsp:setProperty name="beanInstanceName" { property="*" property="propertyName" [param="parameterName"] property="propertyName" value= "{string <%= expression %>}" } /> </pre>
<jsp:useBean>	Locates or instantiates a Bean with a specific name and scope.	<pre> <jsp:useBean id="beanInstanceName" scope="page request session application" { class="package.class" type="package.class" class="package.class" type="package.class" beanName="{ package.class <%= expression %>}" type ="package.class" } {/> > other elements </jsp:useBean> } </pre>