

Java Server Pages

What is *Java Server Pages (JSP)*?

- HTML or XML pages with embedded Java code to generate dynamic contents.
- a text-based document that describes how to process a request and to generate a response
- a mix of template data in HTML/XML with some dynamic actions in Java

Relation with servlet:

- servlet: HTML/XML in Java code
- JSP: Java code in HTML/XML, i.e., *inverted servlet*.

Java Server Pages (cont'd)

JSP's are compiled (by a JSP compiler) to servlets.

- the first time a JSP is invoked, or
- precompiled

JSP's are processed on the server side.

- requires a *JSP container*, usually a part of a servlet container
- clients receive the processed results of JSP's, not the source

JSP versions:

- Current version: JSP 1.1
- Supported by Tomcat 3.2
- JSP 1.2 spec is available for public review.

Advantages of JSP

- ♦ **Powerful**
 - ♦ JSP's may utilize the full power of Java language and libraries.
- ♦ **Efficient**
 - ♦ JSP's are compiled to bytecode then executed, not interpreted.
- ♦ **Portable**
 - ♦ JSP's are platform independent and J2EE standard.
- ♦ **Convenient**
 - ♦ Using *Java Beans* component architecture.
- ♦ **Flexible, extensible, elegant**
 - ♦ Tag extensions with *custom tag libraries*.

Application Models

A web app often consists of both servlets and JSP's

- JSP: presentation heavy
- Servlet: mixed presentation and logic
- Java beans: pure business logic

JSP can interact with:

- JDBC
- server side Java beans
- Enterprise Java Beans (EJB)
- XML, XSLT, etc.

Hello Servlet Revisited

```
public class HelloHTML extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println(
            "<head><title> Hello </title></head>");
        out.println("<body>");
        out.println("<h1> Hello! </h1>");
        out.println(
            "The time is <i>" + new Date() + "</i>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Hello in JSP

```
<html>
<head><title> Hello </title></head>
<body>
<h1> Hello! </h1>
The time is <i>
    <%= new java.util.Date() %></i>
</body>
</html>
```

Deploy and Invoke JSP

- Deploying and invoking JSP is similar to plain HTML.
- Deploying JSP:
 - JSP files can be placed anywhere under the *<doc root>* of the web app context.
 - Do not copy JSP to **WEB-INF/classes**
 - Example: copy **Hello.jsp** to *<tomcat home>/webapps/ROOT/*
- Invoking JSP
 - Use URI relative to the *<doc root>* of the context
 - Example:
http://localhost:8080/Hello.jsp

Scripting Elements of JSP

HTML/XML Comment

- Creates a comment that is sent to the client in the viewable page source.

```
<!-- comment [ <%= expression %> ] -->
```

Hidden Comment

- Documents the JSP file, but is not sent to the client.

```
<%-- comment --%>
```

Declaration

- Declares variables or methods in Java

```
<%! declaration; [ declaration; ]+ ... %>
```

```
<jsp:declaration>  
    declaration; [ declaration; ]+ ...  
</jsp:declaration>
```

Scripting Elements of JSP (cont'd)

Expression

- Contains a Java expression

```
<%= expression %>
```

```
<jsp:expression>  
  expression  
</jsp:expression>
```

Scriptlet

- Contains a Java code fragment

