JavaServer Pages Technology

Kien Tran
Principal Software Engineer
12/13/2015

Client Logo

Introduction

- Your role
- Your background and experience in the subject
- What do you want from this course

Course Objectives

- At the end of the course, you will have acquired sufficient knowledge to:
- perform objective 1
- perform objective 2

CSC

Agenda

I.	Section One	xx
II.	Section Two	xx
III.	Section Three	xx
IV.	Section Four	xx
V.	Section Five	xx
VI.	Section Six	xx
VII.	Section Seven	xx

Course Audience and Prerequisite

- The course is for <whom>
- The following are prerequisites to <course>:
  - <knowledge>
  - <experiences>
  - <course>
  - …

Assessment Disciplines

- Class Participation: <%>
- Assignment: <%>
- Final Exam: <%>
- Passing Scores: <%>

Duration and Course Timetable

- Course Duration: <hrs>
- Course Timetable:
  - From <time> to <time>
  - Break <x> minutes from <time> to <time>

Further References

- <Source 1>
- <Source 2>
- …

Set Up Environment

- To complete the course, your PC must install:
  - Software 1
  - Software 2
  - …

Course Administration

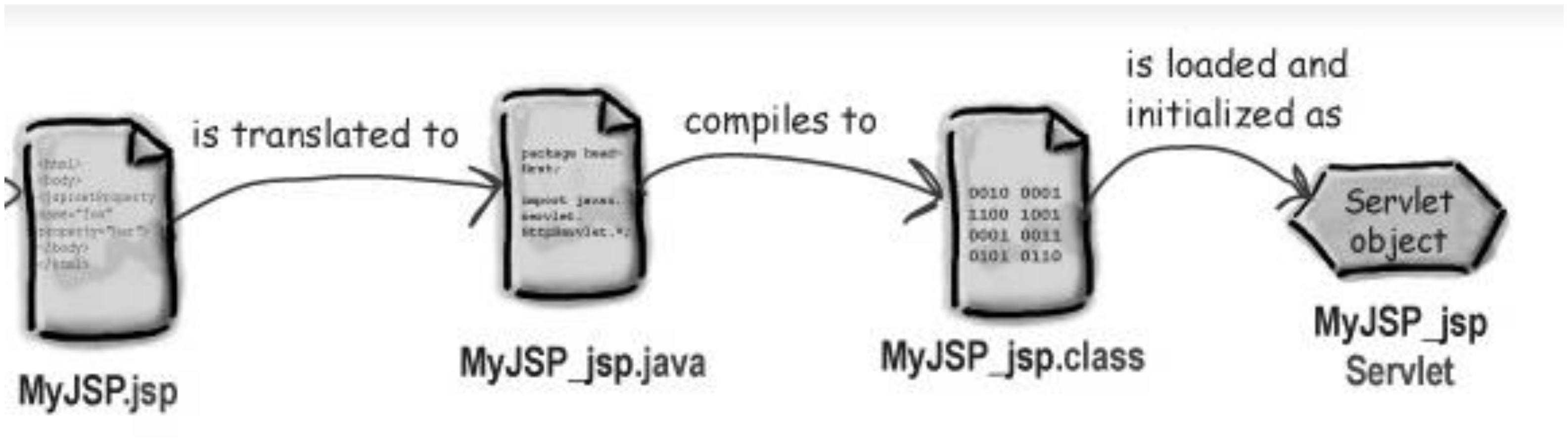
- In order to complete the course you must:
  - Sign in the Class Attendance List
  - Participate in the course
  - Provide your feedback in the End of Course Evaluation

