

# CSBP 46 I

# Internet Computing:

## Java Server Pages (JSP)

**M. Elarbi Badidi**

# Objectives

- Introduce JSP Web components
- Compare JSP with Servlets
- Present the JSP life cycle
- Introduce JSP scripting elements
- Discuss JSP standard actions
- Discuss Custom Tags Library
- Introduce the Expression Language

# Java Server Pages (JSP)

- The Java Server Pages (JSP) technology allows Web developers to generate **static** and **dynamic** HTTP contents. The JSP is just like the ordinary Web pages with some additional embedded Java code. For example, a simplest JSP page can just be a simple HTML page with a .JSP extension.

**<HTML>**

**<BODY>**

**<H1> This is a simplest JSP page <H1>**

**</BODY>**

**<HTML>**

- This is the simplest JSP page, without any embedded Java code. It can be treated just like an HTML page.

# A Simple JSP Page

(Blue: static, Red: Dynamic contents)

```
<html>
```

```
<body>
```

```
  Hello World!
```

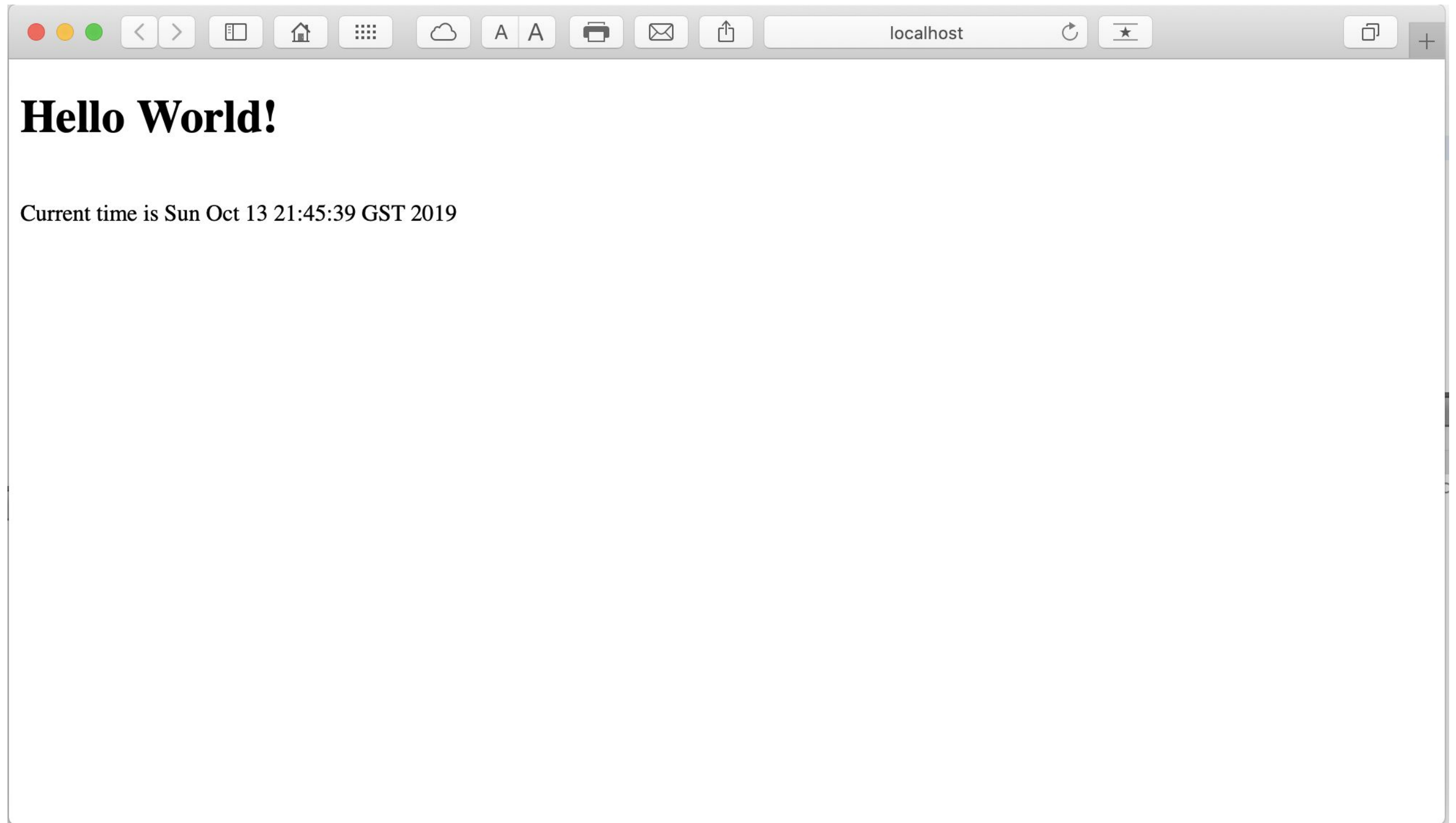
```
  <br>
```

```
  Current time is <%= new java.util.Date() %>
```

```
</body>
```

```
</html>
```

# Output



# Servlet vs. JSP

## Servlets

- HTML code in Java
- Not easy to author

## JSP

- Java-like code in HTML
- Very easy to author
- Code is compiled into a servlet

# JSP Benefits

- Content and display logic are separated
- Simplify web application development with JSP, JavaBeans and custom tags
- Supports software reuse through the use of components (JavaBeans, Custom tags)
- Automatic deployment
  - Recompile automatically when changes are made to JSP pages
- Easier to author web pages
- Platform-independent

# Why JSP over Servlet?

- **Servlets can do a lot of things, but it is pain to:**
  - Use those `println()` statements to generate HTML page
  - Maintain that HTML page
- **Although JSP technically can't do anything servlets can't do, JSP makes it easier to:**
  - Write HTML
  - Read and maintain the HTML
- **JSP makes it possible to:**
  - Use standard HTML tools such as Macromedia DreamWeaver or Adobe GoLive.
  - Have different members of your team do the HTML layout than do the Java programming
- **JSP encourages you to**
  - Separate the (Java) code that creates the content from the (HTML) code that presents it



# Java Server Pages (JSP) (cont.)

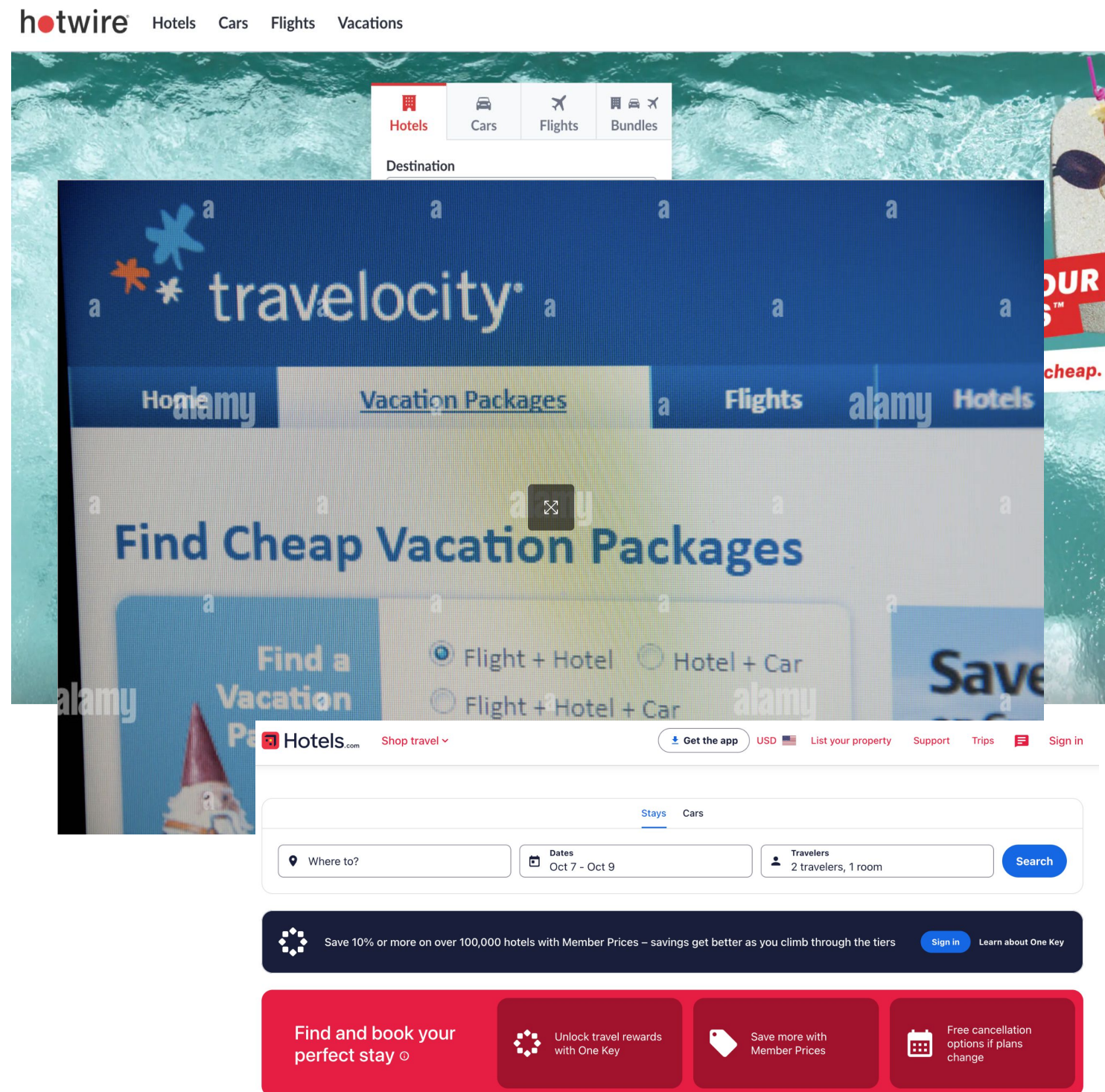
- Most of JSP pages have embedded Java code. This allows access to server-side data and implements business logic.
- **Java Server Pages are an extension of Java Servlet technology.** JSP can not work without Servlet container support.
- When a Web server gets a JSP request from a client the **JSP engine converts the JSP page into a Java Servlet code** and compiles it into Servlet classes.
- The generated Servlet code is run on the Servlet container and finally sends the results back to the Web client.
- **JSP focuses on the data presentation** while the other Web components such as Servlets, JavaBeans, custom tags, and EJB take care of business logic processing.

