

**Chapter
2**

**CODING
JSP**

*Get on the
Fast Track!*

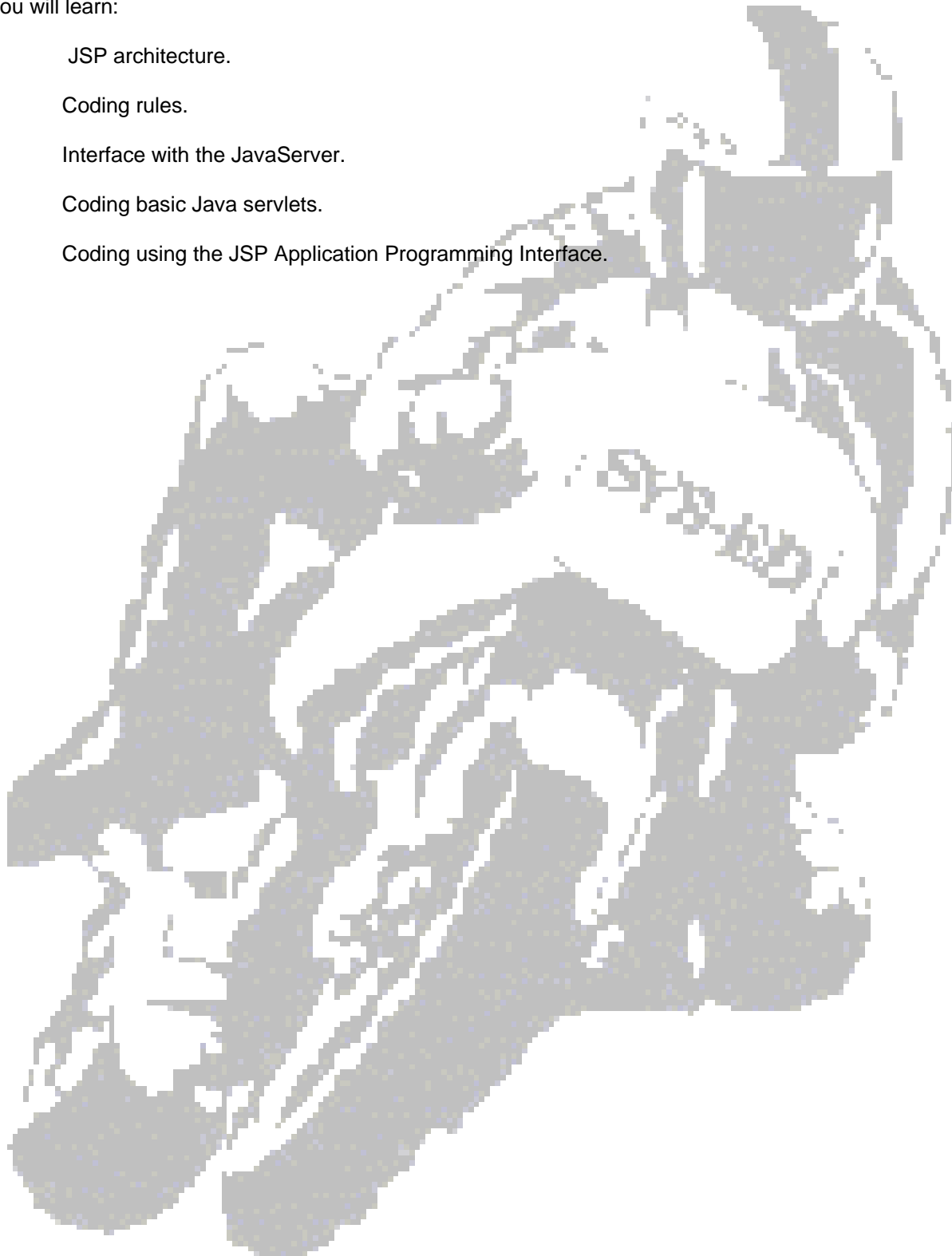


**SYS-ED/
COMPUTER
EDUCATION
TECHNIQUES, INC.**

Objectives

You will learn:

- C JSP architecture.
- C Coding rules.
- C Interface with the JavaServer.
- C Coding basic Java servlets.
- C Coding using the JSP Application Programming Interface.



