Java Server Pages

Shristi Technology Labs

Contents

Introduction to JSP

- JSP is web tier spec that supplements servlet spec and is useful in development of web applications

- Combines HTML/ XML markup languages and elements of Java language to return dynamic content to a web client

- Handles presentation logic, acts as a view

- Goal is to support separation of presentation and business logic
  - web designers can design and update pages without learning Java language
  - programmers for Java can write code without dealing with web page design

Developing JSP pages

Creation

- write JSP page containing HTML and jsp code

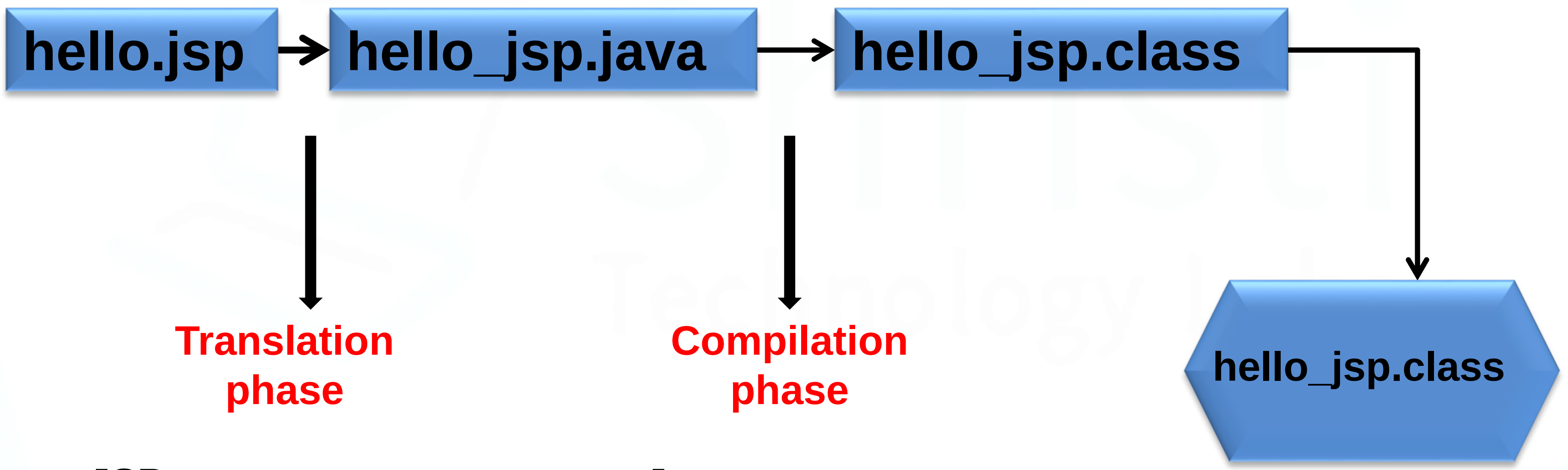
Deployment

- JSP is installed into server which is a full application server, or a standalone server

Translation and compilation

- Page executes inside a container
- Page is translated and compiled into Java class by the container
- This happens on the first clients request
- Container translates jsp file into java source file
- Source file is compiled into page implementation class and run on server

JSP Lifecycle phases



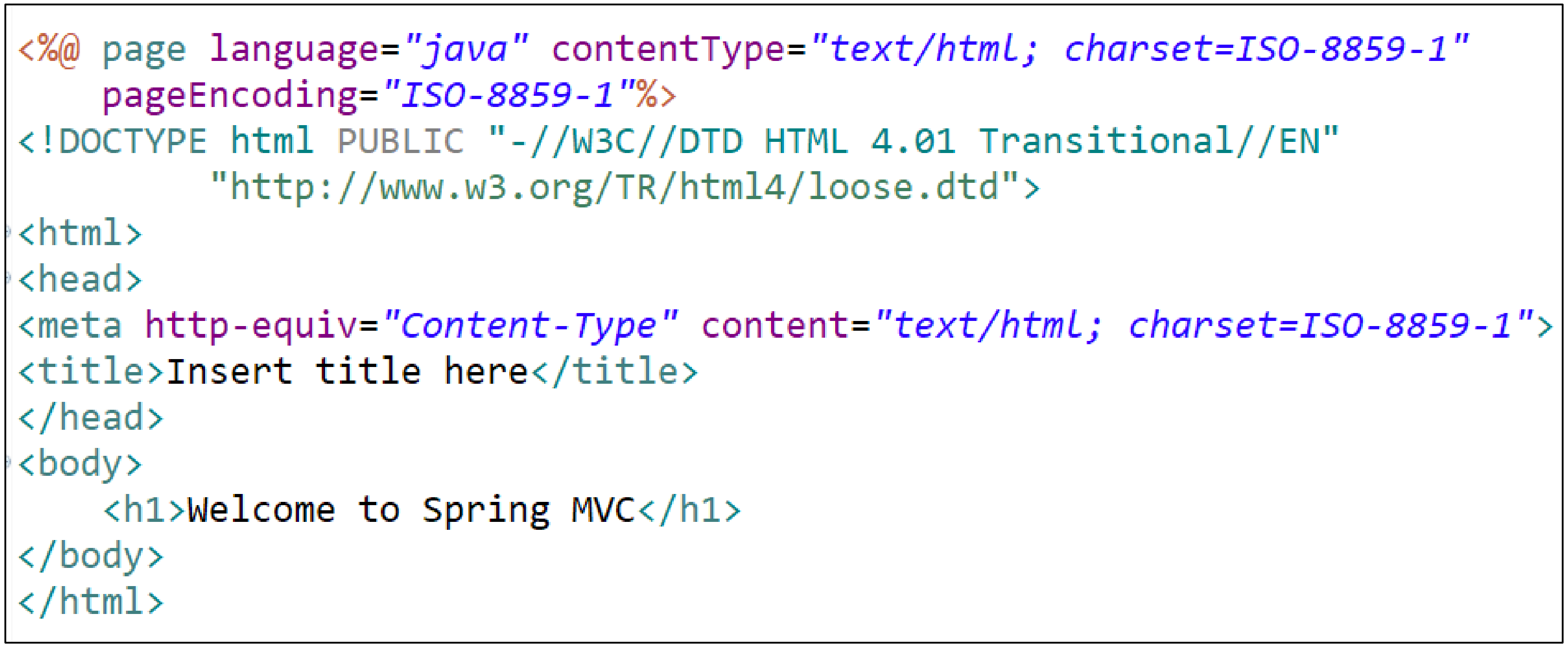
Translation phase

JSP syntax errors captured here

Compilation phase

Java syntax errors captured here

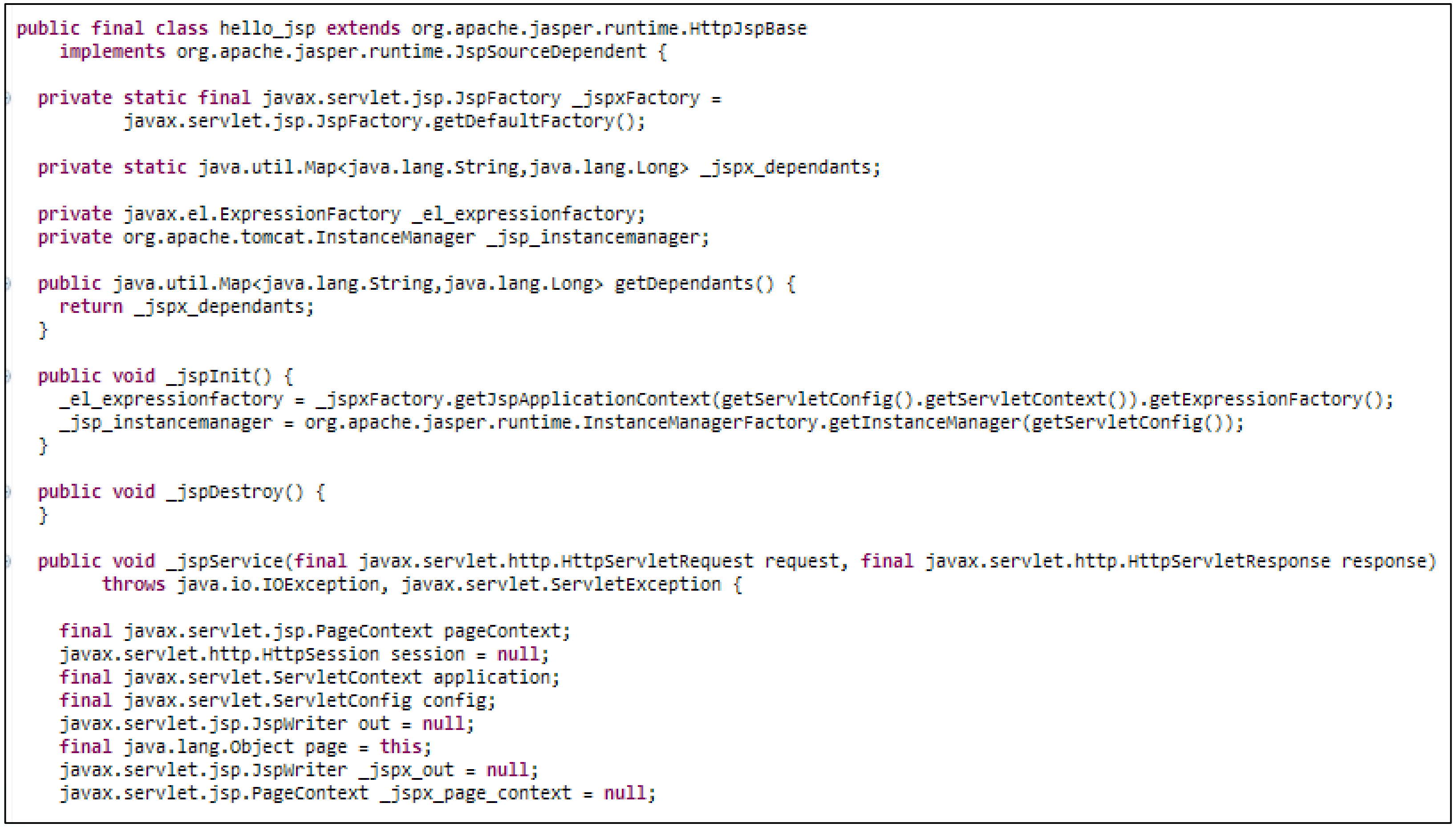
hello.jsp



```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
         "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <h1>Welcome to Spring MVC</h1>
</body>
</html>
```