# Java Server Pages (JSP) (cont.)

- JSP also makes **the tag library** extensible so that developers can develop their own reusable custom tags.
- The new features of JSP 2.0 such as **Expression Language** (EL) make Web server page authoring even easier. JSP provides a more consistent way for Web developers to access different types of data such as implicit server objects, JavaBeans, Servlets, and even other remote distributed data.

# JSP/Servlets in the Real World: Travel Sites

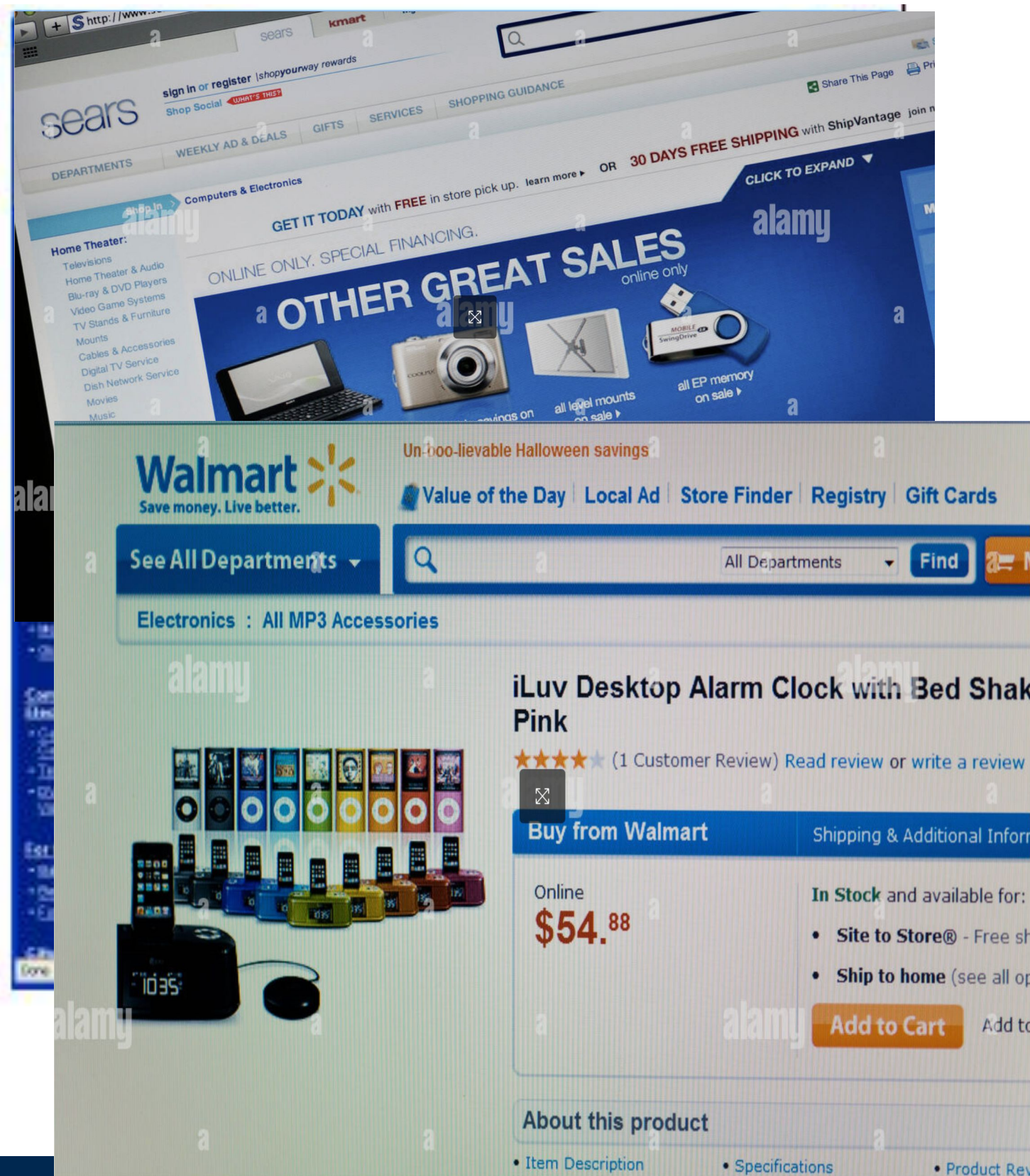
- Travelocity.com
- Orbitz.com
- HotWire.com
- Hotels.com
- CheapTickets.com
- National Car Rental
- Avis Car Rental
- Enterprise Car Rental
- Hertz Car Rental





# JSP/Servlets in the Real World: Retail

- **Sears.com**
- **Walmart.com**
- **HomeDepot.com**
- **SamsClub.com**
- **Macys.com**
- **lbean.com**
- **Kohls.com**
- **Ikea.com**
- **Target.com**
- **Longaberger.com**
- **Nike.com**
- **CircuitCity.com**



# JSP/Servlets in the Real World: Search/Portals

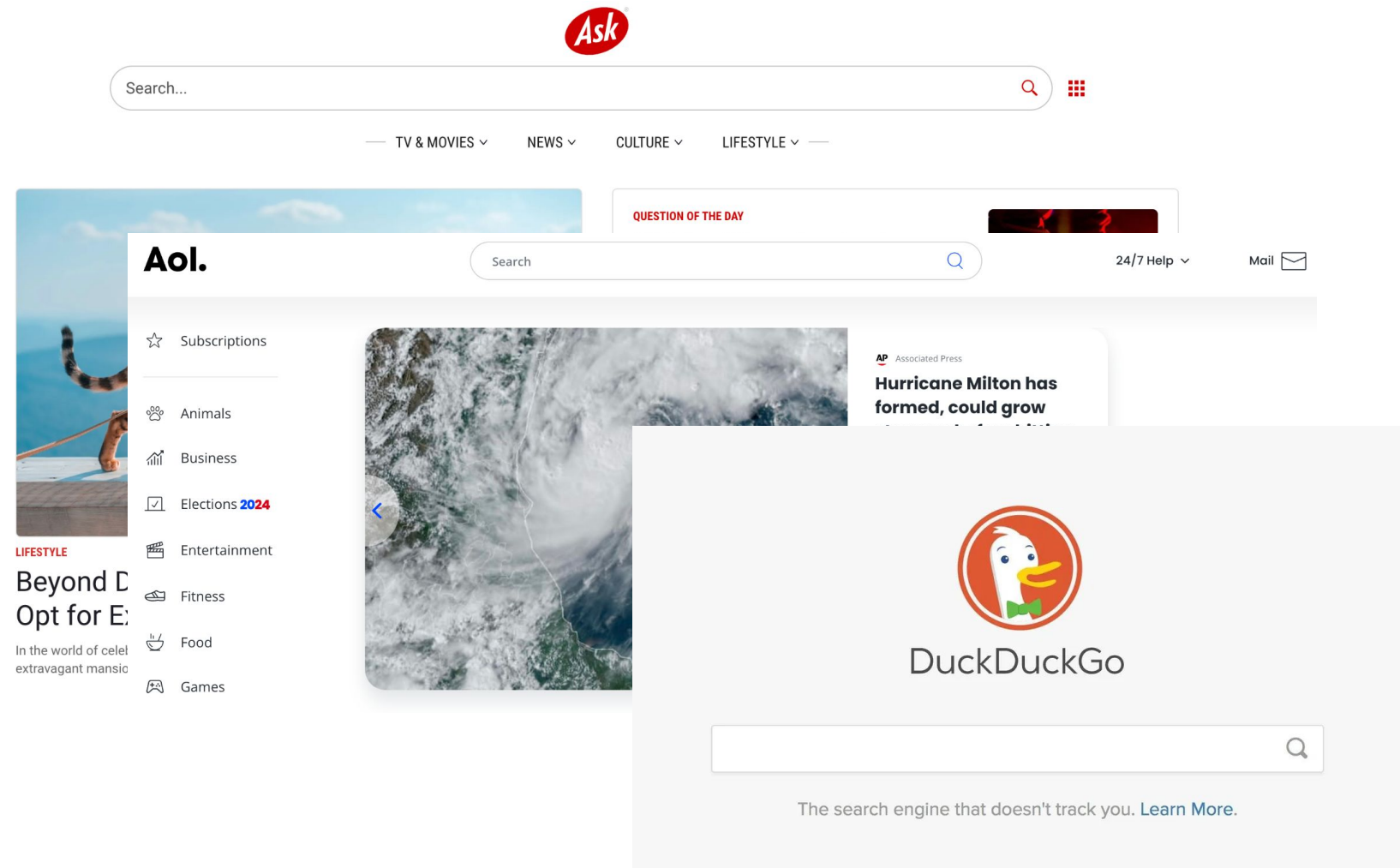
[www.ask.com](http://www.ask.com)

[www.aol.com](http://www.aol.com)

[www.duckduckgo.com](http://www.duckduckgo.com)

[www.yahoo.com](http://www.yahoo.com)

[www.ecosia.org](http://www.ecosia.org)