1 Sample Application: sysed.jsp

This code is a simple HTML application; it is called sysed.jsp.

```
1 <table border="0" width="400" cellspacing="0" cellpadding="0">
2 <tr>
3 <td height="150" width="150"> &nbsp; </td>
4 <td width="250"> &nbsp; </td>
5 </tr>
6 <tr>
7 <td width="150"> &nbsp; </td>
8 <td align="right" width="250">
9  </td>
10 </tr>
11 </table>
12 <br>
```

2 Page Directive

The page directive is a JSP tag that will be used in almost every JSP source file.

The line that looks like this:

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

The page directive gives instructions to the JSP engine that apply to the entire JSP source file.

In our example, page specifies an informative comment that will become part of the compiled JSP file.

Page might specify:

- C the scripting language used in the JSP source file.
- C packages the source file would import.
- C the error page called if an error or exception occurs.

The page directive can be used anywhere in the JSP file, but it's good coding style to place it at the top of the file.

Since it's a JSP tag, it can be placed before the opening <html> tag.

3 Include Directive

The include directive inserts the contents of another file in the main JSP file, where the directive is located.

It is useful for including copyright information, scripting language files, or anything you might want to reuse in other applications.

In this example, the included file is an HTML table that creates a graphic banner.

```
1     <%@ page info="a hello world example" %>
2     <html>
3     <head><title>Hello, World</title></head>
4     <body bgcolor="#ffffff" background="background.gif">
5     <%@ include file="dukebanner.html" %>
6     <table>
7     <tr>
8     <td width=150> &nbsp; </td>
9     <td width=250 align=right> <h1>Hello, World!</h1> </td>
10    </tr>
11    </table>
12    </body>
13    </html>
```

The content of the included file can be seen by viewing the page source of the main JSP file while you are running Hello, World.

The included file does not contain `<html>` or `<body>` tags, because these tags would conflict with the same tags in the calling JSP file.

JSP tags are case sensitive. If you type `<jsp:usebean>` instead of `<jsp:useBean>`, your tag will not be recognized.

Some of the attributes on the tags take class names, package names, pathnames or other case-sensitive values as well.

4 Simple HTML Form

```
1 <HTML>
2 <HEAD>
3 <TITLE>Simple JSP Example</TITLE>
4 </HEAD>
5
6 <BODY>
7
8 <P>How many times?</P>
9
10 <FORM METHOD="GET" ACTION="SimpleJSP.jsp">
11 <INPUT TYPE="TEXT" SIZE="2" NAME="numtimes">
12 <INPUT TYPE="SUBMIT">
13 </FORM>
14
15 </BODY>
16 </HTML>
```

4.1 PageFooter.html

```
<P>SimpleJSP example </P>
```

4.2 Code with Scriptlets (SimpleJSP.jsp)

```
1 <%@ page language="java" %>
2
3 <HTML>
4 <HEAD>
5 <TITLE>Simple JSP Example</TITLE>
6 </HEAD>
7
8 <BODY>
9
10 <P>
11 <% int numTimes = Integer.parseInt(request.getParameter("numtimes"));
12 <% for (int i = 0; i < numTimes; i++) {
13 <%
14 <% Hello, world!<BR>
15 <% }
16 <%
17 </P>
18
19 <%@ include file="PageFooter.html" %>
20
21 </BODY>
22 </HTML>
```

4.3 Date Object

```
1  <!--  
2     This is some comment in the second JSP  
3  -->  
4  <%@ page import="java.util.Date"%>  
5  <%= "The current date is " +new Date() %>
```



5 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 sends the request object to whatever server-side component (JavaBeans component, servlet, or enterprise bean) the JSP file specifies.

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

The JSP engine passes the response object to the JSP page, where its data is formatted according to:

Client

response
request

JSP Engine & Web Server

request
request

JSP File

response
response

Component

response
response

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.