Overview

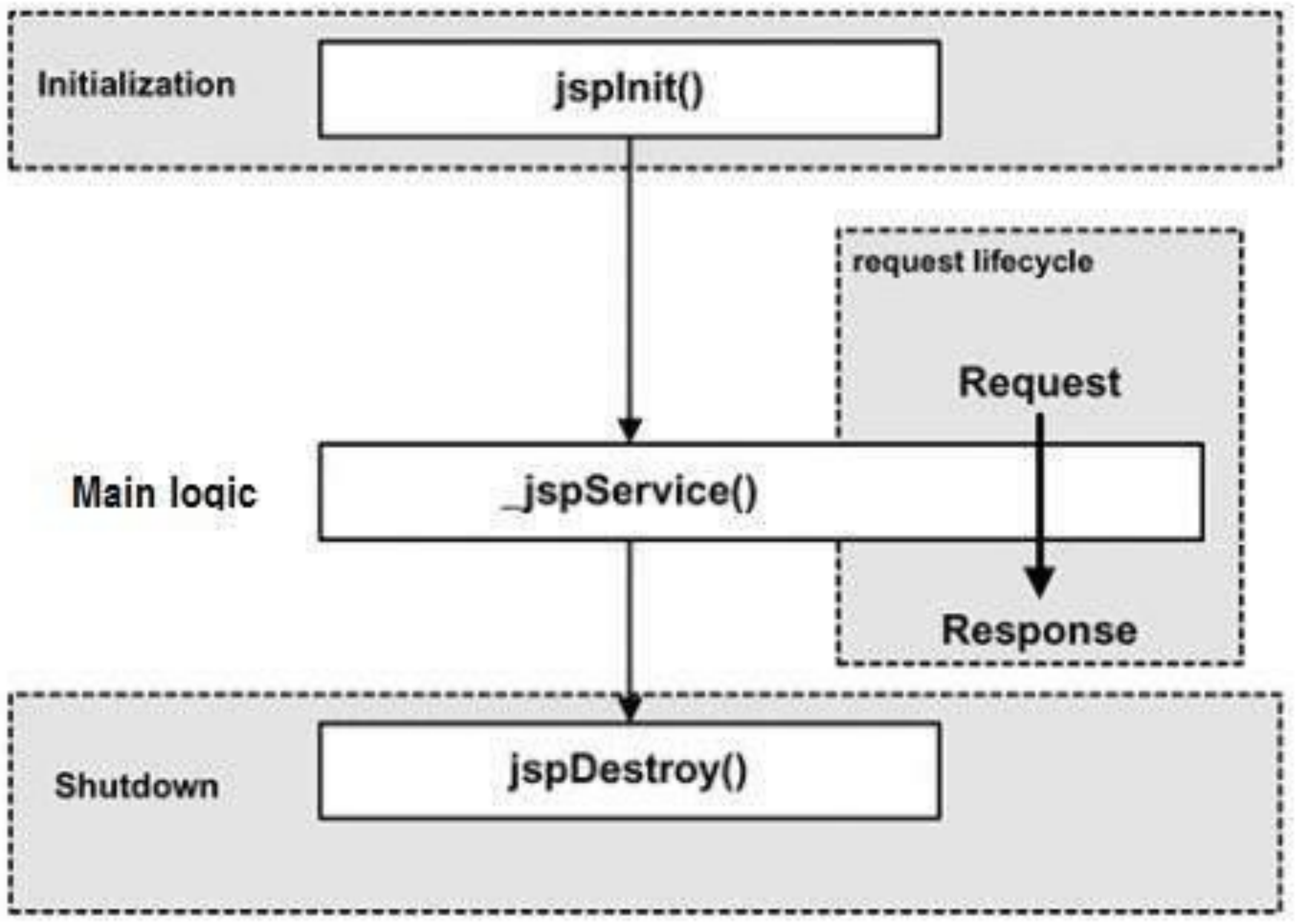
JSP

- JSP technology is used to create web application just like Servlet technology.
- JSP is an extension to servlet because it provides more functionality than servlet such as expression language (EL), JSTL etc.
- A JSP page consists of HTML tags and JSP tags
- The jsp pages are easier to maintain than servlet because we can separate designing and development.

Life cycle of a JSP Page



Life cycle of a JSP Page



JSP Components

- JSP Directives
- Scriptlets
- JSP Action Tags
- JSP Expressions
- JSP Declarations

Scriptlets

Overview

- How to use java code in JSP
- The JSP container moves the scipetlet content into the _jspService() method which is available to the server during processing of the request.
- Syntax: <% code %>

Example

```
<HTML>
<HEAD>
<TITLE>Current Date</TITLE>
</HEAD>
<BODY>

The current date is:
<% out.println(new java.util.Date()); %>

</BODY>
</HTML>
```

JSP Expressions

Overview

- A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

- Syntax: <%= code %>

Example

```
<HTML>
<HEAD>
<TITLE>Current Date</TITLE>
</HEAD>
<BODY>

The current date is:
<%= new java.util.Date() %>

</BODY>
</HTML>
```