# JSP Basics

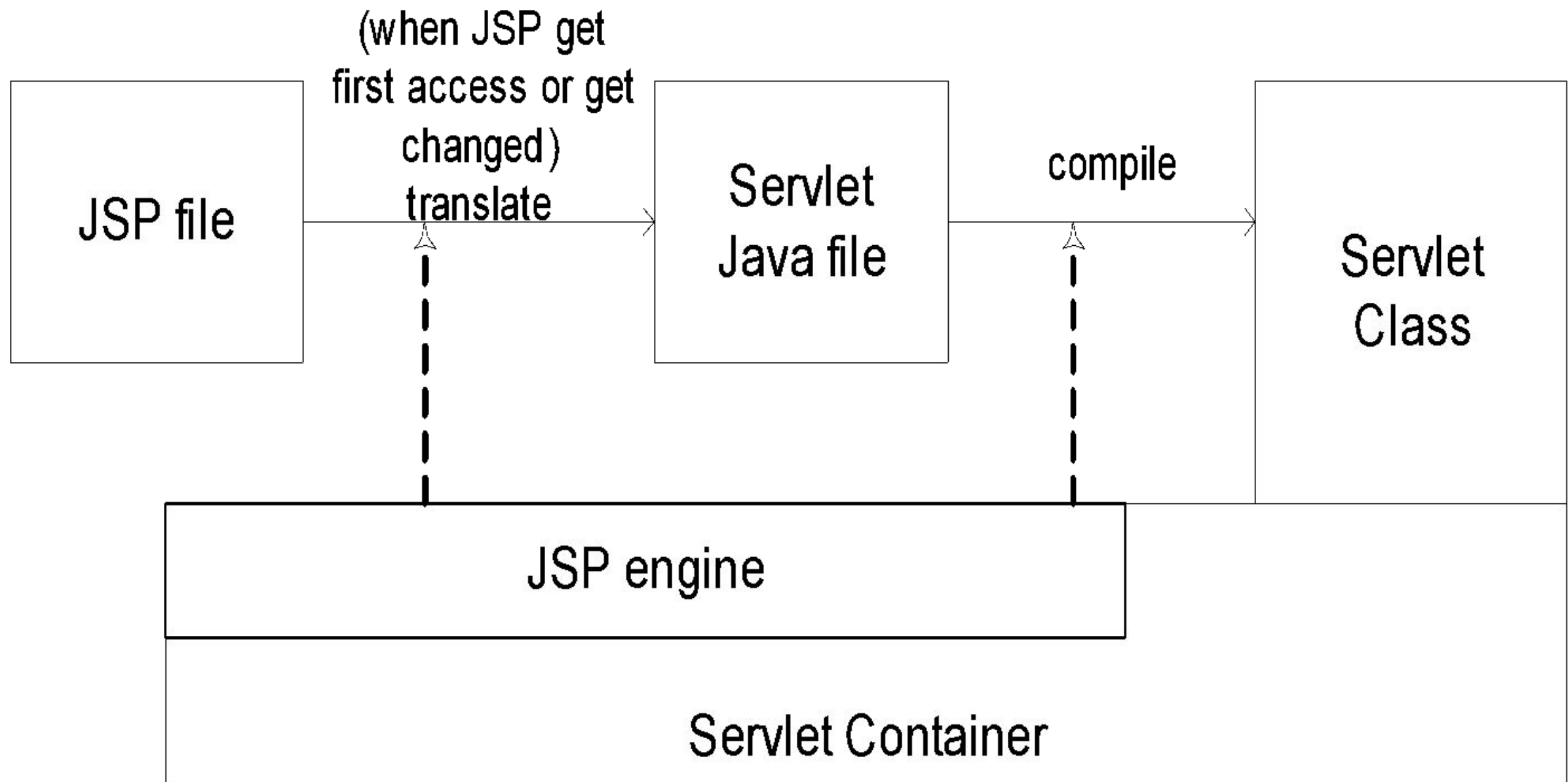
## JSP Life Cycle

- JSP is a Web server side programming technology based on Servlet technology. The JSP specification is built on the top of the Servlet API.
- Any JSP page is translated into a Java Servlet by a JSP engine at runtime so that the JSP life cycle is determined by Servlet technology.
- The JSP engine is a JSP specification implementation which comes with web servers that implement JSP. Tomcat is one example.
- When a request comes to a JSP page, it may come from client browser or come from another Web component such as a Servlet or JSP.

# JSP Life Cycle (cont.)

- The Web server asks the JSP engine to check whether the JSP page has never been accessed before, or it has been modified since its last access. If this is the case the JSP engine will
  1. **Parse the JSP document to translate into a Servlet Java file**
  2. **Compile the Servlet Java file into a class file.**
- Then the Servlet container loads the Servlet class for execution and sends the results back to the client.

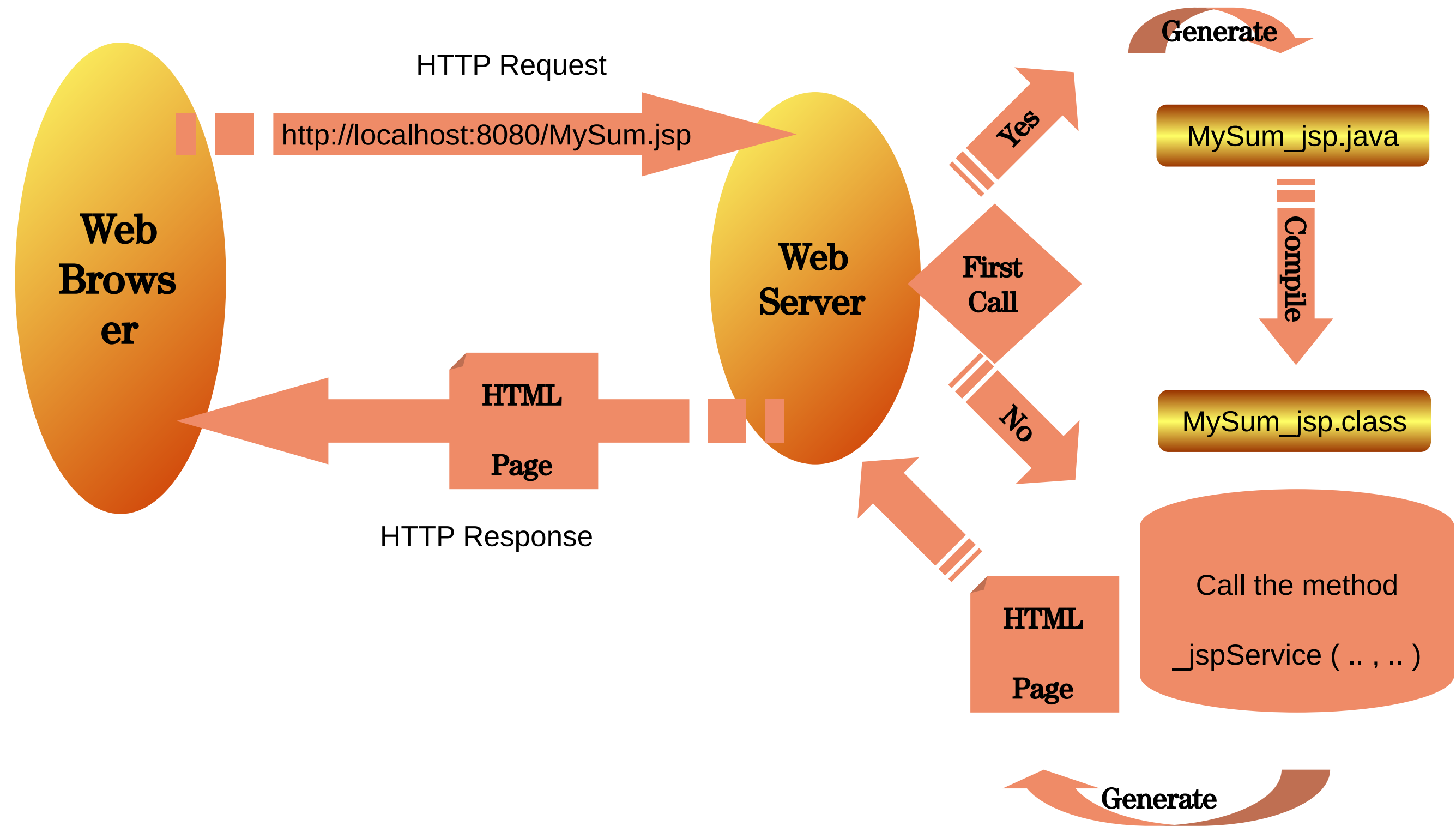
# JSP Life Cycle (cont.)



**Process of a JSP page**



# The JSP Page Life Cycle



# JSP Life Cycle (cont.)

- **JSP initialization**

- The **jspInit()** method is executed first after the JSP is loaded.
- If the JSP developer needs to perform any JSP-specific initialization such as database connections at the beginning this method can be specified.
- **<%! ... %>** is a JSP declaration element which is used to declare variable or methods. The jspInit() is only called once during any JSP component life time.
- **<%! public void jspInit(){ ... } %>**

# JSP Life Cycle (cont.)

- **JSP execution**

- Public `_service` (`HttpServletRequest req`, `HttpServletResponse res`)
- “\_service” is a JSP service method the same as the `service()` method of a Servlet class. This method never needs to be customized. All Java code defined in scripting elements are inserted in this method by JSP engine.

- **JSP termination**

- `<%! public void jspDestroy() { ... } %>`
- This method allows developers to specify resource cleanup jobs such as database disconnections.