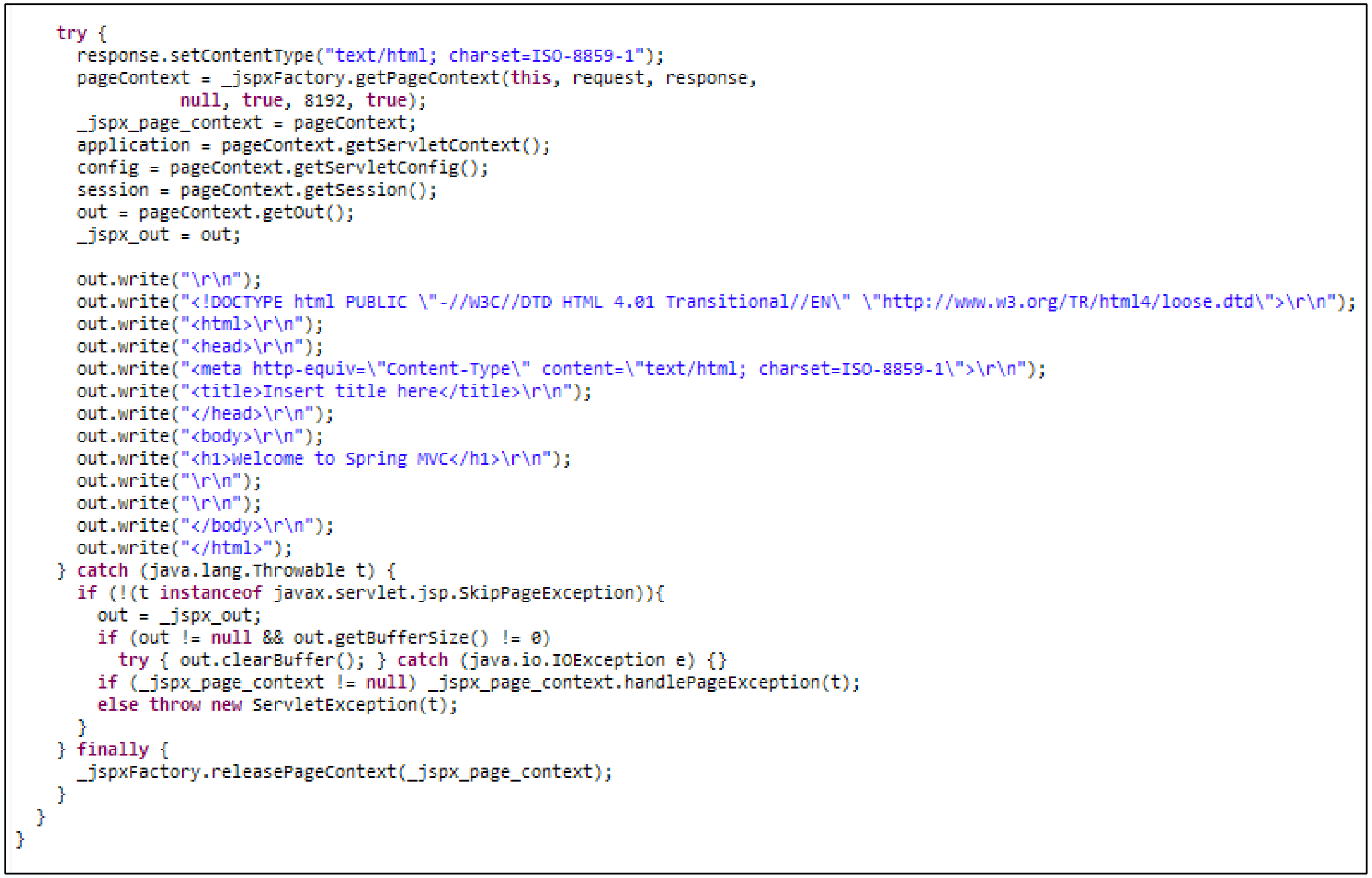
print name

hello_jsp.java



```java
public final class hello_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

  private static final javax.servlet.jsp.JspFactory _jspxFactory =
          javax.servlet.jsp.JspFactory.getDefaultFactory();

  private static java.util.Map<java.lang.String,java.lang.Long> _jspx_dependants;

  private javax.el.ExpressionFactory _el_expressionfactory;
  private org.apache.tomcat.InstanceManager _jsp_instancemanager;

  public java.util.Map<java.lang.String,java.lang.Long> getDependants() {
    return _jspx_dependants;
  }

  public void _jspInit() {
    _el_expressionfactory = _jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressionFactory();
    _jsp_instancemanager = org.apache.jasper.runtime.InstanceManagerFactory.getInstanceManager(getServletConfig());
  }

  public void _jspDestroy() {
  }

  public void _jspService(final javax.servlet.http.HttpServletRequest request, final javax.servlet.http.HttpServletResponse response)
        throws java.io.IOException, javax.servlet.ServletException {

    final javax.servlet.jsp.PageContext pageContext;
    javax.servlet.http.HttpSession session = null;
    final javax.servlet.ServletContext application;
    final javax.servlet.ServletConfig config;
    javax.servlet.jsp.JspWriter out = null;
    final java.lang.Object page = this;
    javax.servlet.jsp.JspWriter _jspx_out = null;
    javax.servlet.jsp.PageContext _jspx_page_context = null;
```

hello_jsp.java



```
    try {
      response.setContentType("text/html; charset=ISO-8859-1");
      pageContext = _jspxFactory.getPageContext(this, request, response,
                null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;

      out.write("\r\n");
      out.write("<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\" \"http://www.w3.org/TR/html4/loose.dtd\">\r\n");
      out.write("<html>\r\n");
      out.write("<head>\r\n");
      out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=ISO-8859-1\">\r\n");
      out.write("<title>Insert title here</title>\r\n");
      out.write("</head>\r\n");
      out.write("<body>\r\n");
      out.write("<h1>Welcome to Spring MVC</h1>\r\n");
      out.write("\r\n");
      out.write("\r\n");
      out.write("</body>\r\n");
      out.write("</html>");
    } catch (java.lang.Throwable t) {
      if (!(t instanceof javax.servlet.jsp.SkipPageException)){
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
          try { out.clearBuffer(); } catch (java.io.IOException e) {}
        if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
        else throw new ServletException(t);
      }
    } finally {
      _jspxFactory.releasePageContext(_jspx_page_context);
    }
  }
}
```

Lifecycle of JSP

jspInit( )

- container calls this method once for each instance
- do one time set up such init variables and acquire resources with declarations (<%! … %>)

_ jspService( )

- container calls it for each request
- passes the request and response
- HTML elements, scriptlets and expressions are part of it