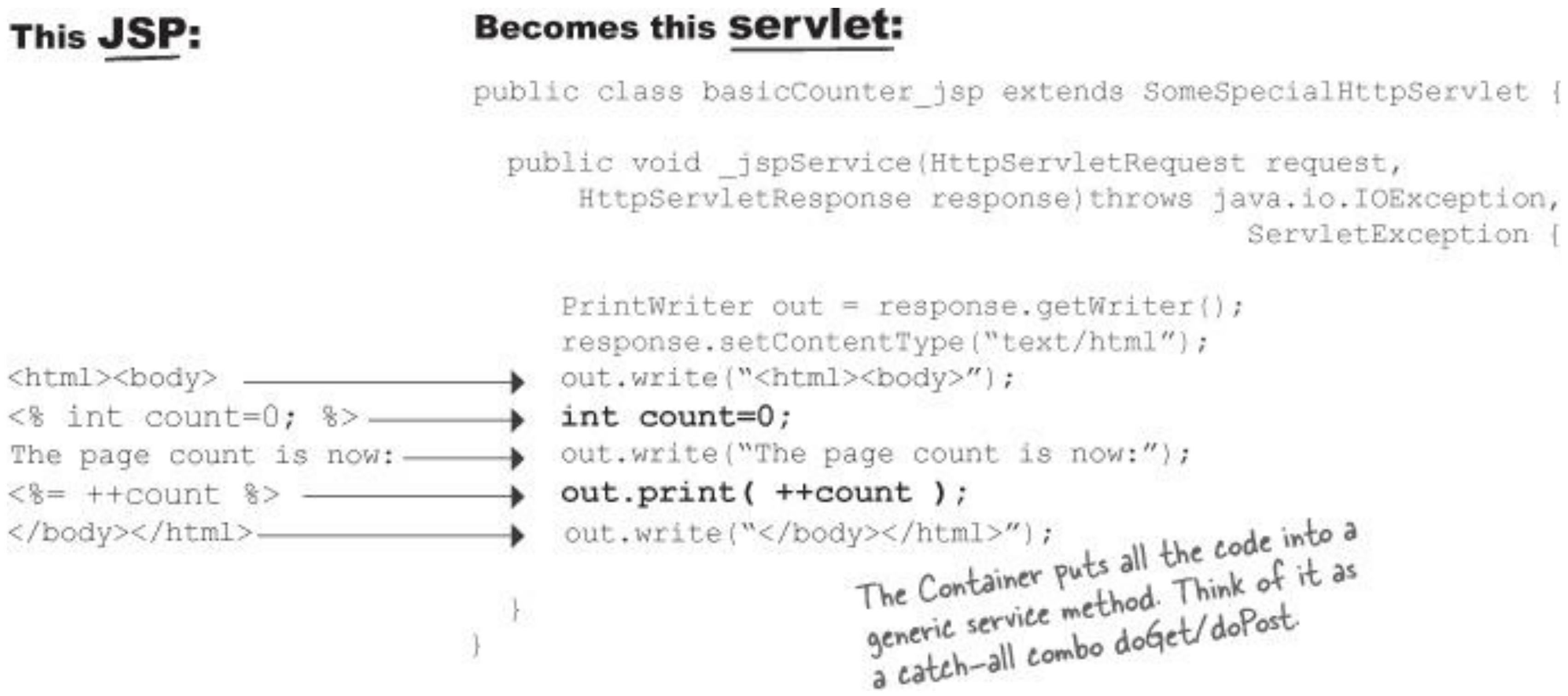
Example

```
<html>
<head>
   <title>JSP expression tag example2</title>
</head>
<body>
  <%
      int a=10;
      int b=20;
      int c=30;
  %>
  <%= a+b+c %>
</body>
</html>
```

Example

```
<html>
<head>
<title>Display Page</title>
</head>
<body>
 <%="This is a String" %><br>
 <%= application.getAttribute("MyName") %>
</body>
</html>
```

What REALLY happens to your JSP code?