```
<% code fragment %>
```

```
<jsp:scriptlet>  
  code fragment  
</jsp:scriptlet>
```

Quoting in JSP

In scripting elements:

- A literal `%>` is quoted as `%\>`

In template text:

- A literal `<%` is quoted as `<\%`

In attributes:

- A `'` is quoted as `\'`
- A `"` is quoted as `\"`
- A `\` is quoted as `\\`
- A `%>` is quoted as `%\>`
- A `<%` is quoted as `<\%`

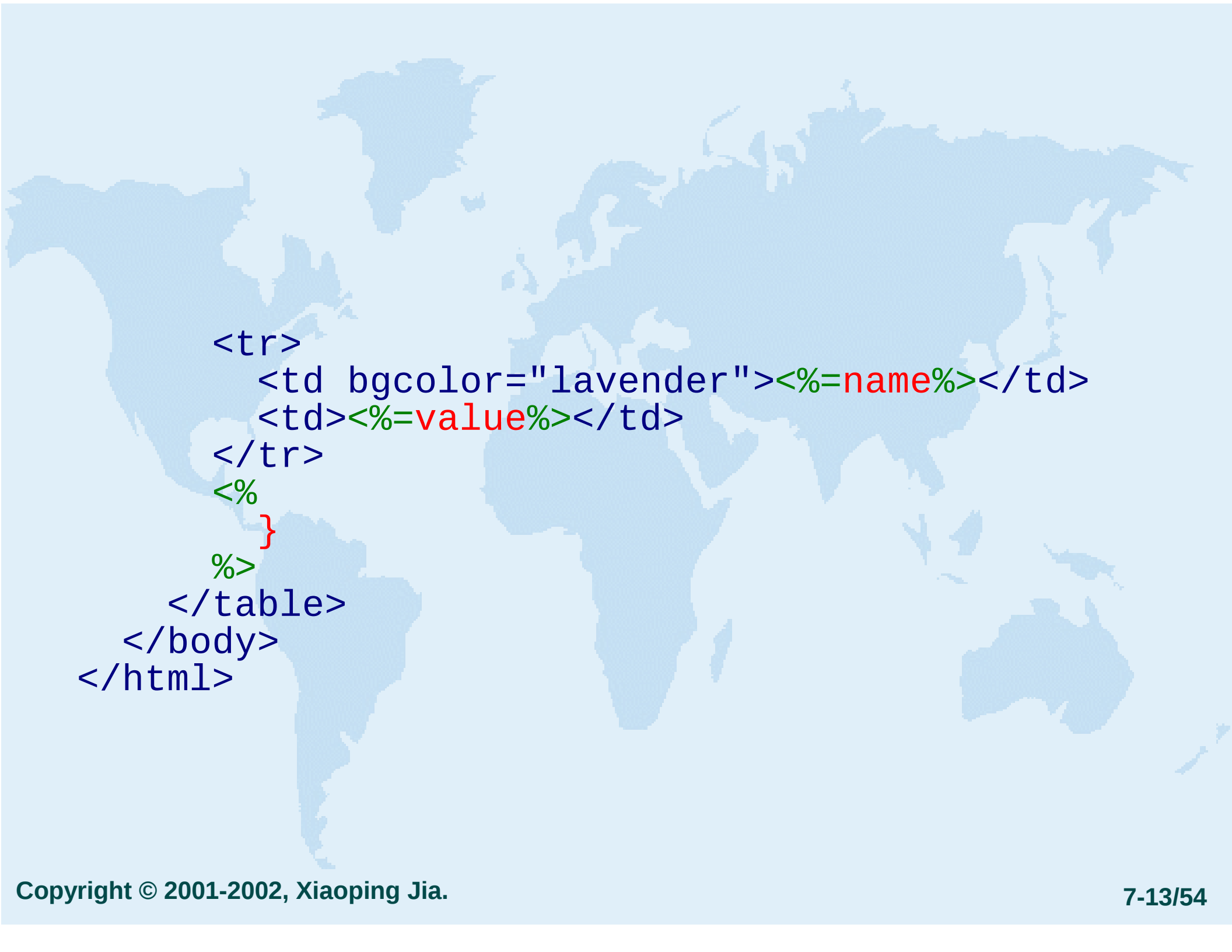
Request Info in JSP

```
<html>
<head><title>JSP Request Info</title></head>
<body>
  <table>
    <tr>
      <td bgcolor="lavender">Content Length</td>
      <td><%=String.valueOf(
        request.getContentLength())%></td>
    </tr>
    <tr>
      <td bgcolor="lavender">Content Type</td>
      <td><%=request.getContentType()%></td>
    </tr>
  </table>
</body>
</html>
```

Request Headers in JSP

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

<html>
  <head><title>JSP Request Header</title></head>
  <body>
    <!-- A Java Server Page:
    list all the request headers -->
    <table>
      <%-- A scriptlet:
      iterate through the enumeration object --%>
      <%
        Enumeration e = request.getHeaderNames();
        while (e.hasMoreElements()) {
          String name = (String) e.nextElement();
          String value = request.getHeader(name);
        %>
```



```
<tr>
  <td bgcolor="lavender"><%=name%></td>
  <td><%=value%></td>
</tr>
<%
  }
%>
</table>
</body>
</html>
```

Request Parameters in JSP

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

<html>
  <head><title>Parameters (JSP)</title></head>
  <body>
    <table border=0>
      <%
        Enumeration e =
          request.getParameterNames();
        while (e.hasMoreElements()) {
          String name = (String) e.nextElement();
          String values[] =
            request.getParameterValues(name);
          if (values != null) {
            for (int i = 0;
              i < values.length; i++) {
```

A faint, light blue world map is visible in the background of the slide.