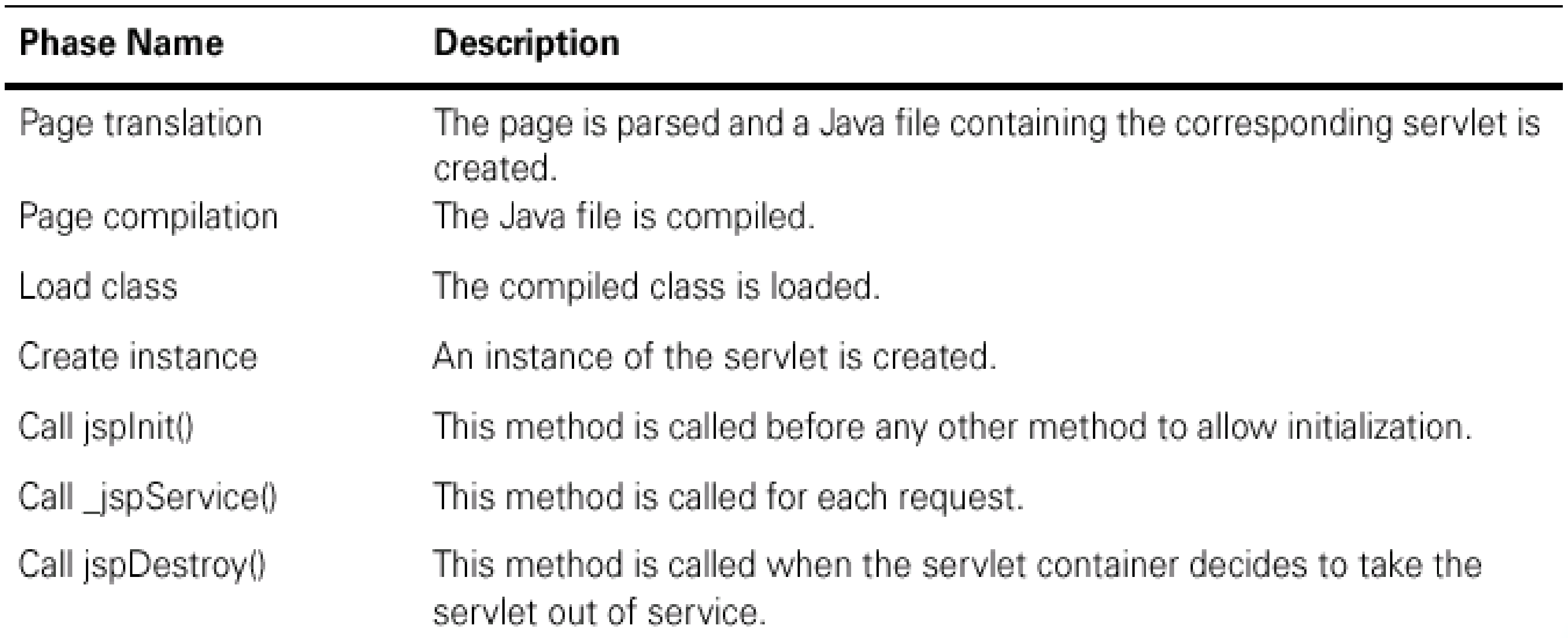
jspDestroy( )

- to clean up acquired resources

- **May override init and destroy, if needed**
- **Do not override service;**

Lifecycle Phases of JSP

- Seven phases
- Slow in loading first time since it has to be translated into



Phase Name	Description
Page translation	The page is parsed and a Java file containing the corresponding servlet is created.
Page compilation	The Java file is compiled.
Load class	The compiled class is loaded.
Create instance	An instance of the servlet is created.
Call jspInit()	This method is called before any other method to allow initialization.
Call _jspService()	This method is called for each request.
Call jspDestroy()	This method is called when the servlet container decides to take the servlet out of service.

SCRIPTING ELEMENTS

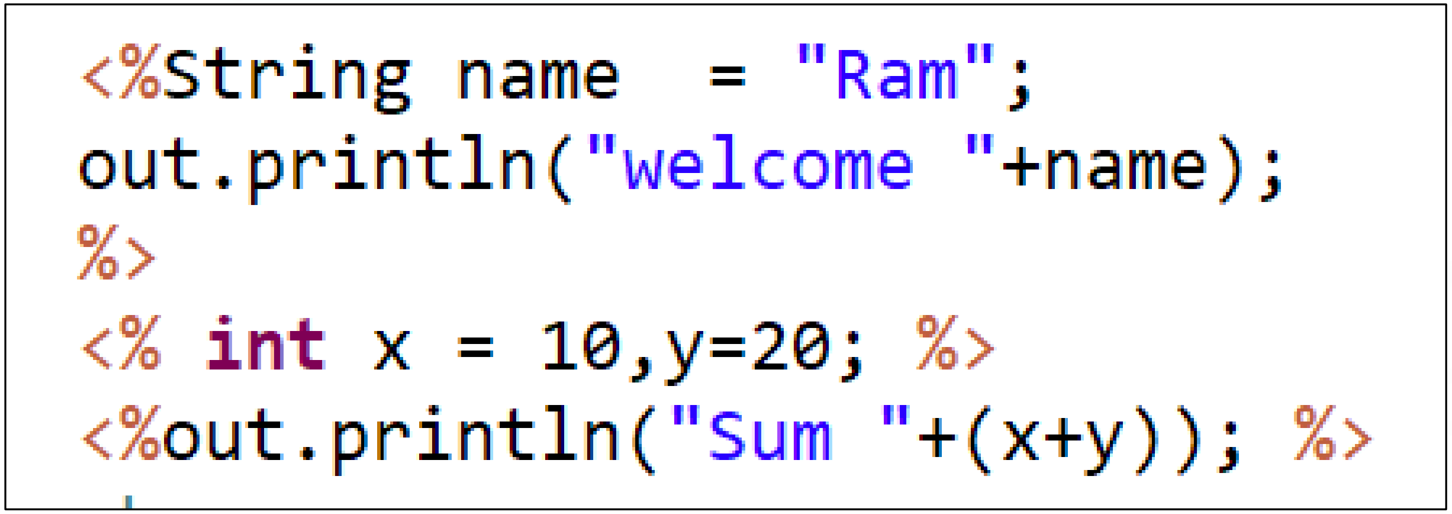
Scripting Elements

- Scriptlets
- Expression
- Declaration
- Directives

Scriptlets

- Java code fragments embedded in HTML page
- Gets executed each time page is accessed
- The java code is embedded within **<%--------- %>** tag
- Is used to embed computing logic in page
- The code within this tag goes inside _***jspService***
- Instead of writing HTML, we can use scriptlets
- ***out*** refers to object of type ***JspWriter***
- Code must be valid Java statement and end with **';'**

Example

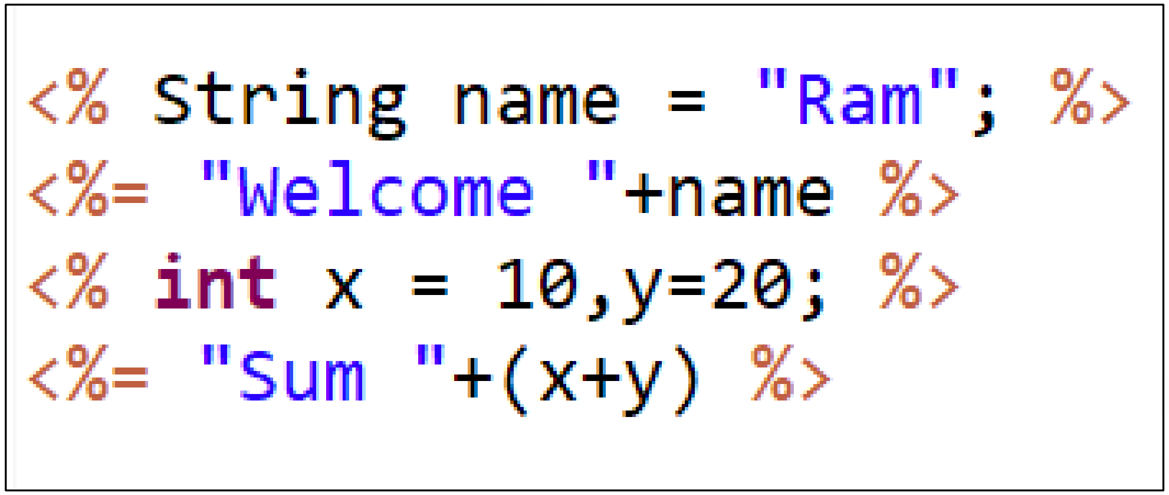


```
<%String name  = "Ram";
out.println("welcome "+name);
%>
<% int x = 10,y=20; %>
<%out.println("Sum "+(x+y)); %>
```

Expressions

- Replaces ***out.println()*** method
- Placeholders for Java expressions
- Evaluated each time page accessed, and value is embedded in HTML page
- Code is embedded within **<%=   -----------   %>**
- **NOT** terminated by ***';'***
- Can print value of any object of primitive, or arithmetic/ boolean expression or value returned by method call