- C The communications protocol used between the client and server can be HTTP, or it can be some other protocol.
- C The request and response objects are always implicitly available to you as you author JSP source files.

6 Creating a Form

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; which is typically a Bean.

The following things need to be done with a JSP application:

1. Start writing a JSP source file, creating an HTML form and giving each form element a name.
2. Write the Bean in a .java file, defining properties, get, and set methods that correspond to the form element names; that is unless the objective is to set one property value at a time explicitly.
3. Return to the JSP source file. Add a `<jsp:useBean>` tag to create or locate an instance of the Bean.
4. Add a `<jsp:setProperty>` tag to set properties in the Bean from the HTML form; the Bean needs a matching set method.
5. Add a `<jsp:getProperty>` tag to retrieve the data from the Bean.
The Bean needs a matching get method.
6. If you need to do even more processing on the user data, use the request object from within a scriptlet..JSP by Example 7.

8 Java Bean Program

```
1      <%@ page import="hello.NameHandler" %>
2      <jsp:useBean id="mybean" scope="page" class="hello.NameHandler" />
3      <jsp:setProperty name="mybean" property="*" />
4      <html>
5      <head><title>Hello, User</title></head>
6      <body bgcolor="#ffffff" background="background.gif">
7      <%@ include file="dukebanner.html" %>
8      <table border="0" width="700">
9      <tr>
10     <td width="150"> &nbsp; </td>
11     <td width="550">
12     <h1>My name is Duke. What's yours?</h1>
13     </td>
14     </tr>
15     <tr>
16     <td width="150" &nbsp; </td>
17     <td width="550">
18     <form method="get">
19     <input type="text" name="username" size="25">
20     <br>
21     <input type="submit" value="Submit">
22     <input type="reset" value="Reset">
23     </td>
24     </tr>
25     </form>
26     </table>
27     <%
28     if ( request.getParameter("username") != null ) {
29     %>
30     <%@ include file="response.jsp" %>
31     <%
32     }
33     %>
34     </body>
35     </html>.JSP by Example 9
```


In other Web applications, the action attribute specifies a CGI script or other program that will process the form data.

In a JSP file, the action attribute can be omitted if the data processed will be processing by the object specified in the `<jsp:useBean>` tag. An action can also be used for specifying another JSP file to which the data should be sent.

The rest of the form is constructed just like a standard HTML form, with input elements, a Submit button, and a Reset button.

Be sure to give each input element a name.

Example:

```
<input type="text" name="username">
```

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	<p>GET used is for getting data from the server</p> <p>GET appends the form data (called a query string) to an URL, in the form of key/value pairs from the HTML form, for example, name=John.</p> <p>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. Because the query string is in the URL, the page can be bookmarked or sent as email with its query string.</p> <p>The query string is usually limited to a relatively small number of characters.</p>
POST	<p>POST is used for sending data there.</p> <p>The POST method passes data of unlimited length as an HTTP request body to the server.</p> <p>The user working in the client Web browser cannot see the data that is being sent, so POST requests are ideal for sending confidential data (such as a credit card number) or large amounts of data to the server.</p>

11 Writing the Bean

If a `<jsp:getProperty>` tag is used in a JSP source file, a corresponding get method will be needed in the Bean.

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

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>`

The first step is to instantiate or locate a Bean with a `<jsp:useBean>` tag before setting property values in the Bean.

In a JSP source file, the `<jsp:useBean>` tag must appear above the `<jsp:setProperty>` tag. The `<jsp:useBean>` tag first looks for a Bean instance with the name you specify, but if it doesn't find the Bean, it 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.

2. Setting property values in the Bean with `<jsp:setProperty>`

The second step is to set property values in the Bean with a `<jsp:setProperty>` tag.

The easiest way to use `<jsp:setProperty>` is to define properties in the Bean with names that match the names of the form elements. It will also be necessary to define the corresponding set methods for each property.

For example, if the form element is named `username`, then in the bean it would be necessary to define:

- a property `username` property.
- methods `getUsername` and `setUsername` in the Bean.