```
<tr>
  <td><i><font color=blue>
      <%= name %><font></i></td>
  <td><%= values[i] %></td>
</tr>
<!-- Make sure the braces are balanced.
      (This is not pretty!) --%>
<%
    }
  }
}
%>
</table>
</body>
</html>
```

Handle Login in JSP

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

<html>
<head><title>Login</title></head>
<body>

<%!
    protected Map users = new HashMap();

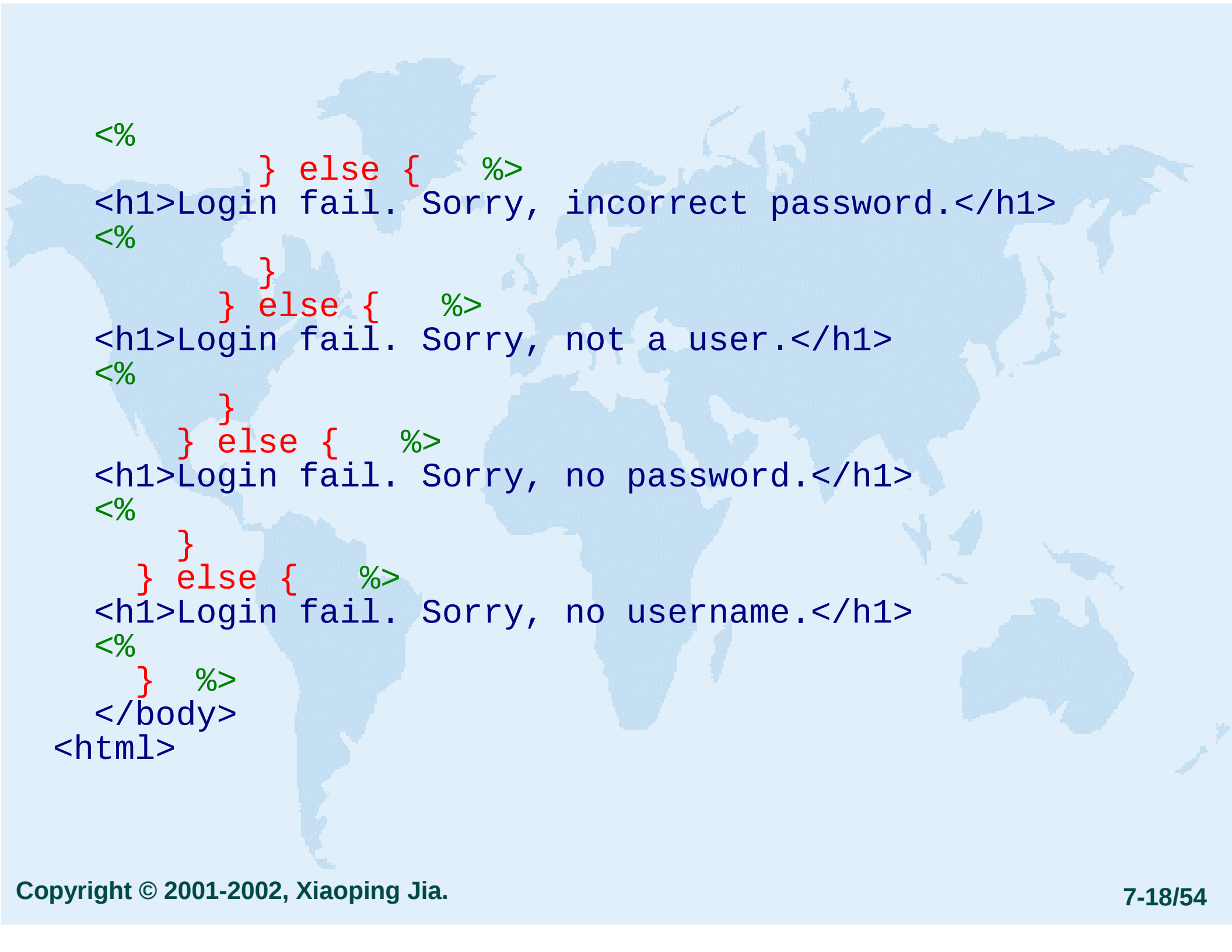
    public void jspInit() {
        // username and password
        users.put("Scott McNealy", "lavender");
        users.put("Steve Jobs", "aqua");
        users.put("Bill Gates", "blue");
    }
%>
```