Example

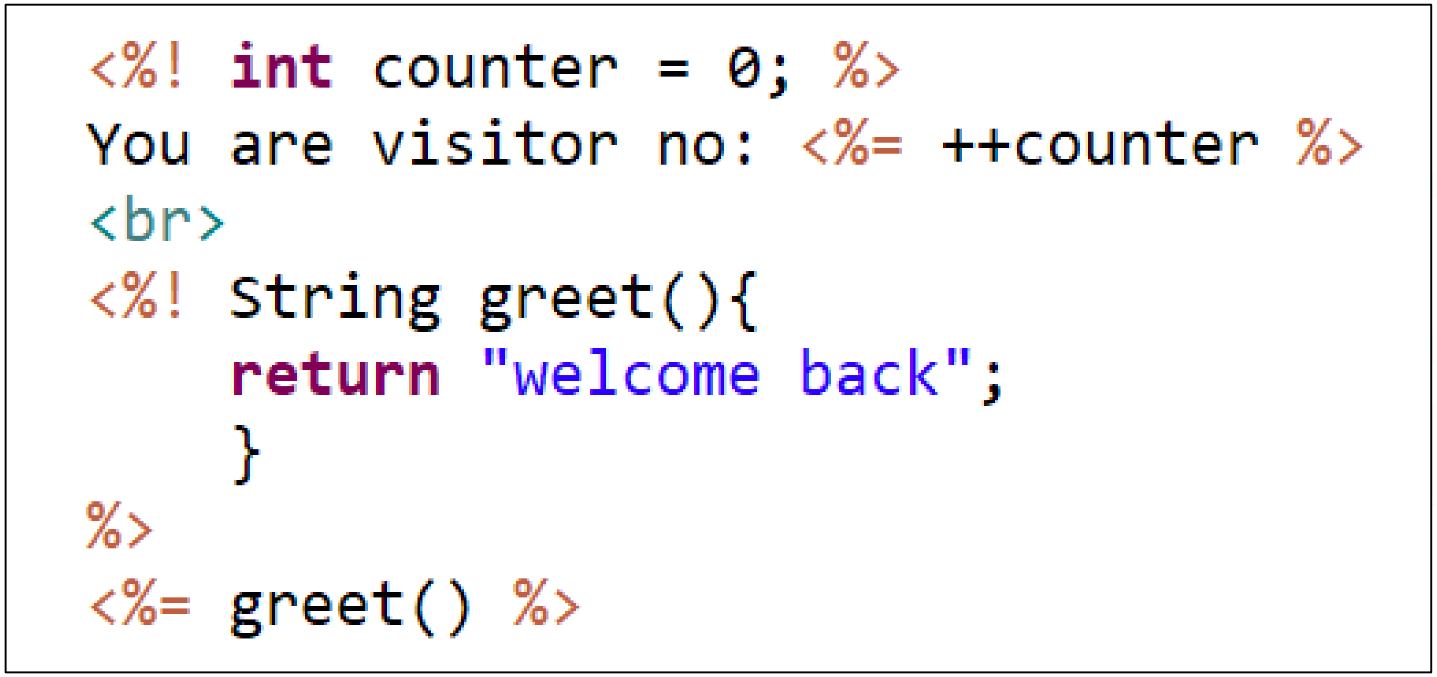


```
<% String name = "Ram"; %>
<%= "Welcome "+name %>
<% int x = 10,y=20; %>
<%= "Sum "+(x+y) %>
```

Declarations

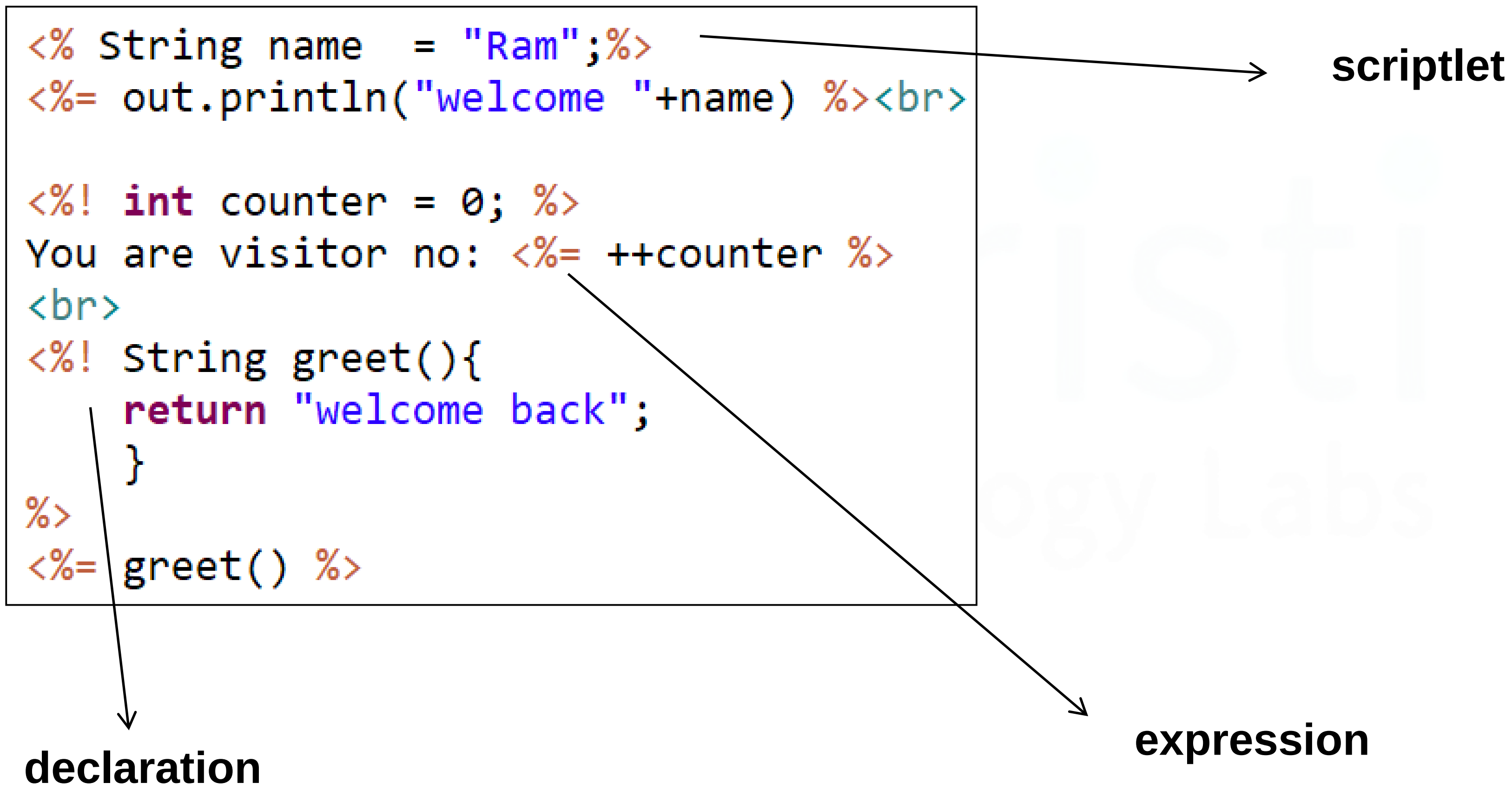
- Declare and define instance variables and methods that can be used in page
- Variable gets initialized only once when page is first loaded(for instance variables)
- The code is within ***<%! ------- --%>***

Example



```
<%! int counter = 0; %>
You are visitor no: <%= ++counter %>
<br>
<%! String greet(){
    return "welcome back";
    }
%>
<%= greet() %>
```

Example – Scripting Elements



```
<% String name  = "Ram";%>
<%= out.println("welcome "+name) %><br>

<%! int counter = 0; %>
You are visitor no: <%= ++counter %>
<br>
<%! String greet(){
    return "welcome back";
    }
%>
<%= greet() %>
```

Directives

- Gives general information of page
- directive always starts with `<%@ - ---------%>`
- Three types
  - **page**
  - **include**
  - **taglib**

Directives

page

- Inform engine of overall properties of JSP page

`<%@page  attributelist %>`

include

- tells engine to include contents of another file (HTML/ JSP) in current page

`<%@include  attributelist %>`

taglib

- Used to add taglibrary to the page

`<%@taglib  attributelist %>`

@page directive - attributes

- Page directive informs the JSP engine of overall properties of page
- applies to entire translation unit and not to just the page
- focus on
  - ***import***
  - ***errorPage***
  - ***isErrorPage***
  - ***isThreadSafe – default(true)***

@page - import

- Similar to 'import' of Java
- Engine inserts import statement in servlet during translation
- Can import multiple packages with comma separation
- auto imports ***java.lang.*, javax.servlet.*, javax.servlet.JSP.* and javax.servlet.http.****
- import is only attribute that can appear more than once in a translation unit

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

@page - errorPage and isErrorPage

- Embedded Java code can throw exceptions
- can use try catch block
- Use declarative handling which separates error handling code from main page
- promotes reusability of exception handling mechanism
- **errorPage** attribute delegates exception to another JSP page that has exception handling code
- error page can be identified by **isErrorPage** attribute

@page - errorPage and isErrorPage

errorPage

- When an exception occurs in the current jsp page, the user will be redirected to errorHandler.jsp