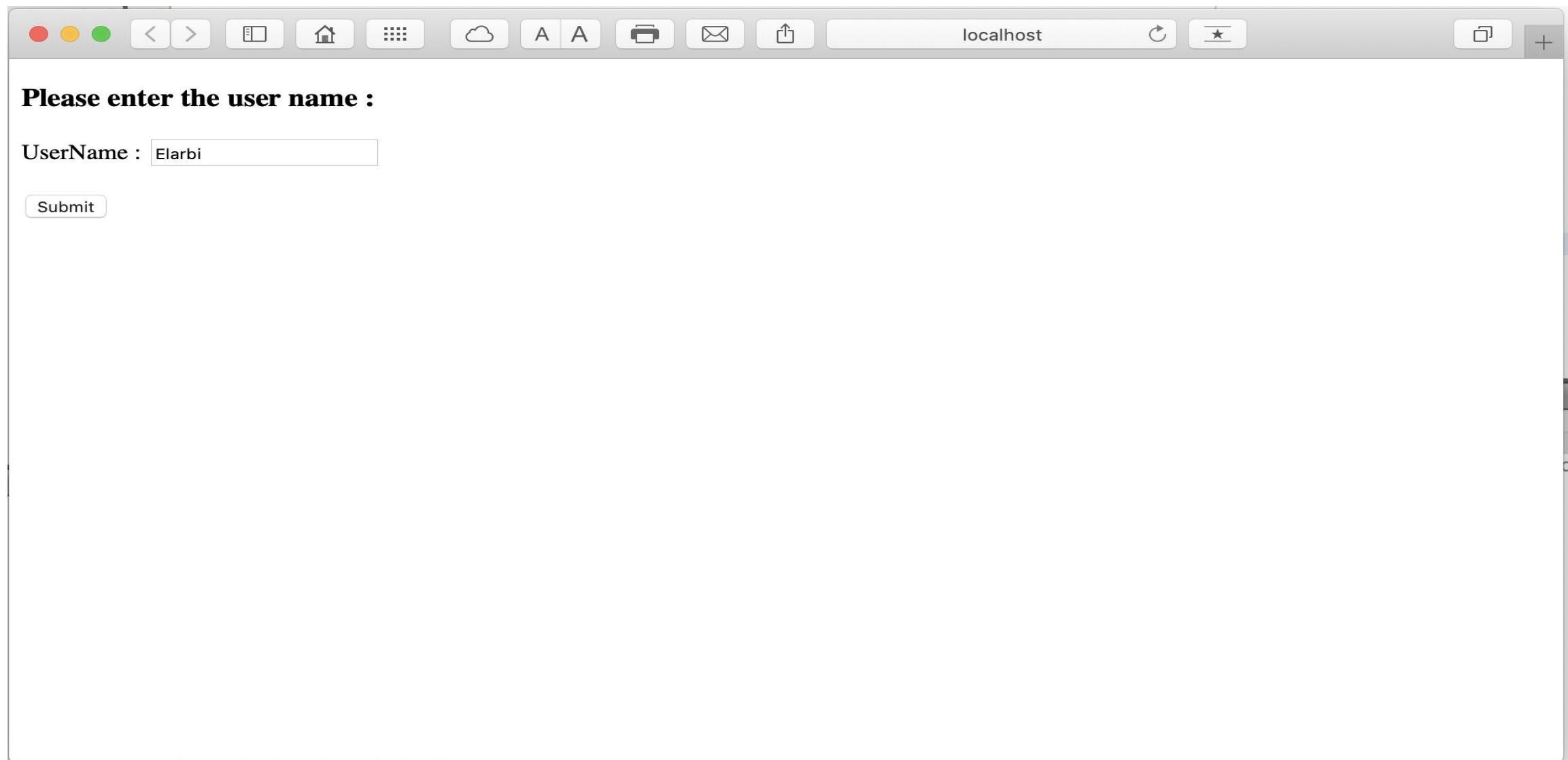
# First Simple Interactive JSP example (cont.)

```
<html>
<head>
  <title>Demo1</title>
</head>
<body>
<h3>Please enter the user name :</h3><p>
<form action="/jsp/helloJsp.jsp">
  UserName : <input type="text" name="userName"><br><br>
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

Basically, this JSP page takes a user name input from a Web client and generates a dynamic page to welcome the user.

# First Simple Interactive JSP example (cont.)

- This HTML takes a string of a user name from the HTML form and submits a request to *hellojsp.jsp* as specified in the action attribute of the HTML form. For example, a user types SPSU in the form and pushes the Submit button.

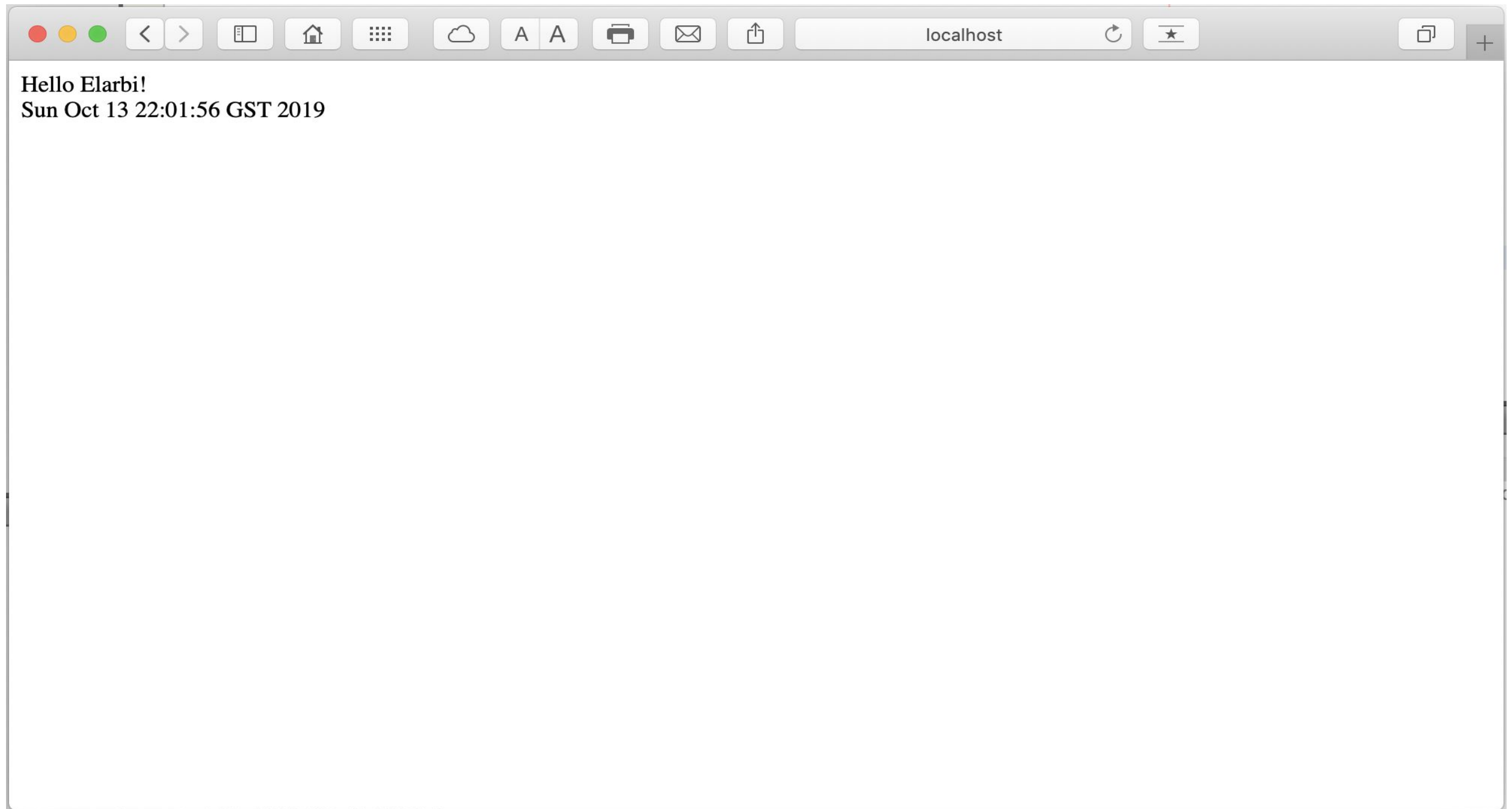


A screenshot of a web browser window. The address bar shows 'localhost'. The page content includes the text 'Please enter the user name :', a text input field with 'Elarbi' entered, and a 'Submit' button.

Please enter the user name :

UserName :

# First Simple Interactive JSP example (cont.)



# First Simple Interactive JSP example (cont.)

- The page *hellojsp.jsp* is placed in the same directory as *index.html* and *WEB-INF*. All HTML tags can be used in a JSP page since JSP is an extension of HTML. A JSP file can be placed anywhere an HTML file can be placed.

```
<%@ page import="java.util.*" info = " This is a simplest JSP
  with Java code embedded" contentType = "text/html" %>
<%! Date today; %>
<html>
    <!-- This is a Simplest JSP (comment from JSP tag -- %>
<body>
    <!-- This is a simplest JSP (Comment from HTML tag -->
    Hello <%= request.getParameter("userName") %>!
<br>
    <% today = new Date(); %>
    <%= today %>
</body>
</html>
```

# JSP Syntax Elements

## Directives

```
<html>
  <body>
    {
      <%@ page language      = "java"           %>
      <%@ page import        = "java.util.*"      %>
      <%@ page session       = "true"             %>
      <%@ page errorPage     = "error.html"        %>
      <%@ page contentType  = "text/html; charset=ISO-8859-1" %>
      <%@ include file      = "welcome.html"      %>
    }
```

## Declarations

```
•---> <%! int count = 0 ; %>
```

## Scriptlets

```
•---> <% count++ ; %>
```

## Expressions

```
•---> <%= count %>
```

## Comments

```
•---> <%-- JSP Comments --%>
```

```
  </body>
</html>
```



# First Simple Interactive JSP example (cont.)

- The first line is a JSP directive element tag (`<%@ ... %>`) which directs the JSP engine to set the page structure. It will not result in any Servlet code, just like the import directive in Java.
- This directive element tells the JSP engine to import all classes in the *java.util* package to allow use of the Date class and tells JSP engine that content type of this page is *text/html* not *text/xml*.
- The second line is a JSP declaration element `<%! ... %>` which tells the JSP engine to insert the enclosed java code into the generated Servlet java source code somewhere outside of any method.
- `<%-- ... %>` is a JSP comment scripting element which stays on the server while `<!-- ... -->` is a regular HTML comment which is sent back to the client.

# First Simple Interactive JSP example (cont.)

- The expression scripting element `<%= . . . %>` will display values of the enclosed expression in string format. In this JSP example, the *hellojsp.jsp* gets the parameter input string by the `getParameter()` method of `HttpServletRequest` object.
- The input string either comes from the input text field of the request form or the URL query string of this JSP address. The “*userName*” of the `getParameter()` method matches the “*userName*” text input component name in the HTML form.
- The next JSP scripting element is **scriptlet** (java servelt code) which is insered into the `service()` method of the generated Servlet code JSP instantiates a new instance of `Date` class and the current date is displayed in the next expression scripting element.

# Directives

➤ Directives provide general information about the JSP page to the JSP engine.

● `<%@ page language = "java" %>`

Informs the JSP engine that we will be using **java** as the scripting language in our JSP page. As of JSP 1.2, java is **the only one** allowed.

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

Imports **classes** and **packages** that we want to use in the JSP page.

## example

```
<html>
  <body>
    <%@ page import="java.util.*" %>
    ...
    <%
    ...
      LinkedList list = new LinkedList();
    ...
    %>
    ...
  </body>
</html>
```

# Directives

- `<%@ page session = "true" %>`

Specifies whether the JSP page takes part in an HTTP session (more details will be given in the Implicit Variables Section).

- `<%@ page errorPage = "error.html" %>`

Used to delegate the exceptions to another HTML or JSP page.

## example

```
<html>
  <body>
    <%@ page errorPage="error.html" %>
    ...
    <%
      ...
      inti = Integer.parseInt ( "1s7" );
      ...
    %>
    ...
  </body>
</html>
```

# Directives

● `<%@ page contentType = "text/html; charset=ISO-8859-1" %>`

Specifies the **content type** and **Character Encoding** for the output.

## example

```
<%@ page contentType="application/vnd.ms-excel" %>
```

```
This is a Test
1  2  3  4
5  6  7  8
```



This is a tabulation

# Directives

● `<%@ include file = "header.html" %>`

Tells the JSP engine to include the contents of another file (HTML, JSP,..) in the current page.

## example1.jsp

```
<html>
  <body>

    <%@ include file="header.html" %>

    ...