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 has not been 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, you may want to retrieve the data and display it in the JSP page.

To do this, use the `<jsp:getProperty>` tag, giving 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.

Example:

```
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 in this example are in two files, but the Bean names still must match. If they don't, JSP throws an error.

11.4 Useful Methods

<code>getRequest</code>	<code>javax.servlet.jsp.PageContext</code>	Returns the current request object.
<code>getParameterNames</code>	<code>javax.servlet.ServletRequest</code>	Returns the names of the parameters request currently contains.
<code>getParameterValues</code>	<code>javax.servlet.ServletRequest</code>	Returns the values of the parameters request currently contains.
<code>getParameter</code>	<code>javax.servlet.ServletRequest</code>	Returns the value of a parameter if you provide the name.JSP.

12 Scripting Elements

At some point, it will be useful to add augment JSP files with the features of a programming language. The JSP tags encapsulate tasks that would be difficult or time-consuming to program. However, a scripting language fragment will be required to supplement the JSP tags.

The scripting languages which are available will depend on the JSP engine being used. Some vendors' JSP engines will include support for other scripting languages.

12.1 Rules for Scripting

Use the following guidelines when adding scripting elements to a JSP source file:

1. Use a page directive to define the scripting language used in the JSP page.
This will not be required when the Java language is used since this is the default value.
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.

13 Difference Between <%, <%=, and <%!

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

Declarations

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 in a JSP page must be declared if it is used in the page.

The scope of a declaration is usually 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

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.

A major 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

Scriptlets (between `<%` and `%>` tags) allow you to write any number of valid scripting language statements:

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

In a scriptlet a language statement must be ended 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 can be used, declared in a declaration, or named in a `<jsp:useBean>` tag.

13.1 Counter Scriptlet

```
1 <html>  
2 <body>  
3 <h1> This is a counter example </h1>  
4  
5 <%! int i=0 ; %>  
6  
7 <%  
8 i++;  
9 %>  
10 Hello World ! <%= "This JSP has been accessed " +i +" times" %>  
11 </body>  
12 </html>
```

14 The Number Guess Game JSP

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

```
1      <!--
2      Number Guess Game
3      Written by Jason Hunter, CTO, K&A Software
4      jasonh@kasoftware.com, http://www.servlets.com
5      Copyright 1999, K&A Software
6      Distributed by Sun Microsystems with permission
7      -->
8      <%@ page import = "num.NumberGuessBean" %>
9      <jsp:useBean id="numguess" class="num.
10     NumberGuessBean" scope="session" />
11     <jsp:setProperty name="numguess" property="*" />
12     <html>
13     <head><title>Number Guess</title></head>
14     <body bgcolor="white">
15     <font size=4>
16     <% if (numguess.getSuccess() ) { %>
17     Congratulations! You got it.
18     And after just <%= numguess.getNumGuesses() %>
19     tries.<p>
20     <% numguess.reset(); %>
21     Care to <a href="numguess.jsp">try again</a>?
22     <% } else if (numguess.getNumGuesses() == 0) { %>
23     Welcome to the Number Guess game.<p>
24     I'm thinking of a number between 1 and 100.<p>
25     <form method=get>
26     What's your guess? <input type=text name=gues>
27     <input type=submit value="Submit">
28     </form>
29     <% } else { %>
30     Good guess, but nope. Try <b><%= numguess.
31     getHint() %></b>.
32     You have made <%= numguess.getNumGuesses() %>
33     guesses.<p>.18 JavaServer Pages Developer's Guide
34     I'm thinking of a number between 1 and 100.<p>
35     <form method=get>
36     What's your guess? <input type=text name=gues>
37     <input type=submit value="Submit">
38     </form>
39     <%}%>
40     </font>
41     </body>
42     </html>.
```

15 Java Program