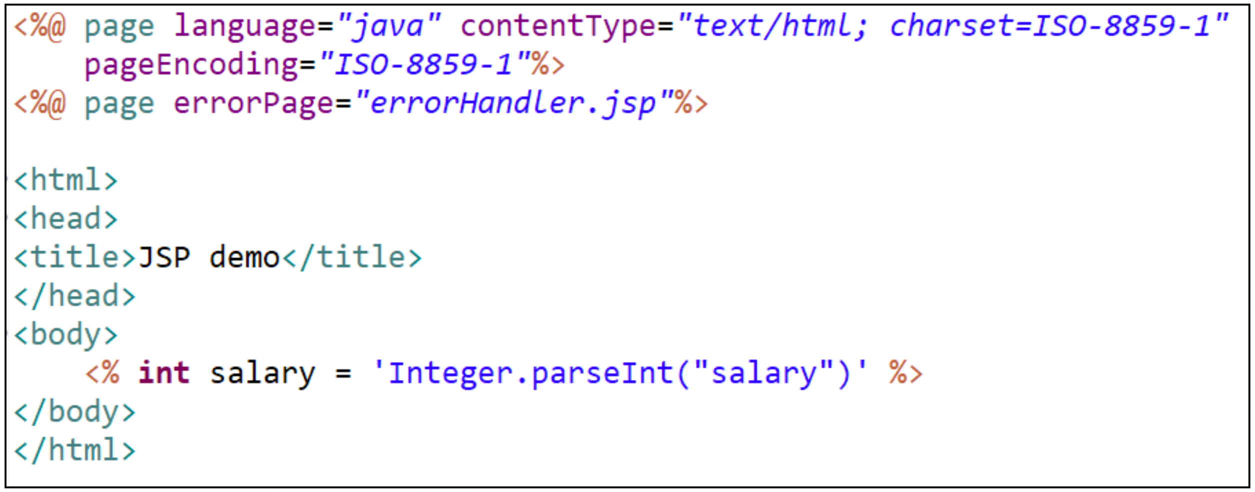
```
<%@ page  errorPage = “errorHandler.jsp”%>
```

isErrorPage

- The user is directed to this page in case of exception

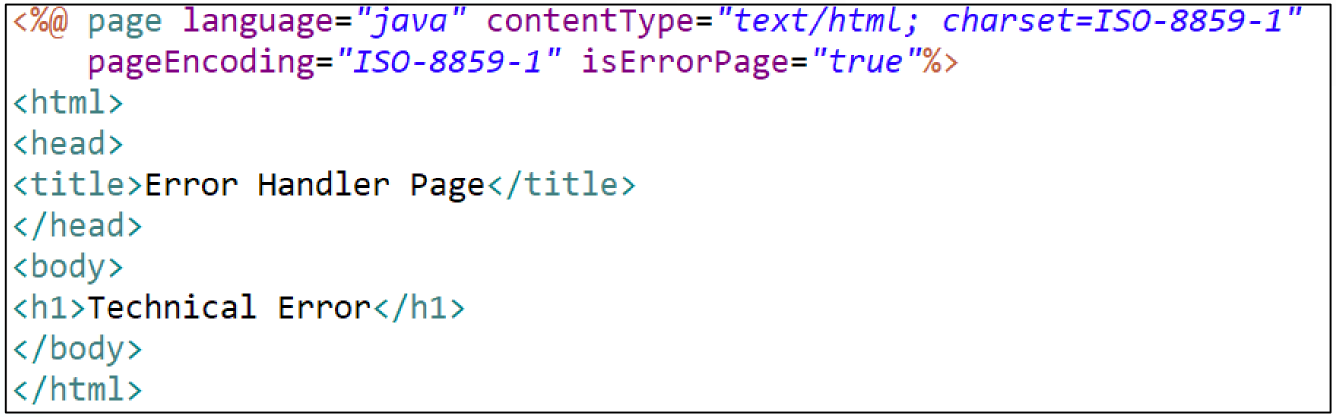
```
<%@ page  isErrorPage = “true”%>
```

Example – errorPage and isErrorPage



```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ page errorPage="errorHandler.jsp"%>

<html>
<head>
<title>JSP demo</title>
</head>
<body>
    <% int salary = 'Integer.parseInt("salary")' %>
</body>
</html>
```

index.jsp



```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" isErrorPage="true"%>
<html>
<head>
<title>Error Handler Page</title>
</head>
<body>
<h1>Technical Error</h1>
</body>
</html>
```

errorHandler.jsp

Handling Exception for whole Application

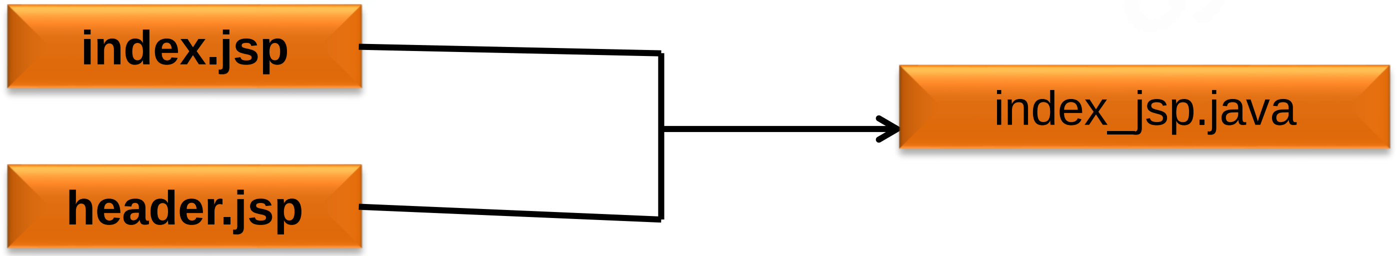
- To handle the exceptions that occur in different servlets and JSP pages

```
<error-page>
<exception-type>java.lang.Exception </exception-type>
<location>/errorHandler.jsp</location>
</error-page>
```

@include

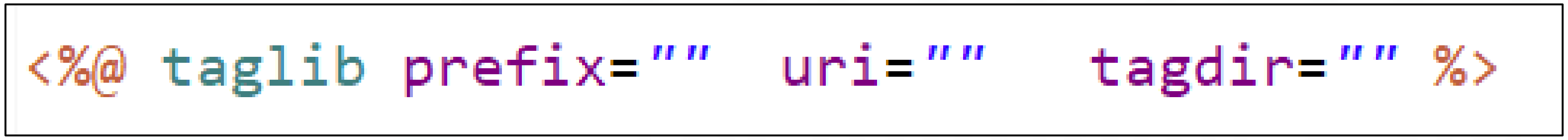
- Is used to include other HTML/jsp pages to the current page
- Happens during the translation time itself.
- Used for static include only

<%@   include file = “header.jsp”%>

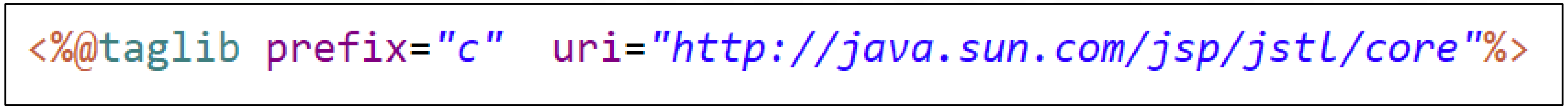


@taglib

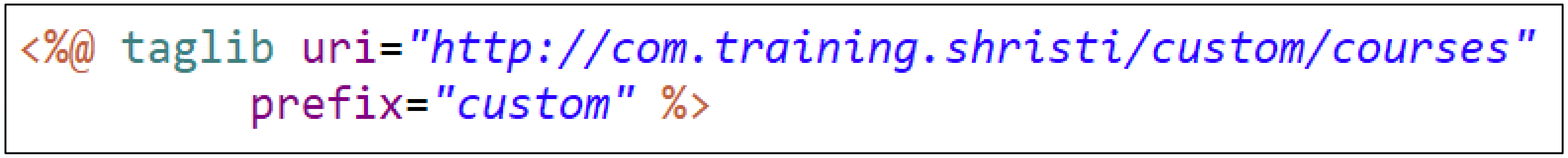
- Is used to add tag libraries to the current page
- A tag library is a collection of tags which provides functionality to the page.
- Can add either an inbuilt tag library like JSTL or custom tag library



```
<%@ taglib prefix=""  uri=""   tagdir="" %>
```



```
<%@taglib prefix="c"  uri="http://java.sun.com/jsp/jstl/core"%>
```



```
<%@ taglib uri="http://com.training.shristi/custom/courses"
        prefix="custom" %>
```

jsp:include

- Used to include output of another JSP page current page
- Happens only during runtime.
- Used for dynamic input
- So ***header_jsp.class*** file also will be generated

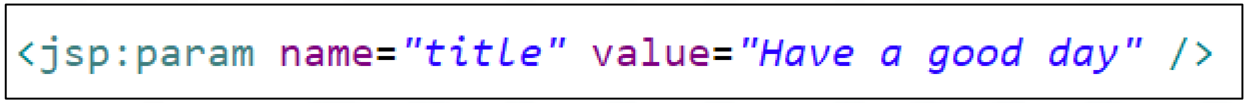
```
<jsp:include  page = "header.jsp" >
```

$$\begin{tikzcd}
\text{index.jsp} \arrow[r] & \text{index\_jsp.java} \arrow[r] & \text{index\_jsp.class} \\
\text{header.jsp} \arrow[r] & \text{header\_jsp.java} \arrow[r] & \text{header\_jsp.class}
\end{tikzcd}$$

jsp:include with jsp:param

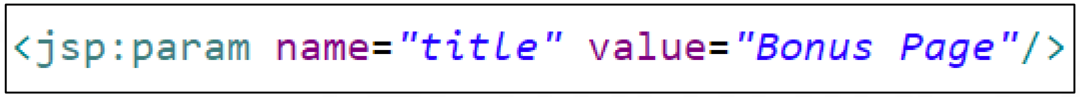
- Use jsp:param to get dynamic value in each page
- param value will change in every jsp page.