<%

```
String username = request.getParameter("username");
String password = request.getParameter("password");
if (username != null)
    username = username.trim();
if (password != null)
    password = password.trim();
if (username != null &&
    username.length() > 0) {
    if (password != null &&
        password.length() > 0) {
        String pw = (String) users.get(username);
        if (pw != null) {
            if (pw.equals(password)) {
                String firstname = username;
                int i = username.indexOf(' ');
                if (i > 0)
                    firstname = username.substring(0, i);
```

%>

```
<h1>Login successful. Hello <%= firstname %>!</h1>
```



```
<%  
    } else { %>  
<h1>Login fail. Sorry, incorrect password.</h1>  
<%  
    }  
    } else { %>  
<h1>Login fail. Sorry, not a user.</h1>  
<%  
    }  
    } else { %>  
<h1>Login fail. Sorry, no password.</h1>  
<%  
    }  
    } else { %>  
<h1>Login fail. Sorry, no username.</h1>  
<%  
    } %>  
</body>  
<html>
```

Separate Presentation and Logic

Separate Java code from HTML code

- ♦ Java code responsible for logic
- ♦ HTML code responsible for presentation

Java code: the `LoginManager` class

```
public class LoginManager {  
    public static final int  
        LOGIN_SUCCESSFUL = 0;  
    public static final int  
        LOGIN_FAIL_INCORRECT_PASSWORD = 1;  
    public static final int  
        LOGIN_FAIL_NOT_A_USER = 2;  
    public static final int  
        LOGIN_FAIL_NO_PASSWORD = 3;  
    public static final int  
        LOGIN_FAIL_NO_USERNAME = 4;  
}
```

Login Manager

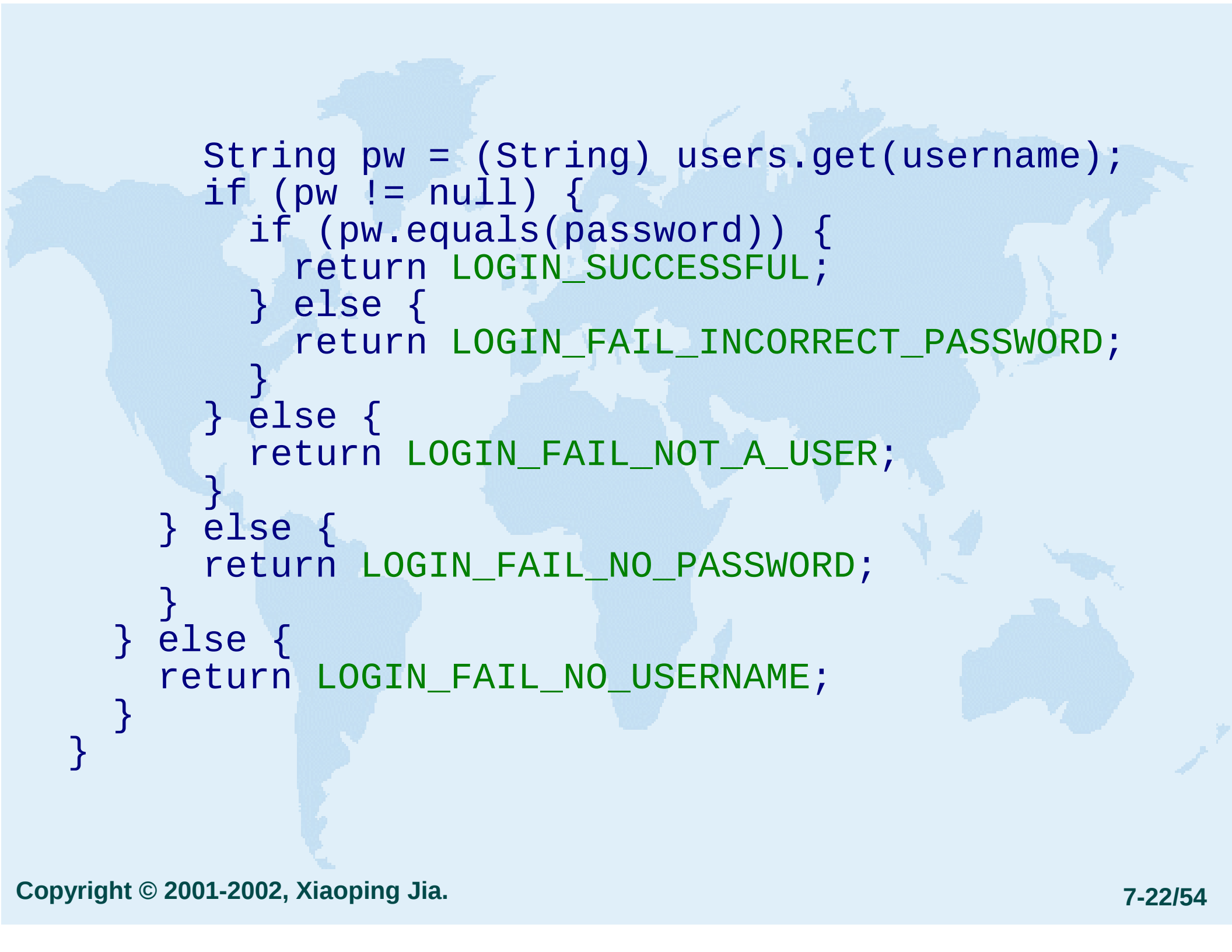
A singleton class

```
public static LoginManager getInstance() {  
    if (theInstance == null) {  
        theInstance = new LoginManager();  
    }  
    return theInstance;  
}  
  
protected LoginManager() {  
    // username and password  
    users.put("Scott McNealy", "lavender");  
    // ...  
}  
  
protected static LoginManager theInstance = null;  
protected Map users = new HashMap();
```