  </body>
</html>
```

## header.html

```
<html>
  <body>

    Welcome to the ADU

  </body>
</html>
```

## example2.jsp

```
<html>
  <body>

    <%@ include file="header.jsp" %>

    ...

  </body>
</html>
```

## header.jsp

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

    <%= (new Date()).toString() %>

  </body>
</html>
```

# Declarations

● Directives

● **Declarations**

● Scriptlets

● Expressions

● Comments

```
<html>
  <body>

    {
      <%@ page language      = "java"           %>
      <%@ page import        = "java.util.*"     %>
      <%@ page session       = "true"            %>
      <%@ page errorPage     = "error.html"       %>
      <%@ page contentType  = "text/html; charset=ISO-8859-1" %>

      <%@ include file      = "welcome.html"     %>

      <%! int count = 0 ; %>

      <% count++ ; %>

      <%= count %>

      <%-- JSP Comments --%>

    }

  </body>
</html>
```



# Declarations

➤ Declarations define **Variables** and **Methods** that can be used in the JSP page.

`<%! int count = 0 ; %>`

## example 1

```
<html>
  <body>
    <%! int count = 0 ; %>
    ...
  </body>
</html>
```

Don't forget it

## example 2

```
<html>
  <body>
    <%!
      String color [] = { "red" , "green" , "blue" };
      String getColor ( int i )
      {
        return color[ i % 3 ] ;
      }
    %>
  </body>
</html>
```



# Scriptlets

- Directives

```
<html>
  <body>

  {
    <%@   page language      = "java"                %>
    <%@   page import       = "java.util.*"          %>
    <%@   page session      = "true"                 %>
    <%@   page errorPage    = "error.html"            %>
    <%@   page contentType = "text/html; charset=ISO-8859-1" %>

    <%@   include file     = "welcome.html"          %>
  }
```

- Declarations

```
•---> <%!   int      count = 0 ;                %>
```

- **Scriptlets**

```
•---> <%      count++ ;                          %>
```

- Expressions

```
•---> <%=      count                               %>
```

- Comments

```
•---> <%--      JSP Comments                      --%>
```

```
    </body>
  </html>
```

# Scriptlets

➤ Scriptlets are **Java code** fragments that are embedded in the JSP page.

● `<% count++ ; %>`

## example

```
<html>
  <body>
    <%

      int sum = 0 ;

      for ( int i = 0 ; i <= 1000 ; i++ )
        sum += i ;

    %>
  </body>
</html>
```

# Expressions

- Directives

- Declarations

- Scriptlets

- **Expressions**

- Comments

```
<html>
  <body>
    {
      <%@ page language = "java" %>
      <%@ page import = "java.util.*" %>
      <%@ page session = "true" %>
      <%@ page errorPage = "error.html" %>
      <%@ page contentType="text/html; charset=ISO-8859-1" %>

      <%@ include file = "welcome.html" %>

      <%! int count = 0 ; %>

      •---> <% count++ ; %>

      •---> {
        <%= count %>
        <%= myBean.getSalary() %>
      }
      •--->

      <%-- JSP Comments --%>

      •---> </body>
    }
  </html>
```

# Expressions

- The expression is evaluated each time the JSP Page is accessed, and its value is then embedded in the output HTML.

`<%= count %>`

## example

```
<html>
  <body>

    <%
      int sum = 0 ;

      for ( int i = 0 ; i <= 1000 ; i++ )
        sum += i ;

    %>

    <%= sum %>

  </body>
</html>
```

# Comments



Directives

```
<html>
  <body>

  {
    <%@ page language      = "java"                %>
    <%@ page import        = "java.util.*"          %>
    <%@ page session       = "true"                 %>
    <%@ page errorPage     = "error.html"            %>
    <%@ page contentType="text/html; charset=ISO-8859-1" %>

    <%@ include file      = "welcome.html"          %>
  }
```



Declarations

```
    <%! int count = 0 ; %>
```



Scriptlets

```
    <% count++ ; %>
```



Expressions

```
    <%= count %>
```



Comments

```
    <!-- JSP Comments -->
```

```
  </body>
</html>
```

# Comments

- Comments do not affect the output of a JSP page but are useful for documentation purposes.

`<%-- JSP Comment --%>`

## example

```
<html>
  <body>

    <%--      JSP  Comment      --%>

    <%  //  Java Comment  %>

    <!--      HTML Comment      -->

  </body>
</html>
```

# Example

## MySum.jsp

```
<html>
<body>

    <%! int sum = 0 ; %>

    <%
for ( int i = 1 ; i <= 1000 ; i++ )
    sum += i ;
    %>

    The sum is <%= sum %>

</body>
</html>
```

## HTML Page

```
<html>
<body>

    The sum is 500500

</body>
</html>
```

# MySum\_jsp.java

```
...  
public final class MySum_jsp extends ...  
{  
    ...  
    int sum = 0 ;  
    ...  
    public void _jspService ( .. , .. )  
    {  
        ...  
        ("<html>\r\n    ");  
        out.write("<body>\r\n\r\n    ");  
        out.write("\r\n\r\n\t ");  
  
        for ( int i = 1 ; i <= 1000 ; i++ )  
            sum += i ;  
  
        out.write("\r\n\r\n\t The sum is ");  
        out.write(String.valueOf( sum ));  
        out.write("\r\n\r\n    ");  
        out.write("</body>\r\n");  
        out.write("</html>\r\n\r\n");  
        ...  
    }  
    ...  
} out.write
```



# Implicit Variables

- During the translation phase, the JSP engine declares and initializes a set of variables in the `_jspService()` method. They are called **implicit variables**.

## ● `application ( javax.servlet.ServletContext )`

Refers to the web application's environment.

### example

```
<html>
  <body>
    <%@ page import="java.util.*" %>
    ...
    <%
    ...
      Date currentDateAndTime = new Date();

      application.log ("Invocation Date and Time : " +
        currentDateAndTime );
    ...
    %>
    ...
  </body>
</html>
```

# Implicit Variables

- **request** ( `javax.servlet.http.HttpServletRequest` )

Refers to the current request the page.

## example 1

```
<html>
  <body>

    The Remote Host is      : <%= request .getRemoteHost() %>

  <hr>

    The Remote Address is   : <%= request .getRemoteAddr() %>

  <hr>

    The Remote Port is      : <%= request .getRemotePort() %>

  </body>
</html>
```

## Login.jsp

```
<html>
  <body>
    <%
      String login      = request.getParameter ( "login" );
      String password    = request.getParameter ( "password" );
    %>
```

The values entered by the user are :

```
    <br>
    login   : <%= login %>
    <br>
    password : <%= password %>
```

```
  </body>
</html>
```

## The FORM Page

```
<html>
  <body>
    Please enter your user name and password :

    <form action="Login.jsp " method="POST" >
      Login      : <input type="text"          name="login"          >
        <br>
      Password   : <input type="password"      name="password"      >
        <br>
        <input type="submit" name="Send" value="Send" >
    </form>

  </body>
</html>
```

# Implicit Variables



**session** ( `javax.servlet.http.HttpSession` )



A **session** is an **uninterrupted series** of **request-response** interactions between a client and a server.



For each request that is a part of this session, the server is able to identify the request as coming from the **same client** .



A session **starts** when an unknown client sends the **first request** to the web application server.



It **ends** when either the client explicitly **ends** the session **or** the server does not receive any requests from the client within a predefined **time limit** .



When the session **ends** , the server conveniently **forgets** the client as well as all the requests that the client may have made.

# Example

## Page1.jsp

```
<html>
  <body>
    <%@ page session = "true" %>

    <H1> Page 1 </H1>
    <%= session.getId () %>

    <HR>

    <a href="Page1.jsp"> Page1.jsp </a>
    <a href="Page2.jsp"> Page2.jsp </a>
    <a href="Page3.jsp"> Page3.jsp </a>
  </body>
</html>
```

## Page2.jsp

```
<html>
  <body>
    <%@ page session = "true" %>

    <H1> Page 2 </H1>
    <%= session.getId () %>

    <HR>

    <a href="Page1.jsp"> Page1.jsp </a>
    <a href="Page2.jsp"> Page2.jsp </a>
    <a href="Page3.jsp"> Page3.jsp </a>
  </body>
</html>
```

## Page3.jsp

```
<html>
  <body>
    <%@ page session = "true" %>

    <H1> Page 3 </H1>
    <%= session.getId () %>

    <HR>

    <a href="Page1.jsp"> Page1.jsp </a>
    <a href="Page2.jsp"> Page2.jsp </a>
    <a href="Page3.jsp"> Page3.jsp </a>
  </body>
</html>
```

# Passing Parameters between JSP Pages

## Login.jsp

```
<html>
  <body>
    <%@ page session="true" %>

    <%
      String login    = request.getParameter ( "login" );
      String password  = request.getParameter ( "password" );

      session.setAttribute ( "user" , login  );
      session.setAttribute ( "pass" , password );
    %>

    <H3> Stored Values : </H3>

    <%= login %> , <%= password %>

    <HR>

    <a href="NextPage.jsp">NextPage.jsp </a>

  </body>
</html>
```



# Passing Parameters between JSP Pages ( cont. )

## NextPage.jsp

```
<html>
  <body>
    <%@ page session="true" %>

    <%
      String login    = ( String ) session.getAttribute ( "user" );
      String password = ( String ) session.getAttribute ( "pass" );
    %>

    <H3> Extracted Values : </H3>

    <%= login %> , <%= password %>

  </body>
</html>
```

# Passing Parameters between JSP Pages ( cont. )

## User.java

```
package is300;

public class User
{
    private String userName ;
    private String password ;

    public User ( String _userName , String _password )
    {
        userName = _userName;
        password = _password;
    }

    public String getUsername ()
    {
        return userName ;
    }

    public String getPassword ()
    {
        return password ;
    }
}
```



# Passing Parameters between JSP Pages ( cont. )

## Login2.jsp

```
<html>
  <body>
    <%@ page session = "true" %>

    <%@ page import = "is300.*" %>

    <%
      String login    = request.getParameter ( "login" );
      String password = request.getParameter ( "password" );

      User userObject = new User ( login , password );

      session.setAttribute ( "user" , userObject );
    %>

    <H3> Stored Values : </H3>

    <%= login %> , <%= password %>

    <HR>

    <a href="NextPage2.jsp">NextPage2.jsp </a>

  </body>
</html>
```

# Passing Parameters between JSP Pages ( cont. )

## NextPage2.jsp

```
<html>
  <body>
    <%@ page session="true" %>

    <%@ page import="is300.*" %>

    <%
      User user = ( User ) session.getAttribute ( "user" );
    %>

    <H3> Extracted Values : </H3>

    <%= user.getUserName() %> , <%= user.getPassword() %>

  </body>
</html>
```

# JSP STANDARD ACTIONS



# JSP forward Action

- JSP standard action elements simplify the access actions to other Web components such as **Servlet**, **JSP**, or **JavaBean** components.