```
1 package num;
2 import java.util.*;
3 public class NumberGuessBean {
4     int answer;
5     boolean success;
6     String hint;
7     int numGuesses;
8     public NumberGuessBean() {
9         reset();
10    }
11    public void setGuess(String guess) {
12        numGuesses++;
13        int g;
14        try {
15            g = Integer.parseInt(guess);
16        }
17        catch (NumberFormatException e) {
18            g = -1;
19        }
20        if (g == answer) {
21            success = true;
22        }
23        else if (g == -1) {
24            hint = "a number next time";
25        }
26        else if (g < answer) {
27            hint = "higher";
28        }
29        else if (g > answer) {
30            hint = "lower";
31        }
32    }
33    public boolean getSuccess() {
34        return success;

```

16 Scripting Elements in a JSP File

The file numguess.jsp utilizes scripting elements. It has been structured in a form similar to a source file, with a large if ... else statement within scriptlet tags.

The difference is that the body of each statement clause is written in HTML and JSP tags, rather than in a programming language. It is not a required that scriptlets be mingled with HTML and JSP tags, as shown in numguess.jsp.

Between the `<%` and `%>` tags, as many lines of scripting language code can be written as are required. In general, doing less processing in scriptlets and more in components like servlets or Beans makes application code more reusable and portable. Nonetheless, how a JSP application is written determined by the programmer.

Sun's JSP 1.0 reference implementation specifies no limit on the length of a scriptlet.

16.1 Mingling Scripting Elements with Tags

When mingling scripting elements with HTML and JSP tags, a scripting element must always end before tags can be used. The scripting element can then be reopened.

Example:

```
1     public String getHint() {
2     return "" + hint;
3     }
4     public int getNumGuesses() {
5     return numGuesses;
6     }
7     public void reset() {
8     answer = Math.abs(new Random().nextInt() % 100)
9     +1;
10    success = false;
11    numGuesses = 0;
12    }
13    }.JSP by Example 21
14    <%}else {%> <!-- closing the scriptlet before the tags start -->
15    ... tags follow ...
16    <%}%><!-- reopening the scriptlet to close the language block -->
```

16.2 Executing Scripting Elements

A JSP source file is processed in two stages:

- C HTTP translation time.
- C request processing time.

HTTP Translation Time

HTTP translation time occurs when a user first loads a JSP page. The JSP source file is compiled to a Java class, usually a Java servlet. The HTML tags and as many JSP tags as possible are processed at this stage, before the user makes a request.

Request processing time occurs when a user clicks in the JSP page to make a request. The request is sent from the client to the server by way of the request object. The JSP engine then executes the compiled JSP file, or servlet, using the request values the user submitted.

When using scripting elements in a JSP file, you should know when they are evaluated. Declarations are processed at HTTP translation time and are available to other declarations, expressions, and scriptlets in the compiled JSP file.

Expressions are also evaluated at HTTP translation time.

The value of each expression is converted to a String and inserted in place in the compiled JSP file.

Scriptlets, however, are evaluated at request processing time, using the values of any declarations and expressions that are made available to them.

17 Handling Exceptions

When something is entered incorrectly in a JSP application, if the application has been well written, it will throw an exception and display an error page.

Exceptions that occur while a JSP application is running are called runtime exceptions.

As with a Java application, an exception is an object that is an instance of `java.lang.Throwable` or one of its subclasses.

`Throwable` has two standard subclasses:

- C `java.lang.Exception`, which describes exceptions.
- C `java.lang.Error`, which describes errors.

Errors are different from exceptions. Errors usually indicate linkage or virtual machine problems which a Web application probably will not recover from, such as running out of memory.

Exceptions, however, are conditions that can be caught and recovered from. An example of an exceptions would be a `NullPointerException` or a `ClassCastException`. These exceptions indicate that a null value or a value of the wrong data type has been passed to the application while it is running.

Runtime exceptions are easily handled in a JSP application, because they are stored one at a time in the implicit object named `exception`. The exception object can be used in a special type of JSP page called an error page. It is used for displaying the exception's name and class, its stack trace, and an informative message for the user.

A runtime exception is thrown by the compiled JSP file, which is the Java class file that contains the translated version of the JSP page. This indicates that the application has already been compiled and translated correctly.

17.1 Adding Error Pages

Even though they are referred to as error pages, the specialized JSP pages display information about exceptions.

The following steps will add error pages that display exception information to a Web application:

1. Write your Bean (or enterprise bean, servlet, or other component) so that it throws certain exceptions under certain conditions.
2. Use a simple tracking mechanism in your component to help you gather information about what your 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 your error page file or included from other files, to give your user an informative message relevant to what he or she was doing when the exception was thrown.