index.jsp



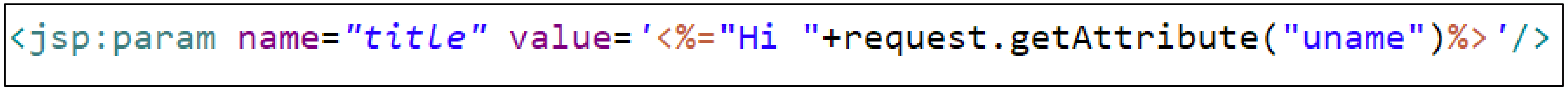
```
<jsp:param name="title" value="Have a good day" />
```

final.jsp



```
<jsp:param name="title" value="Bonus Page"/>
```

success.jsp



```
<jsp:param name="title" value='<%="Hi "+request.getAttribute("uname")%>'/>
```

index.jsp

```
<jsp:include page="mylogo.jsp">
    <jsp:param name="title" value="Have a good day" />
</jsp:include>
```

final.jsp

```
<jsp:include page="mylogo.jsp">
    <jsp:param name="title" value="Bonus Page" />
</jsp:include>
```

success.jsp

```
<jsp:include page="mylogo.jsp">
    <jsp:param name="title"
        value='<%="Hi "+request.getAttribute("uname")%>' />
</jsp:include>
```

mylogo.jsp

```
<div style="background-color: green; color: orange;">
    <strong><i> <%= request.getParameter("title") %>
    </i></strong>
</div>
```

Comments

```
<% -- JSP comment   --%>
<%   // java comment  %>
<!--  HTML comment  -- >
```

Implicit Objects

- request
- response
- out
- session
- config
  - *`config.getInitParameter(String name)`*
- exception
- application
  - *`application.getInitParameter(String name)`*

Scope

page

- objects visible only within page
- thread safe

request

- available to components to which request forwarded or included

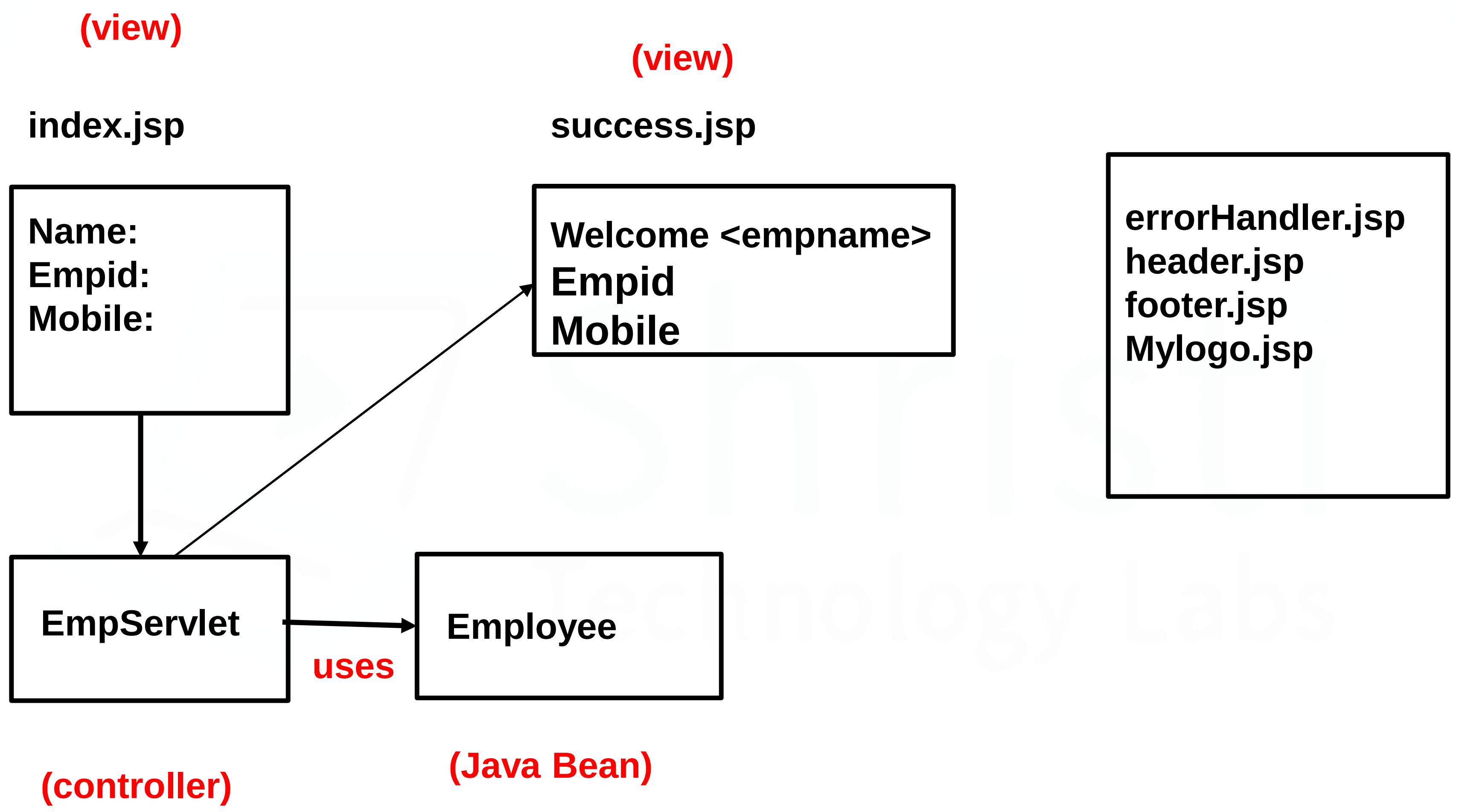
session

- object available to all components that participate in client session
- not thread safe, may synchronize

application

- available for life cycle of application

Project



Project using JSP concepts

Example

index.jsp

Employee.java (java bean)

EmpServlet.java (Servlet)

success.jsp

Example

```
<form action="empServlet">
    Name<input type="text" name="name"><br>
    EmpId<input type="text" name="empid"><br>
    Mobile<input type="text" name="mobile"><br>
    <input type="submit" value="Click here">
</form>
```

index.jsp

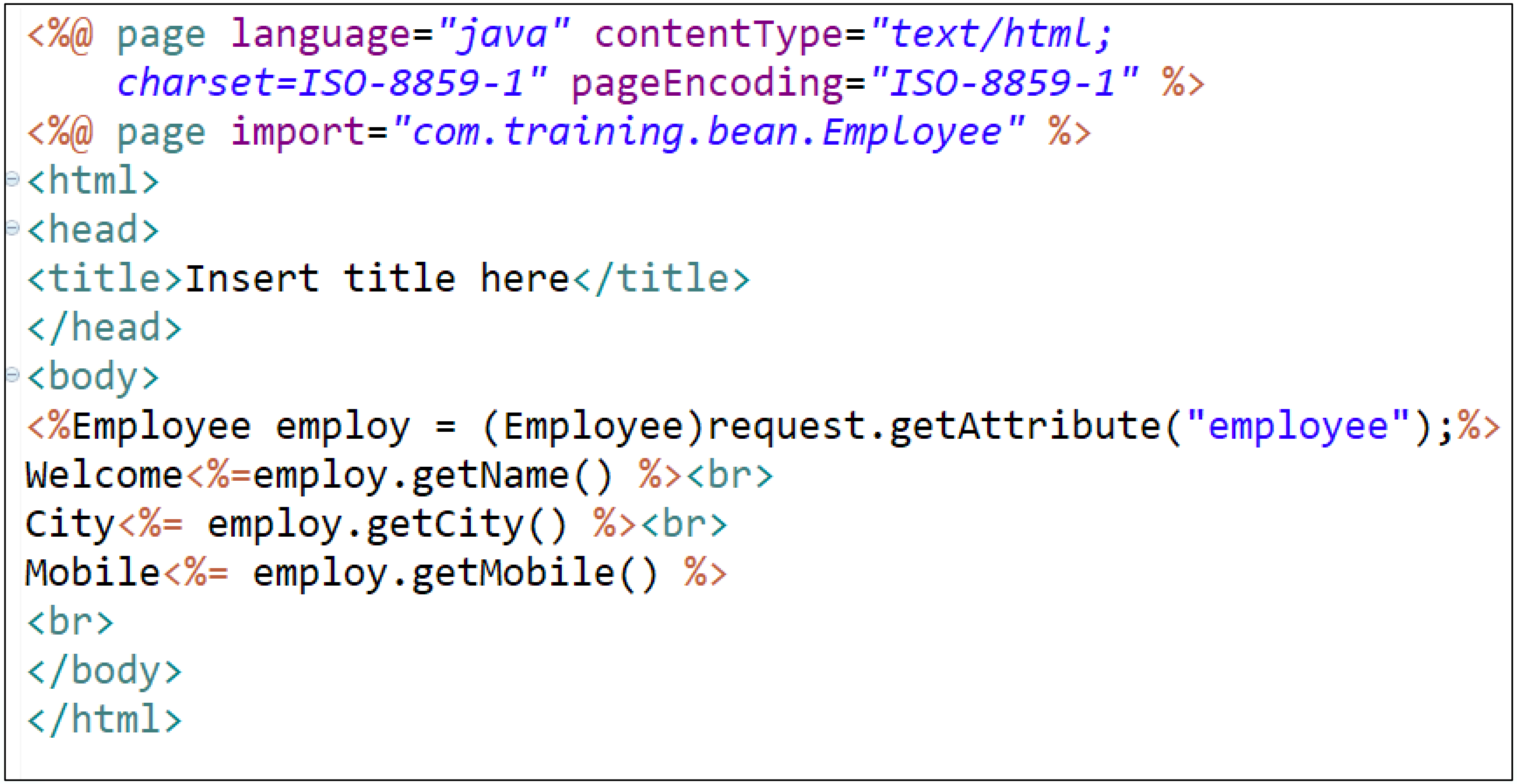
```
public class Employee {
    String name;
    long mobile;
    int empid;
   // getter, setter methods
  //generate toString

}
```