● `<jsp:forward page="another page" />`

- This action forwards the request to another page, *i.e.* an internal redirect.
- For example, it forwards the control from current page to *second.jsp*.

● `<jsp:forward page='second.jsp' />`

# JSP include Action

- In addition to the JSP page include directive that includes a page at the JSP translation time, JSP also supports an action that can dynamically include a page at run-time.

● `<jsp:include page = "<JSP or html page>" />`

- The jsp:include action element is like a function call. At runtime, the included file will be 'executed' and the result content will be included with the source JSP page.
- When the included JSP page is called, both the request and response objects are passed as parameters.
- *Remember that the include directive only copies the contents once and never changes afterwards since it is static.*

# Example 1: <jsp:include> without parameters

In this example we will use <jsp:include> action tag without parameters. As a result the page will be included in the current JSP page as it is:

index.jsp

```
<html>

<head>

<title>JSP Include example</title>

</head>

<body>

<b>index.jsp Page</b><br>

<jsp:include page="Page2.jsp" />

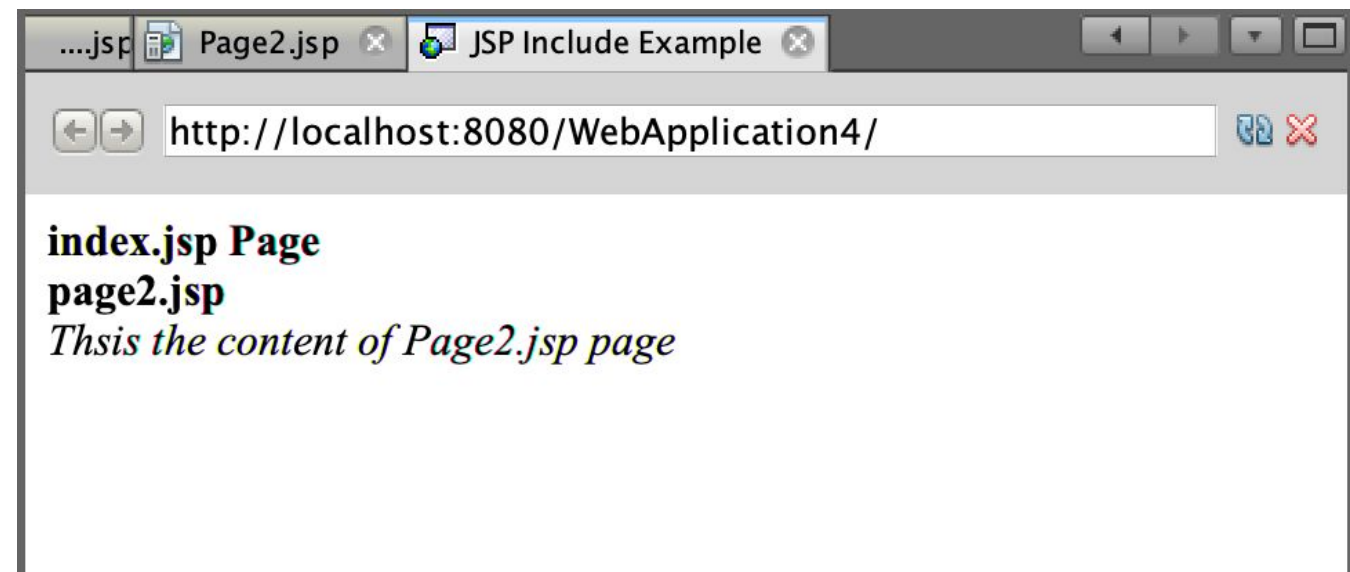
</body>

</html>
```

Page2.jsp

```
<b>Page2.jsp</b><br>

<i> This is the content of Page2.jsp page</i>
```



# Example 2: Use of <jsp:include> along with <jsp:param>

index.jsp

I'm using <jsp:include> action here along with <jsp:param> for passing parameters to the page which we are going to include.

```
<html>

<head>

<title>JSP Include example with parameters</title>

</head>

<body>

<h2>This is index.jsp Page</h2>

<jsp:include page="display.jsp">

<jsp:param name="userid" value="ali" />

<jsp:param name="password" value="12345" />

<jsp:param name="name" value="Ali Mohammed" />

<jsp:param name="age" value="22" />

</jsp:include>

</body>
</html>
```

display.jsp

```
<html>

<head>

<title>Display Page</title>

</head>

<body>

<h2>Hello this is a display.jsp Page</h2>

UserID: <%=request.getParameter("userid") %><br>

Password is: <%=request.getParameter("password") %><br>

User Name: <%=request.getParameter("name") %><br>

Age: <%=request.getParameter("age") %>

</body>
</html>
```





# JSP JavaBean Actions

## JavaBean Components

- JSP provides a number of **JSP javaBean action** elements to free JSP page designers from complex Java Servlet programming or JSP scripting constructs.
- A JavaBean is a wrapper for business logic and data processing. Any Java class can be converted to a JavaBean class as long as it satisfies the JavaBean convention (design pattern).
- You can easily create and initialize beans and set and get the values of their **properties**. A simple bean named Book is given below.
- The JSP actions can only call **set** and **get methods**. Other JSP scriptlets can call other methods. The bean class file is placed in Tomcat *WEB-INF/classes* along with the Servlet class.

# JavaBean Components (cont.)

```
package shopping;
```

```
public class Book implements  
java.io.Serializable{
```

```
    String title;  
    String isbn;  
    float price;  
    int quantity;
```

```
public Book() {  
    title="";  
    isbn="";  
    price=0;  
    quantity=0;  
}
```

```
public void setTitle (String name) {  
    title=name;  
}
```

```
public String getTitle () {  
    return title;  
}
```

```
public void setIsbn (String id) {  
    isbn=id;
```

```
    }  
    public String getIsbn () {  
        return isbn;  
    }
```

```
    public void setPrice (float p) {  
        price=p;  
    }
```

```
    public float getPrice () {  
        return price;  
    }
```

```
    public void setQuantity (int q) {  
        quantity=q;  
    }
```

```
    public int getQuantity () {  
        return quantity;  
    }
```

```
}
```

# jsp:useBean Action Element

- The **jsp:useBean** action instantiates an instance of the bean class if there is not an existing one or creates a reference to an existing one. It also specifies the visibility and accessibility scope of this bean. The other Web components in JSP can reference this bean object by its *id*. Its syntax is given as:

● **<jsp:useBean id="name" class="<package>.<Java bean class>" scope="...">**

- The *id* is the access name of this bean. The scope attribute defines the visibility of this bean.

- |                      |   |
|----------------------|---|
| ▪ <b>page</b>        | Only active in the page , default scope |
| ▪ <b>request</b>     | Active for the current request          |
| ▪ <b>Session</b>     | Active for the current session          |
| ▪ <b>application</b> | Active for the current application      |

# jsp:useBean Action Element (cont.)

- After you get a reference to a bean you can use **jsp:setProperty** action and **jsp:getProperty** action to modify and retrieve the bean properties.
- Once you have a bean reference, you can also modify its properties by calling the bean's **getXXX()** or **setXXX()** methods explicitly in a scriptlets element.
- Note that the class specified for the bean must be in the server's regular class path. For example, in Tomcat, this class and all the classes it uses should go in the *classes* directory or be in a **jar** file in the **lib** directory.
- You can also incorporate Java code into the body of a `jsp:useBean` tag so that the incorporated code will be executed when the bean is instantiated for the first time. If the instance exists already the code will be ignored.

```
<jsp:useBean ...>
```

```
    Java code
```

```
</jsp:useBean>
```

# jsp:setProperty Action

- You use **jsp:setProperty** to assign values to properties of a bean. You can either use **jsp:setProperty** after, but outside of, a **jsp:useBean** element or within the body of **jsp:useBean** action element.

- Option 1:

- **<jsp:useBean id="myBook" ... />**

- **<jsp:setProperty name="myBook" property="price" ... />**

- In this case, the **jsp:setProperty** is executed regardless of whether a new bean was instantiated or an existing bean was found.

# jsp:setProperty Action (cont.)

- Option 2:

```
<jsp:useBean id="myBook "...>
.....

<jsp:setProperty name="myBook" property="price" ...
/>

</jsp:useBean>
```

In this case, the **jsp:setProperty** is executed only if a new object is instantiated, *i.e.*, no existing bean object exists yet.

The following action sets *myBook*'s *price* property to 31.99.

- **<jsp:setProperty name="myBook" property="price" value = "31.99" />**

# jsp:setProperty Action (cont.)

- The required “**name**” attribute is the *id* of the bean specified in **<jsp:useBean>** action tag.
- The required “**property**” attribute specifies the property you want to set. You can also assign a “**\***” to the property **if all request parameter names match bean property names**. It will set all corresponding properties of the bean with the values of request parameters either from request form or request string.
- The “**value**” attribute specifies the value for the property. The String type value will be converted to the correct type for the property.



# jsp:setProperty Action (cont.)

- You can also associate the bean properties with the input parameters. The next example shows that the *bookPrice* input parameter in the request form or query string is associated with the *price* property of *myBook* bean.

**<jsp:setProperty name="myBook" property="price" param="bookPrice" />**

- Or you can associate all properties with input request parameters with the "\*" wildcard character. Assume that there are four input request parameters whose names match four property names: *title*, *isbn*, *price*, and *quantity*. Whatever data you input to the parameters will be assigned to the properties with same names. This is one of the most useful javaBean features.