Login Manager (cont'd)

Validate username and password

```
public int validate(String username,  
                    String password) {  
    if (username != null) {  
        username = username.trim();  
    }  
    if (password != null) {  
        password = password.trim();  
    }  
    if (username != null &&  
        username.length() > 0) {  
        if (password != null &&  
            password.length() > 0) {
```

A faint, light blue world map is visible in the background of the slide, centered behind the code text.

```
String pw = (String) users.get(username);
if (pw != null) {
    if (pw.equals(password)) {
        return LOGIN_SUCCESSFUL;
    } else {
        return LOGIN_FAIL_INCORRECT_PASSWORD;
    }
} else {
    return LOGIN_FAIL_NOT_A_USER;
}
} else {
    return LOGIN_FAIL_NO_PASSWORD;
}
} else {
    return LOGIN_FAIL_NO_USERNAME;
}
}
```

Login Manager (cont'd)

Generate messages

```
public String getMessage(int status) {  
    switch (status) {  
        case LOGIN_SUCCESSFUL:  
            return "successful";  
        case LOGIN_FAIL_INCORRECT_PASSWORD:  
            return "incorrect password";  
        case LOGIN_FAIL_NOT_A_USER:  
            return "not a user";  
        case LOGIN_FAIL_NO_PASSWORD:  
            return "no password";  
        case LOGIN_FAIL_NO_USERNAME:  
            return "no username";  
        default:  
            return "unknown";  
    }  
}
```

Login JSP

```
<%@ page import="java.util.*,login.*" %>
<html>
  <head><title>Login</title></head>
  <body>
    <%
      String username =
        request.getParameter("username");
      String password =
        request.getParameter("password");
```

```
    LoginManager loginMgr =
        LoginManager.getInstance();
    int result =
        loginMgr.validate(username, password);
    if (result == LoginManager.LOGIN_SUCCESSFUL) {
%>
<h1>Login successful. Hello <%= username %>!</h1>
<%
    } else {
%>
<h1>Login fail. Sorry,
    <%= loginMgr.getMessage(result) %>.</h1>
<%
    }
%>
</body>
<html>
```

Compiling JSP

- Servlets
 - Precompiled.
 - Class files in `<doc root>/WEB-INF/classes`.
 - Servlet engine appends this path to the *servlet engine classpath* for each web app.
- JSP
 - Compiled the first time it is invoked
 - JSP compiler: compile JSP to servlet
 - Java compiler: compile servlet to bytecode
 - Class files in `<doc root>/WEB-INF/classes`.

CLASSPATH for Running JSP

- Locations for extra classes used by JSP, including Java beans, must be added to the servlet engine classpath.
- The servlet engine classpath is set before the servlet engine starts up.
- In Tomcat 3.2
 - Modify `<tomcat home>/bin/tomcat.bat` to add the extra path.
 - Set the extra path in the **CLASSPATH** environment variable.
 - Do not include jars from JDK/JRE in the **CLASSPATH** environment variable.