Employee.java

EmpServlet.java

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    String empname = request.getParameter("name");
    String mobile = request.getParameter("mobile");
    String empid = request.getParameter("empid");
    int employid = Integer.parseInt(empid);
    long mob = Long.parseLong(mobile);

    Employee employee =new Employee();
    employee.setName(empname);
    employee.setMobile(mob);
    employee.setEmpid(employid);

    request.setAttribute("employee",employee);
    RequestDispatcher rd = request.getRequestDispatcher("success.jsp");
    rd.forward(request, response);
}
```

success.jsp



```
<%@ page language="java" contentType="text/html;
    charset=ISO-8859-1" pageEncoding="ISO-8859-1" %>
<%@ page import="com.training.bean.Employee" %>
<html>
<head>
<title>Insert title here</title>
</head>
<body>
<%Employee employ = (Employee)request.getAttribute("employee");%>
Welcome<%=employ.getName() %><br>
City<%= employ.getCity() %><br>
Mobile<%= employ.getMobile() %>
<br>
</body>
</html>
```

Expression Language

Overview of EL

Syntax

- `${first.second}`

where

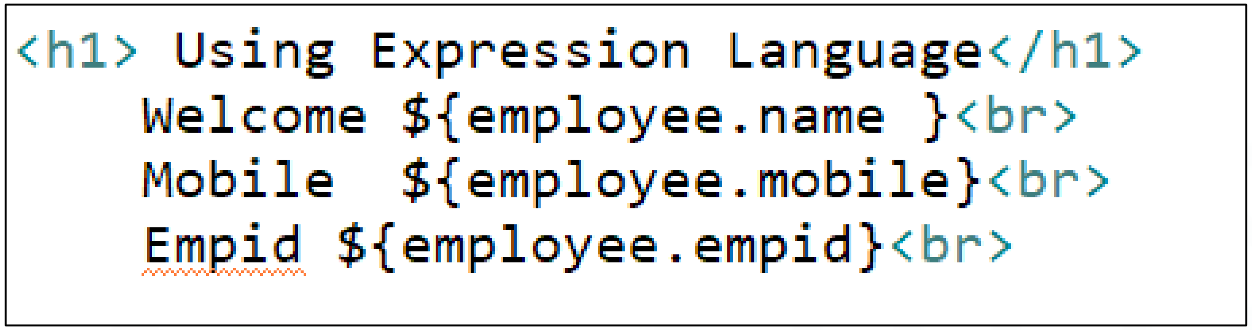
first - can be the dummy name of bean, or array , implicit object

second - can be instance variable of bean, index , methods

- By default el gets evaluated.
- To ignore evaluation, print $ as such use

```
<%@ page  isELIgnored="true" %>
```

success.jsp using EL



```
<h1> Using Expression Language</h1>
    Welcome ${employee.name }<br>
    Mobile  ${employee.mobile}<br>
    Empid ${employee.empid}<br>
```

employee - dummy name of bean
name, mobile - instance variables of bean

Implicit Objects of EL

- pageScope
- requestScope
- sessionScope
- applicationScope
- initParam
- cookie
- param/paramValues

Expression language operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/ or div	Division
% or mod	Modulus (Remainder)
== or =	Equality
!= or !=	Inequality
< or lt	Less than
> or gt	Greater than
<= or le	Less than or equal to
>= or ge	Greater than or equal to
&& or and	Logical AND
|| or or	Logical OR
! or not	Boolean complement
empty	Check for empty value
a ? b : c	Conditional operator

JSTL

Overview of JSTL

- JSP Standard Tag Library (JSTL) is the standard tag library that provides tags to control the JSP page behavior.
- Types of JSTL Tags
  - Core Tag
  - Formatting Tag
  - SQL Tag
  - XML Tag
  - Function Tag

How to Use JSTL

- Download ***jstl.jar*** and ***standard.jar*** *for Tomcat Server(7.0)*
- Download **taglibs-standard-impl-1.2.5*.jar*** and **taglibs-standard-spec-1.2.5*.jar*** *for Tomcat Server(8.0)*
- Copy them to ***WEB-INF/lib*** folder
- Add the taglib in the JSP page as

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

uri is a dummy value which is written in the taglibrary descriptor(tld). Take the **uri** value from the examples in **TOMCAT server** installation.

CORE Tags

Tag	Description
<c:out>	To print in JSP page, can use EL with this tag
<c:set>	To set values in a particular scope
<c:remove>	Remove the variables from particular scope
<c:catch>	All error prone statements are written inside this. Catches them that occurs in its body and optionally exposes it.
<c:if>	Simple conditional logic, Used with EL and we use it process the exception from <c:catch>
<c:import>	Same as <jsp:include> or include directive
<c:param>	used with tag <c:import> to pass dynamic parameter values

CORE Tags

Tag	Description
<c:choose>	Simple conditional tag establish a context for mutually exclusive conditional operations, marked by <c:when> and <c:otherwise>
<c:otherwise>	Sub tag of <c:choose> executes if other conditions evaluate to ‘false’.
<c:forEach>	for iteration over a collection
<c:redirect>	Redirect the request to another resource (new)
<c:url>	to create a URL with optional query string parameters

Example

<c:out>

- To print some message in browser
- **value** is the value to be printed

<c:set>

- Sets the result of an expression evaluation in a 'scope‘
- **var** is the dummy name
- **value** is the value to be added

```
<c:set value="welcome to JSTL" var="message" scope="session">
</c:set>
<c:out value="hello! ${message}"></c:out>
```

Example

<c:remove>

- Removes the value from a particular scope

```
  Removing an attribute:<br>
<c:remove var="val" />
<%=pageContext.getAttribute("val")%><br> <!--  returns null -->
   ${val}<br>  <!-- no output -->
```

Example

<c:catch>

- Takes up statements that may throw exception
- Similar to try block

```
<c:catch>
    <%  int x = 10 / 0; %>
hello will not be printed
</c:catch>
```

Example

<c:if>

- Simple conditional tag. Evaluates if condition is true
- **test** is the attribute to check condition