**<jsp:setProperty name="myBook" property="\*" />**

# jsp:setProperty Action (cont.)

- You can't use both value and param at same time, but it is allowed to use neither. If you use neither param nor value, that indicates that a parameter name matches the property name. JSP will take the input from the parameter which has the same name as the property name.
- **jsp:setProperty** action is much more powerful than the **setXXX()** methods in that it can automatically convert String type data to the required data type of the bean property and set multiple properties using only one single command.

# jsp:getProperty Action Element

- This element retrieves the value of a bean property, converts it to a string, and inserts it into the JSP outputs. The two required attributes are *name* – the *id* of a bean defined via jsp:useBean, and *property* – the property whose value should be inserted.

```
<jsp:useBean id='myBook'  
            class='shopping.Book'  
            scope='session' >
```

...

- The next action can return the number of copies of myBook, which may reference a specific book at the time.

```
<jsp:getProperty name="myBook" property="quantity" />
```

- To return the price of the same book:

```
<jsp:getProperty name="myBook" property="price" />
```

```
</jsp:useBean>
```

# jsp:getProperty Action Element(cont.)

- The *form.jsp* takes user name input on the form request, and calls ***response.jsp***. The *response.jsp* JSP page saves the input in a `JavaBean`, gets the username from the bean, and finally it responds to the user with a message “Hello, <user name>”.

This is the **form.jsp**.

```
<html>
<body>
<form method="get" action="response.jsp">
<input type="text" name="userName" size="20">
<br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

# jsp:getProperty Action Element (cont.)

- Here is the *NameBean* javaBean file.

```
package user;

public class NameBean {
    String userName;

    public NameBean() { userName=null; }

    Public void setUserName (String name) { userName=name; }

    public String getUserName() { return userName; }
}
```

- Here is the *response.jsp* file

```
<%@ page import=user.NameBean" %>
<jsp:useBean id="myBean" scope="request" class="user.NameBean" />
<jsp:setProperty name="myBean" property="*" />
<html><body>
<h1>Hello, <jsp:getProperty name="myBean" property="userName" /><h1>
</body></html>
```

# jsp:plugin Action Element

- The **jsp:plugin** action can insert an Java Applet client-side component into a server-side JSP page component. It downloads Java plug-in software (if necessary) and client-side component such as Applet and executes the client-side component.
- The syntax is as follows.

```
<jsp:plugin type="applet" code="MyApplet.class" width="400"
height="200">
...
<!-- Parameter lists passed on the current JSP -->
  <jsp:param name="username" value="Smith" />
...
</jsp:plugin>
```

# jsp:plugin Action Element (cont.)

- The `<jsp:plugin>` element can display an Applet object or a bean object in the client Web browser, using a Java plug-in which is part of the browser or downloaded from a specified URL.
- When the JSP file sends an HTML response to the client, the `<jsp:plugin>` element is replaced by an `<object>` element in HTML specification. In general, the attributes to the `<jsp:plugin>` element specify whether the object is a bean or an applet, locate the code that will be run, position the object in the browser window, specify an URL from which to download the plug-in software, and pass parameter names and values to the object.



# JSP implicit variables (cont.)

- **application:** represents the `ServletContext` obtained from servlet configuration object. You can set a attribute by `application.setAttribute(key,value)` and get the attribute value by `application.getAttribute (key)`. The application object has Application scope.
- **config:** represents the `ServletConfig` for the JSP. The JSP initialization parameters can be read via it. It has a page scope.
- **page:** synonym for the "this" operator. Not used often by page authors.
- **session:** an `HttpSession` object with session scope. `session.setAttribute(key,data)`; `session.getAttribute(key)`, and `session.getId()` are examples.
- These implicit objects are only visible within the system generated `_jspService()` method.

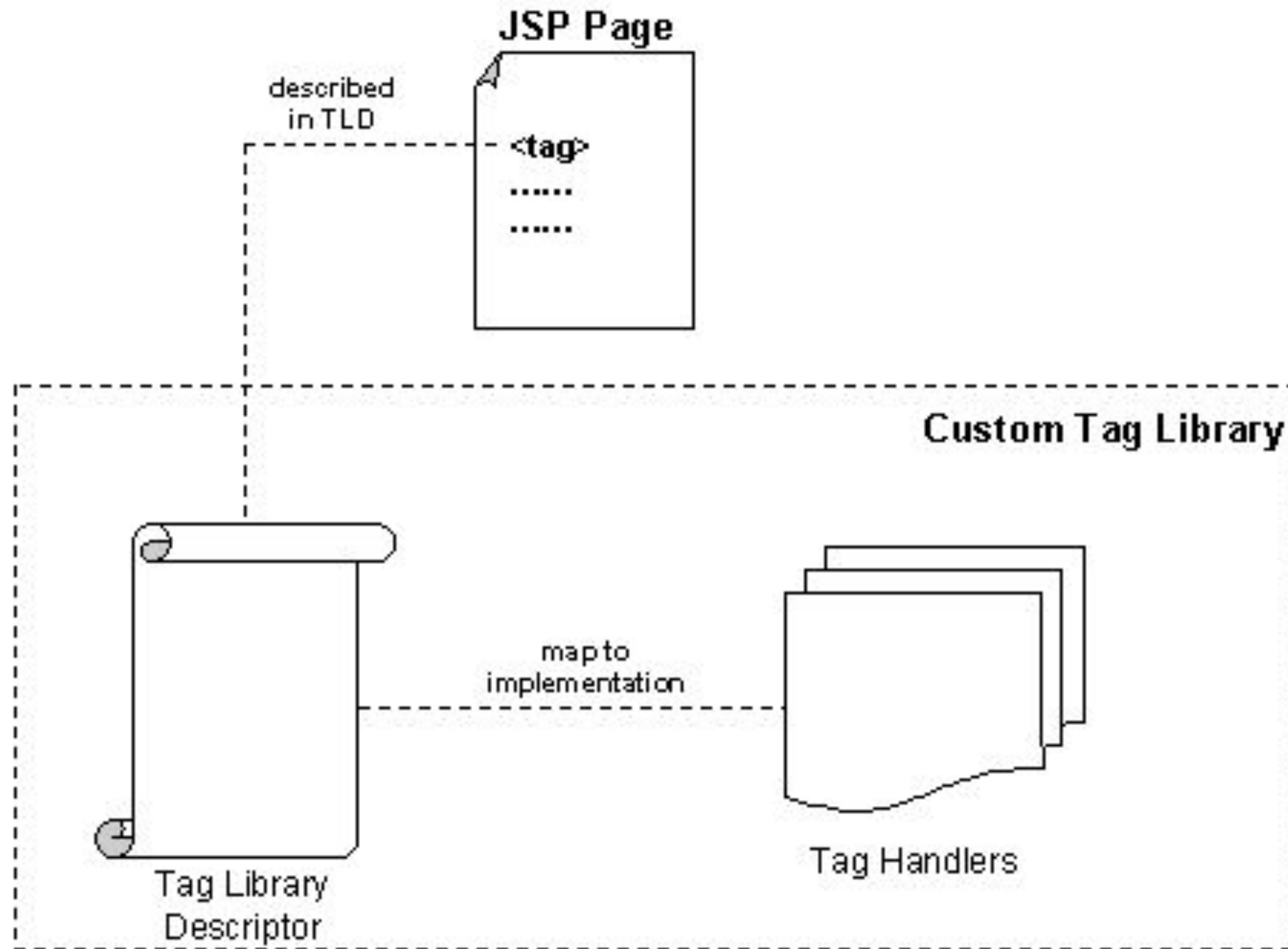
# JSP CUSTOMIZED TAG

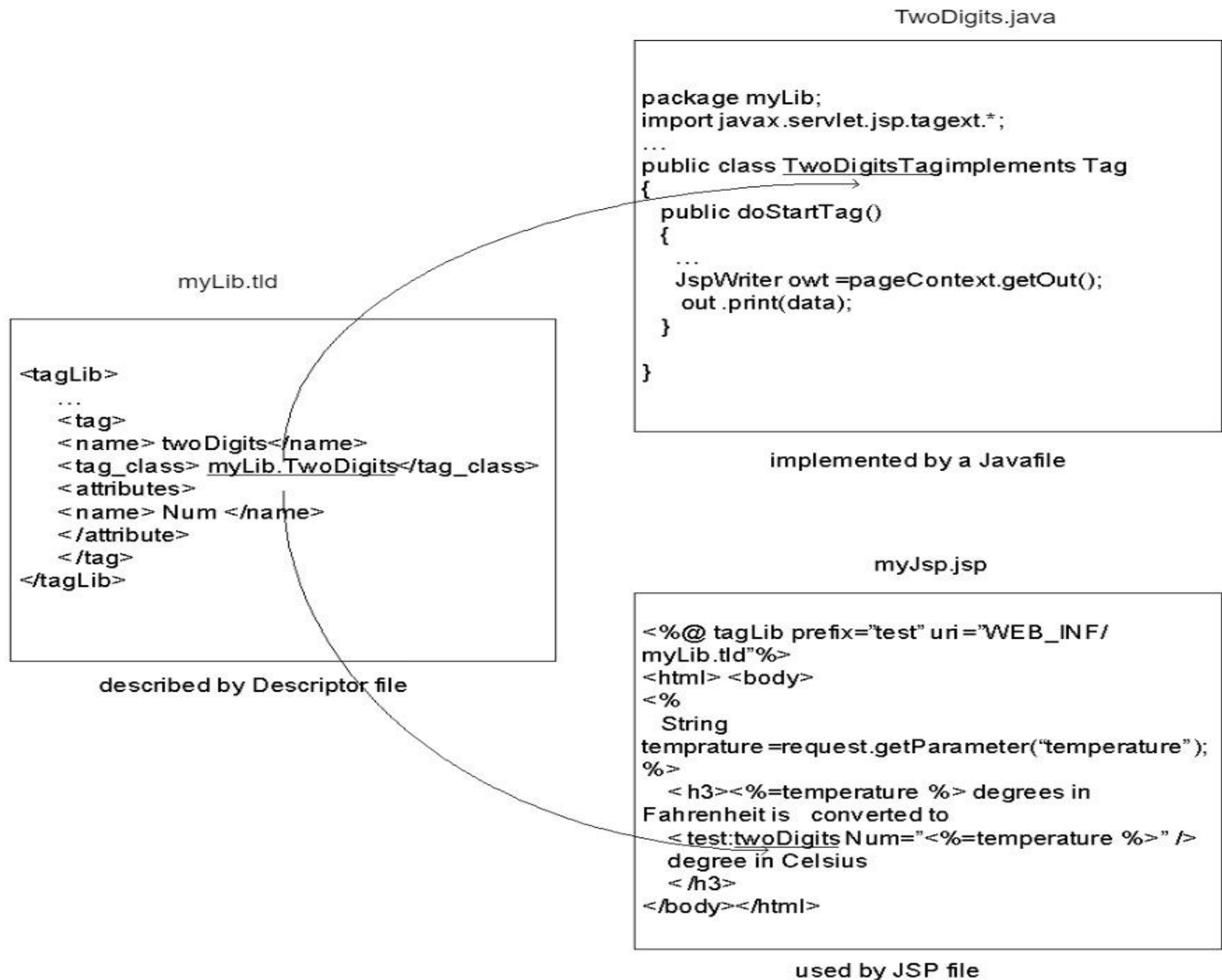


# Custom Tag Library

- A custom tag library feature allows JSP developers to define their own reusable custom tags. Customized tags are stored in tag libraries called *taglibs*. The advantages of custom tags are as follows:
  - Separates HTML page authoring from Java programming.
  - Encapsulates complex Java code behind the tag interface that makes it easy to reuse.
  - Easy to maintain because any change of tag implementation will not result in any change of the tag application.
- In order to make custom tags work, there must be a XML descriptor file to map the tag to its implementation class file, called the **tag handler class**; there must also be a **Java tag handler class**. Of course, the custom defined tag is used in a JSP file. Figure 4.4 shows the relations among these three files.

# Custom Tag Library





## Custom Tag Library

# Tag implementation class

- First, let's discuss the *WelcomeTag* tag handler class which just displays "Welcome!" The file is saved in *webapps\tagLib\myLib\WelcomeTag.java* in Tomcat where *taglib* is this Web application root directory.