Scope of Objects in JSP

- Objects can be manipulated in JSP.
- Objects exist in different scopes:
 - *page* scope: a single page
 - *request* scope: pages processing the same request
 - *session* scope: pages processing requests that are in the same session
 - *application* scope: all pages in an application
- A object with a given scope is accessible to pages that are in the same given scope as the page where the object was created.

Implicit Objects

In JSP, certain *implicit objects* are always available for use within scriptlets and expressions, without being declared first.

The implicit objects in JSP are:

- request
- response
- pageContext
- session
- application
- out
- config
- page
- exception

Implicit Object: Request

The **request** object

- Type: `HttpServletRequest` or a subclass of `ServletRequest`
- Scope: request scope
- Objects of request scope are stored in this object.
- Some useful methods:
 - `getAttribute(name)`
 - `getParameter(name)`
 - `getParameterNames()`
 - `getParameterValues(name)`

Implicit Object: Response

The **response** object

- Type: `HttpServletResponse` or a subclass of `ServletResponse`
- Scope: page scope
- Not typically used in JSP pages.

Implicit Object: `pageContext`

The `pageContext` object

- Type: `javax.servlet.jsp.PageContext`
- Scope: page scope
- Objects of page scope are stored in this object.
- Some useful methods:
 - `findAttribute(name)`
 - `getAttribute(name)`
 - `getAttributesScope(name)`
 - `getAttributeNamesInScope()`

Implicit Object: session

The **session** object

- Type: `HttpSession`
- Scope: session scope
- Objects of session scope are stored in this object.
- Some useful methods
 - `getId()`
 - `getValue(name)`
 - `getValueNames()`
 - `putValue(name, obj)`

Implicit Object: application

The **application** object

- Type: `ServletContext`
- Scope: application scope
- Objects of application scope are stored in this object.
- Some useful methods:
 - `getMimeType()`
 - `getRealPath()`
 - `getAttribute(name)`
 - `getAttributeNames()`

Implicit Object: out

The **out** object

- Type `javax.servlet.jsp.JspWriter`, a subclass of `PrintWriter`
- Scope: page scope
- Some useful methods:
 - `clear()`
 - `clearBuffer()`
 - `flush()`
 - `getBufferSize()`
 - `getRemaining()`

Implicit Object: config

The **config** object

- Type: `ServletConfig`
- Scope: page scope
- Some useful methods:
 - `getInitParameter(name)`
 - `getInitParameterNames()`

Implicit Object: page

The **page** object

- Type: **Object**
- Scope: page scope
- The **this** reference to the servlet generated from the current JSP page.
- Not typically used in JSP.

Implicit Object: exception

The **exception** object

- Type: **Throwable**
- Scope: page scope
- Available only in error pages.
- Some useful methods:
 - **getMessage()**
 - **getLocalizedMessage()**
 - **printStackTrace()**
 - **toString()**

JSP Directives

- Directives are messages or instructions to the JSP container.
- Directives do not produce any output.
- JSP 1.1 standard directives:
 - `page`
 - `include`
 - `taglib`
- Directives have the following syntax in JSP 1.1

```
<%@ directive
    attr1 = value1
    attr2 = value2
    ...
%>
```

Page Directive

Defines attributes that apply to an entire JSP page.

```
<%@ 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" ] %>
```

Page Directive Attributes

contentType

- The character encoding and the MIME type of the response page.
- Default MIME type: `text/html`
- Default character set: `ISO-8859-1`

language

- The scripting language.
- JSP 1.1: must be `java`.

extends

- The fully qualified *class name* of the superclass.
- The specified class must implement the `HttpJspPage` interface for HTTP protocol, or the `JspPage` interface for non-HTTP protocols.
- Use this attribute with care.