```
41     <table border="1">
42         <tr>
43             <th colspan="2">Request Attributes</th>
44         </tr>
45     <%
46         Enumeration attributeEnum = request.getAttributeNames();
47         while (attributeEnum.hasMoreElements()) {
48             String name = (String)attributeEnum.nextElement();
49         %>
50
51         <tr>
52             <th><%= name %> </th>
53             <td><%= request.getAttribute(name) %> </td>
54         </tr>
55
56     <%
57     }
58     %>
59 </table>
60
61 <!-- Print Parameters -->
62
63 <p>
64
65 <table border="1">
66     <tr>
67         <th colspan="2">Parameters</th>
68     </tr>
69 <%
70     Enumeration parameterEnum = request.getParameterNames();
71     while (parameterEnum.hasMoreElements()) {
72         String name = (String)parameterEnum.nextElement();
73     %>
74
75     <tr>
76         <th><%= name %> </th>
77         <td><%= request.getParameter(name) %> </td>
78     </tr>
79
80 <%
81     }
82     %>
83 </table>
84
85 <p>
86 <a href="calc.html">Restart</a>
87
88 </body>
89 </html>
90
```

18.2 Calculator

```
1   <html>
2   <head>
3   <title>Calculator</title>
4   </head>
5   <body>
6
7   <h1>Our Calculator</h1>
8
9   <form method="POST" action="output.jsp" >
10  <table border="1">
11    <tr>
12      <th colspan="2">Next Operation</th>
13    </tr>
14    <tr>
15      <th align="left">Operand 1</th>
16      <td><input type="text" size="10" name="operand1"></td>
17    </tr>
18    <tr>
19      <th align="left">Operation</th>
20      <td><input type="radio" checked name="operation"
21        value="Addition">Addition<br>
22        <input type="radio" name="operation"
23        value="Subtraction">Subtraction<br>
24        <input type="radio" name="operation"
25        value="Multiplication">Multiplication<br>
26        <input type="radio" name="operation"
27        value="Division">Division<br>
28        <input type="radio" name="operation" value="Modulo">Modulo</td>
29    </tr>
30    <tr>
31      <th align="left">Operand 2</th>
32      <td><input type="text" size="10" name="operand2"></td>
33    </tr>
34    <tr>
35      <td align="center" colspan="2"><input type="submit"
36        value="Submit"></td>
37    </tr>
38  </table>
39  </form>
40
41  </body>
42  </html>
```

18.3 Calculator JSP

```
1      <%@ page errorPage="error.jsp" %>
2
3      <html>
4      <head>
5      <title>Calculator Output</title>
6      </head>
7      <body>
8
9      <jsp:useBean id="calc" scope="page"
10         class="com.wrox.projsp.ch06.Calculator" />
11      <jsp:setProperty name="calc" property="*" />
12
13      <table border="1">
14         <tr>
15             <th colspan="2">Last Operation</th>
16         </tr>
17         <tr>
18             <th>Operand 1</th>
19             <td><jsp:getProperty name="calc" property="operand1" /> </td>
20         </tr>
21         <tr>
22             <th>Operand 2</th>
23             <td><jsp:getProperty name="calc" property="operand2" /> </td>
24         </tr>
25         <tr>
26             <th>Operation</th>
27             <td><jsp:getProperty name="calc" property="operation" /> </td>
28         </tr>
29         <tr>
30             <th>Solution</th>
31             <td><jsp:getProperty name="calc" property="solution" /> </td>
32         </tr>
33     </table>
34
35     <p>
36     <a href="calc.html">Restart</a>
37
38 </body>
39 </html>
```