```
package myLib;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;

public class WelcomeTag implements TagSupport {
{
```

# Tag implementation class (cont.)

```
public int doStartTag() throws JspException
{
    try
    {
        JspWriter out = pageContext.getOut();
        out.print(" Welcome! ");
    }
    catch(Exception e)
    {
        throw new JspException("error");
    }
    return SKIP_BODY;
}
```



# Tag implementation class (cont.)

- The container invokes the **doStartTag()** when the start-tag is encountered. It is processed in the same way as the Java Simple API for XML parser (SAX) handles XML tag elements.
- Next, let's look at the application of this **welcome** tag used in the *webapps>tagLib\myFirst.jsp*
- This JSP specifies the location of the **myLib.tld** file which maps the definition of *Welcome* custom tag to its
- WelcomeTag tag handler class file. The prefix attribute in the **taglib** directive plays a role of “namespace”. When this **myFirst.jsp** file is browsed the word “Welcome!” is displayed on the browser screen.

```
<%@ taglib prefix="test" uri="/WEB-INF/myLib.tld" %>
<html><body>
<test:welcome/>
</body></html>
```

# Tag implementation class (cont.)

- The *webapps\tagLib\WEB-INF\myLib.tld* defines the *welcome* tag and maps the *welcome* tag name to its tag handler class *myLib.Welcome.class*.

```
<!DOCTYPE taglib
  PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
  "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
  <!-- INFO ABOUT THIS LIBRARY -->
    <tlib-version>1.0</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>myLib</short-name>
  <!-- GREET TAG -->
    <tag>
      <name>welcome</name>
      <tag-class>myLib.WelcomeTag</tag-class>
      <body-content>empty</body-content>
    </tag>
</taglib>
```

# JSP 2.0 EXPRESSION LANGUAGE

# Expression Language (EL)

- A new feature of JSP technology version 2.0 is the Expression Language (EL). The EL is a simple and very powerful notation which makes it easier for the JSP page author to write code for accessing application data such as data stored in JavaBean components and attribute data of pageContext, request, session, and application. EL uses a shorthand notation of `${expr}` instead of `<%= expr %>`.

- For example, you have a *Book* JavaBean class in *shopping* package

```
package shopping;
public class Book {
    ...
    float price;
    public Book() {
        ...
        price= 0;
    }
    ...
    public void setPrice(float p) {
        price=p;
    }
    public float getPrice() {
        return price;
    }
    ...
}
```

# Expression Language (EL) (cont.)

## After you specify

```
<jsp:useBean id="myBook" class="shopping.Book" scope="session" />
```

you can use **`${myBook.price}`** to get the value of the *price* property of the bean instead of using the following scripting elements:

```
<%@ page import="shopping.Book" %>
```

```
<% Book myBook = new Book(); %>
```

```
...
```

```
<%= myBook.getPrice() %>
```

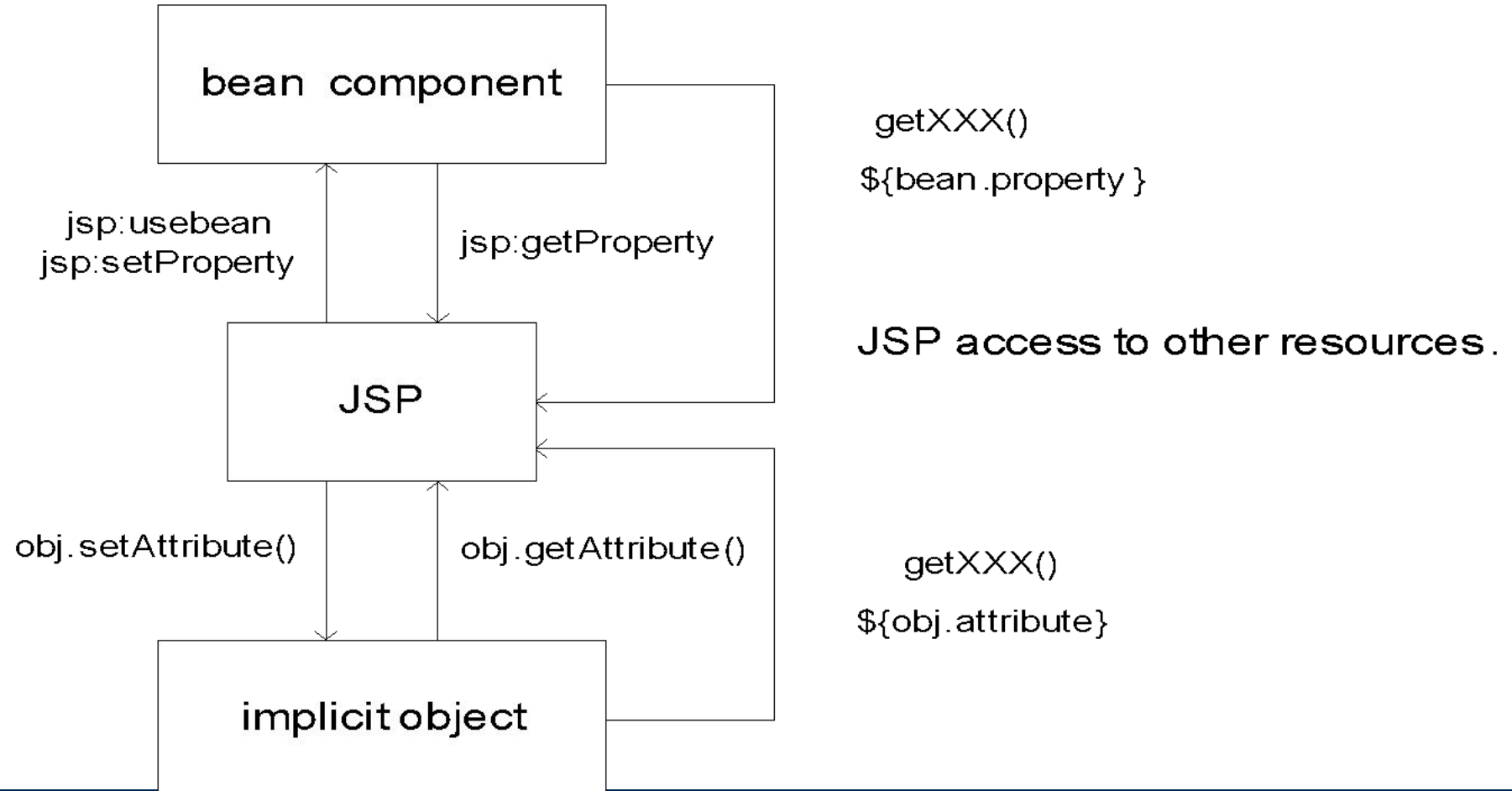
You can also use **`${myBook.price}`** to replace the following JSP action elements:

```
<jsp:getProperty name="myBook" property="price" />
```

- If the *myBook.price* does not exist the expression just returns Null value rather than throwing an exception.

# Expression Language (EL) (cont.)

- The Expression Language also simplifies access to collection type data: request parameters, cookies, etc. It supports the conditional expression **`${test? Opt1, Opt2}`** which is an alternative to a scriptlet element. It also supports automatic type conversion.
- Addition to above, another useful feature of EL is the nested notation, such as **`${student.address.state}`** where state is a property of address class and the address is a property of the student class. EL can retrieve data but can never set any data because EL is an expression language only.



# General syntax for bean access

- $\{\text{expr1}.\text{expr2}\}$  or  $\{\text{expr1}["\text{expr2}"]\}$  where *expr1* is the bean reference and *expr2* is the property of the bean.



# Syntax for collection data access

- **`${expr1[expr2]}`** where **`expr1`** is a reference to an array or a list and **`expr2`** is an integer index.
- For example, **`${myList[2]}`** returns a value at 3rd position of this list.
- **`${expr1[expr2]}`** or **`${expr1.expr2}`** where **`expr1`** is a map and **`expr2`** is the key of the map that the value of this key is returned. For example, **`${myMap["Smith"]}`** returns the phone number of Smith. There is an entry in the MyMap for Smith.

## Examples

**`${param['userName']}`** is equivalent EL notation to **`<%=request.getParameter("userName") %>`**

**`${pageContext.session.id}`** returns an *id* number of the session the page belongs to.