Page Directive Attributes (cont'd)

import

- A comma separated import declaration list consisting of either
 - a fully qualified *class name*, or
 - a *package name* followed by `.*`
- The default import list is

```
java.lang.*  
javax.servlet.*  
javax.servlet.jsp.*  
javax.servlet.http.*
```

Page Directive Attributes (cont'd)

session

- Indicates whether the page requires participation in an HTTP session.
- The **session** implicit variable is available if and only if this attribute is **true**.
- Default: **true**

isThreadSafe

- **false**: requests are processed one at a time, in the order they were received
- **true**: requests are processed simultaneously.
 - Page authors must ensure that they properly synchronize access to page shared state.
- Default: **true**

Page Directive Attributes (cont'd)

buffer

- Specifies the buffering model for the initial **out** implicit variable (JspWriter).
- **none**: no buffering
- Default: buffer size no less than **8kb**.

autoFlush

- **true**: the buffered output is flushed automatically.
- **false**: an exception should be raised when the buffer is filled, to indicate buffer overflow.

Page Directive Attributes (cont'd)

errorPage

- a URL to JSP for error processing
- The error page is invoked when Throwable object is thrown but not caught.

isErrorPage

- Indicates if the current JSP page is an error page.
- Default: **false**
- The **exception** implicit variable is available if and only if this attribute is **true**.

info

- Defines an arbitrary string that can be retrieved by the `Servlet.getServletInfo()` method.

JSP Actions

- JSP *actions* are special tags that may affect the current out stream and use, modify and/or create objects.
- Standard actions
 - `<jsp:forward>`
 - `<jsp:include>`
 - `<jsp:param>`
 - `<jsp:plugin>`
 - `<jsp:useBean>`
 - `<jsp:getProperty>`
 - `<jsp:setProperty>`
- Custom actions
 - Custom tag extensions, taglib

Including Files in JSP

- Include directive
 - Includes a static or JSP page
 - Processed at translation time
 - Syntax:

```
<%@ include file="relativeURL" %>
```
- Include action
 - Includes a static page, or
 - Sends a request to a JSP or servlet, and includes the result page, which must be a static page.
 - Processed at request time.
 - An included page only has access to the **out** (**JspWriter**) object, and it cannot set headers.

Syntax of Include Action

```
<jsp:include  
  page="{relativeURL | <%= expression %>}"  
  flush="true" />
```

```
<jsp:include  
  page="{relativeURL | <%= expression %>}"  
  flush="true" >  
  [ <jsp:param name="name"  
    value="{value | <%= expression %>}"  
    /> ]+  
</jsp:include>
```

Forward Action

- Forward action
 - Forwards a client request to a static page, JSP page, or servlet for processing.
 - Terminates the execution of the current page.
 - If the page output (the **out** object) is buffered then the buffer is cleared prior to forwarding.
 - If the page output (the **out** object) is unbuffered and anything has been written to it, an attempt to forward the request will result in an Exception.

Syntax of Forward Action

```
<jsp:forward  
  page="{relativeURL | <%= expression %>}"  
  flush="true" />
```

```
<jsp:forward  
  page="{relativeURL | <%= expression %>}"  
  flush="true" >  
  [ <jsp:param name="name"  
    value="{value | <%= expression %>}"  
    /> ]+  
</jsp:forward>
```

The JSP Container

- ♦ JSP containers manage the life cycles of JSP pages.
- ♦ JSP containers interact with JSP pages through the `javax.servlet.jsp.HttpJspPage` interface, which
 - ♦ extends `javax.servlet.jsp.JspPage`, which
 - ♦ extends `javax.servlet.Servlet`
- ♦ Methods:
 - ♦ `void jspInit()`
 - ♦ `void jspDestroy()`
 - ♦ `void _jspService(HttpServletRequest request, HttpServletResponse response)`

The JSP Container (cont'd)

- JSP pages may define the `jspInit()` and `jspDestroy()` methods in declarations.

```
<%!  
    public void jspInit() { ... }  
    public void jspDestroy() { ... }  
%>  
<html>  
    ...  
</html>
```

- JSP pages should not define the `_jspService()` method.
- JSP pages should not define any of the servlet methods.

Generated Servlet

A skeleton of servlets generated from JSP pages

```
import package.name;  
  
class _jspXXX extends SuperClass {  
    <Declaration Section>  
  
    public void _jspService(...) {  
        <Implicit Objects Section>  
        <Main Section>  
    }  
}
```

Translating the JSP Elements

- A declaration:
 - Verbatim copy to the *<Declaration Section>*
- Template data:
 - Statement fragment in the *<Main Section>*
`out.print(template);`
- A scriptlet:
 - Verbatim copy to the *<Main Section>*
- An expression:
 - Statement in the *<Main Section>*
`out.print(expresion);`
- An action:
 - Statement fragment in the *<Main Section>* to
 - declare and create objects, and
 - invoke the action handler.