```
<% pageContext.setAttribute("obj", "Admin");    %>
<c:if test="${obj eq 'Admin'}">
    Hello ${obj}
</c:if>
```

Example

<c:choose>

- a context for mutually exclusive conditional operations, marked by <when> and <otherwise>
- Similar to switch in java

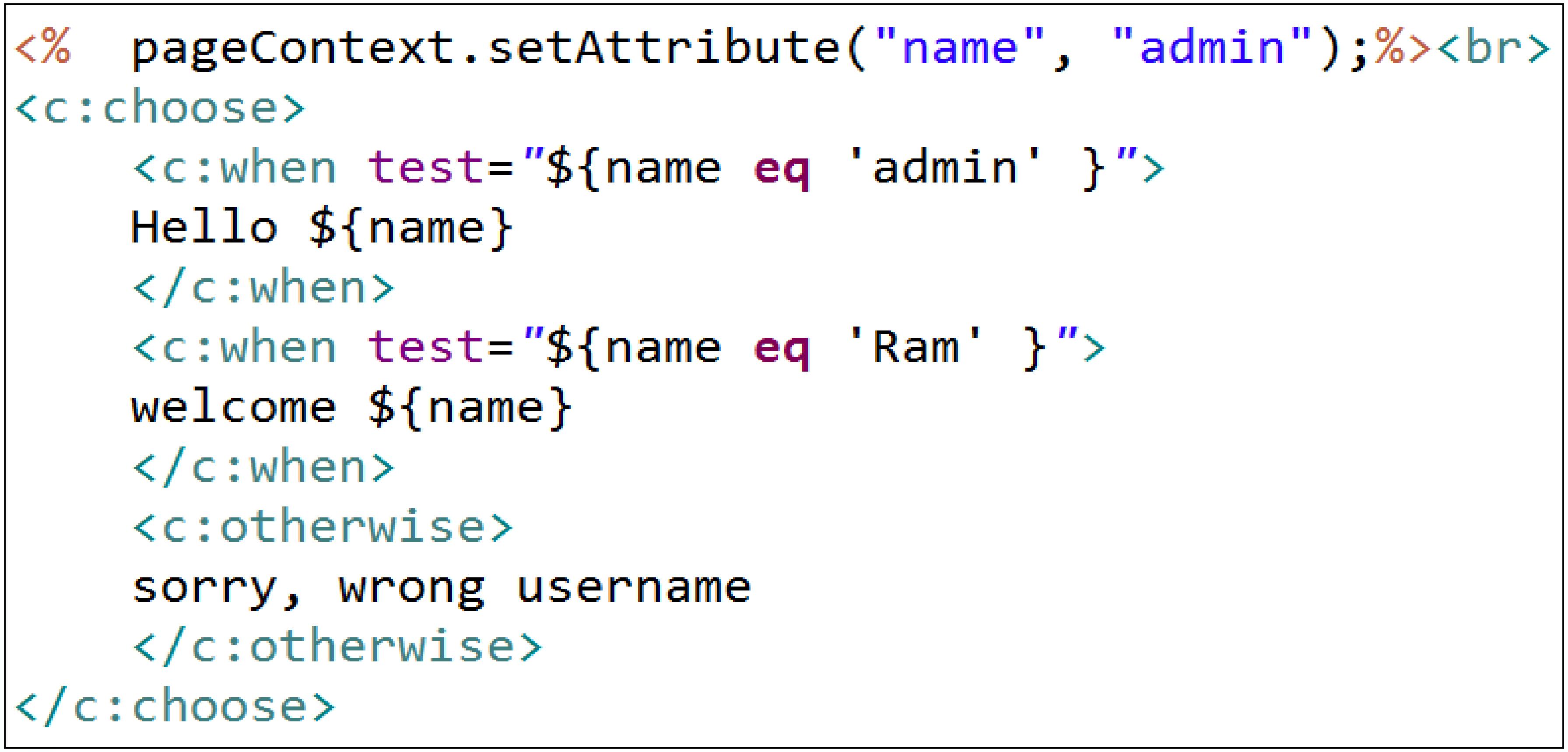
<c:when>

- Subtag of <choose> evaluates its body if its condition is true
- **test** is the attribute to check condition

<c:otherwise>

- Runs only if all the other conditions evaluate to **false**

Example for <c:choose>



```
<%  pageContext.setAttribute("name", "admin");%><br>
<c:choose>
    <c:when test="${name eq 'admin' }">
    Hello ${name}
    </c:when>
    <c:when test="${name eq 'Ram' }">
    welcome ${name}
    </c:when>
    <c:otherwise>
    sorry, wrong username
    </c:otherwise>
</c:choose>
```

Example

<c:forEach>

- Is a basic iterating tag and accepts different collection types
- **var** is the dummy name. **step** for increment
- **items** is the array to be iterated

```
<%  ArrayList<String> al = new ArrayList<String>();
    al.add("Ram");
    al.add("Tom");
    al.add("John");
    al.add("Rohan");
    pageContext.setAttribute("mylist", al);
%>
Values of Arraylist using For loop<br>
<c:forEach var="arr" items="${mylist}" step="2">
    ${arr}<br>
</c:forEach>
```

Example

<c:import>

- Is used to dynamically import pages from a different web application
- Similar to <jsp:include>
- **url** is the path of the web page to be imported from the deployed application

<c:param>

- Has **title** and **value** as attributes

```
<!--similar to jsp:include  -->
<c:import url="http://localhost:8080/Demo/index.jsp">
    <c:param name="title" value="Queens Land"></c:param>
</c:import>
```

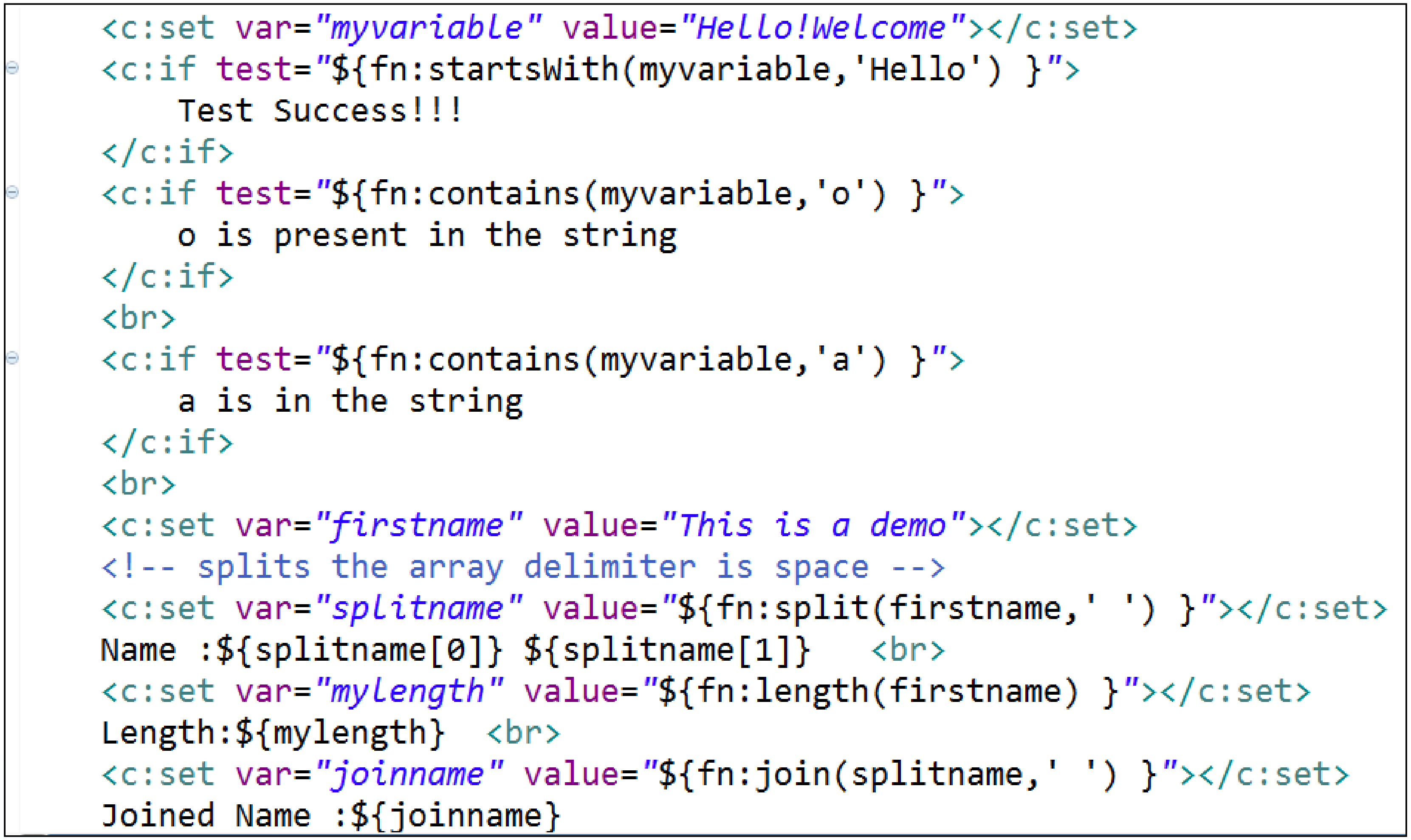
Function Tags

Tag	Description
fn:contains()	an input string contains the specified substring.
fn:containsIgnoreCase()	input string contains the specified substring in a case insensitive way.
fn:endsWith()	input string ends with the specified suffix.
fn:substringAfter()	Returns a subset of a string following a specific substring.
fn:substringBefore()	Returns a subset of a string before a specific substring.
fn:toLowerCase()	Converts all of the characters of a string to lower case.
fn:toUpperCase()	Converts all of the characters of a string to upper case.

Function Tags

Tag	Description
fn:split()	string into array of substrings
fn:startsWith()	an input string starts with the specified prefix
fn:substring()	Returns a subset of a string
fn:trim()	Removes white spaces from both ends of a string.
fn:join()	Joins all elements of array into a string.

Example



```
<c:set var="myvariable" value="Hello!Welcome"></c:set>
<c:if test="${fn:startsWith(myvariable,'Hello') }">
    Test Success!!!
</c:if>
<c:if test="${fn:contains(myvariable,'o') }">
    o is present in the string
</c:if>
<br>
<c:if test="${fn:contains(myvariable,'a') }">
    a is in the string
</c:if>
<br>
<c:set var="firstname" value="This is a demo"></c:set>
<!-- splits the array delimiter is space -->
<c:set var="splitname" value="${fn:split(firstname,' ') }"></c:set>
Name :${splitname[0]} ${splitname[1]}   <br>
<c:set var="mylength" value="${fn:length(firstname) }"></c:set>
Length:${mylength}  <br>
<c:set var="joinname" value="${fn:join(splitname,' ') }"></c:set>
Joined Name :${joinname}
```

Summary

- Introduction to JSP
- Lifecycle phases
- Scripting elements
- Handling Exceptions in JSP
- Implicit objects & scope
- Project
- EL
- JSTL

Shristi
Technology Labs

Thank You