This JSP:

```
<html><body>
<% int count=0; %>
The page count is now:
<%= ++count %>
</body></html>
```

Becomes this servlet:

```
public class basicCounter_jsp extends SomeSpecialHttpServlet {

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

     PrintWriter out = response.getWriter();
     response.setContentType("text/html");
     out.write("<html><body>");
     int count=0;
     out.write("The page count is now:");
     out.print( ++count );
     out.write("</body></html>");

  }
}
```

The Container puts all the code into a generic service method. Think of it as a catch-all combo doGet/doPost.

JSP Declarations

Overview

- A declaration declares one or more variables or methods that you can use in Java code later in the JSP file.
- Syntax: <%! code %>

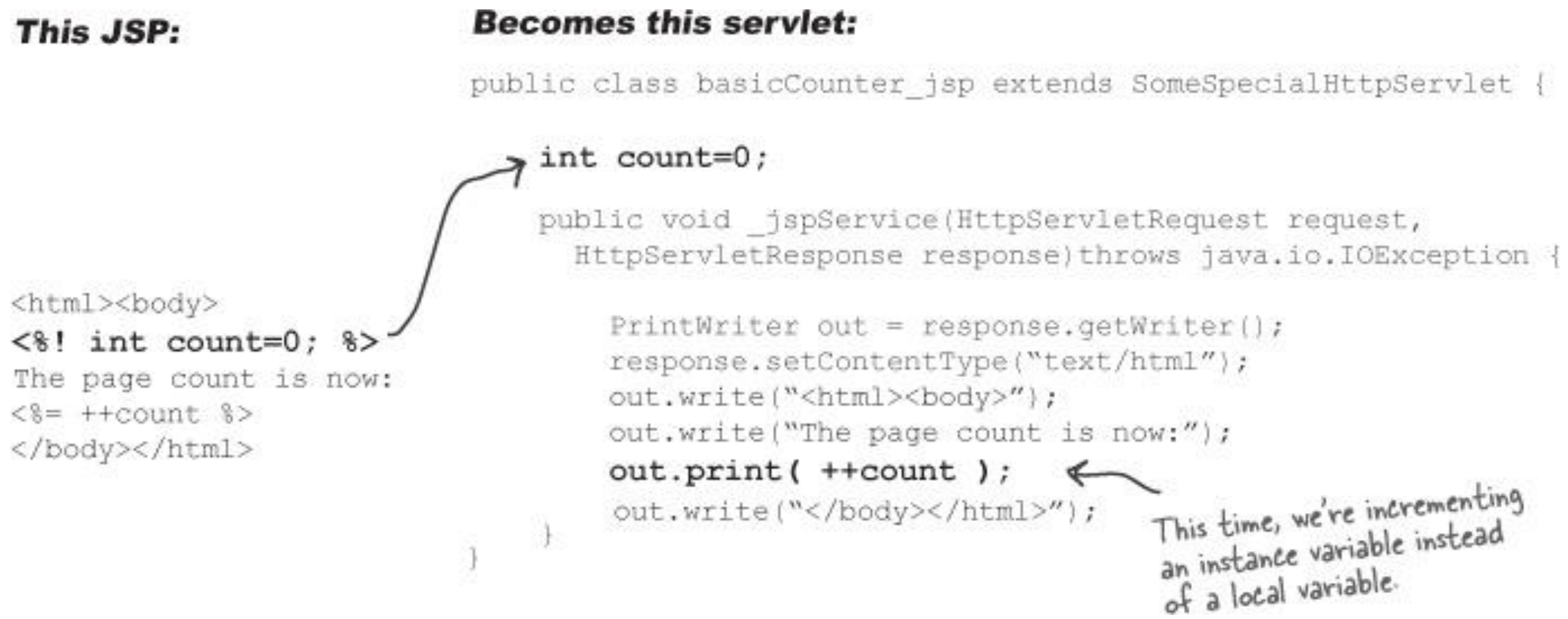
Example

```
 <html>
<head>
 <title>Declaration tag Example1</title>
</head>
<body>
<%! String name="Chaitanya"; %>
<%! int age=27; %>
<%= "Name is: "+ name %><br>
<%= "AGE: "+ age %>
</body>
</html>
```

Example

```
<%! public java.util.Date PrintDate()
    {
      return(new java.util.Date());
    }
%>

<HTML>
<HEAD>
<TITLE>Current Date</TITLE>
</HEAD>
<BODY>

The current date is:
<%= PrintDate() %>

</BODY>
</HTML>
```

What REALLY happens to your JSP code?



This JSP:

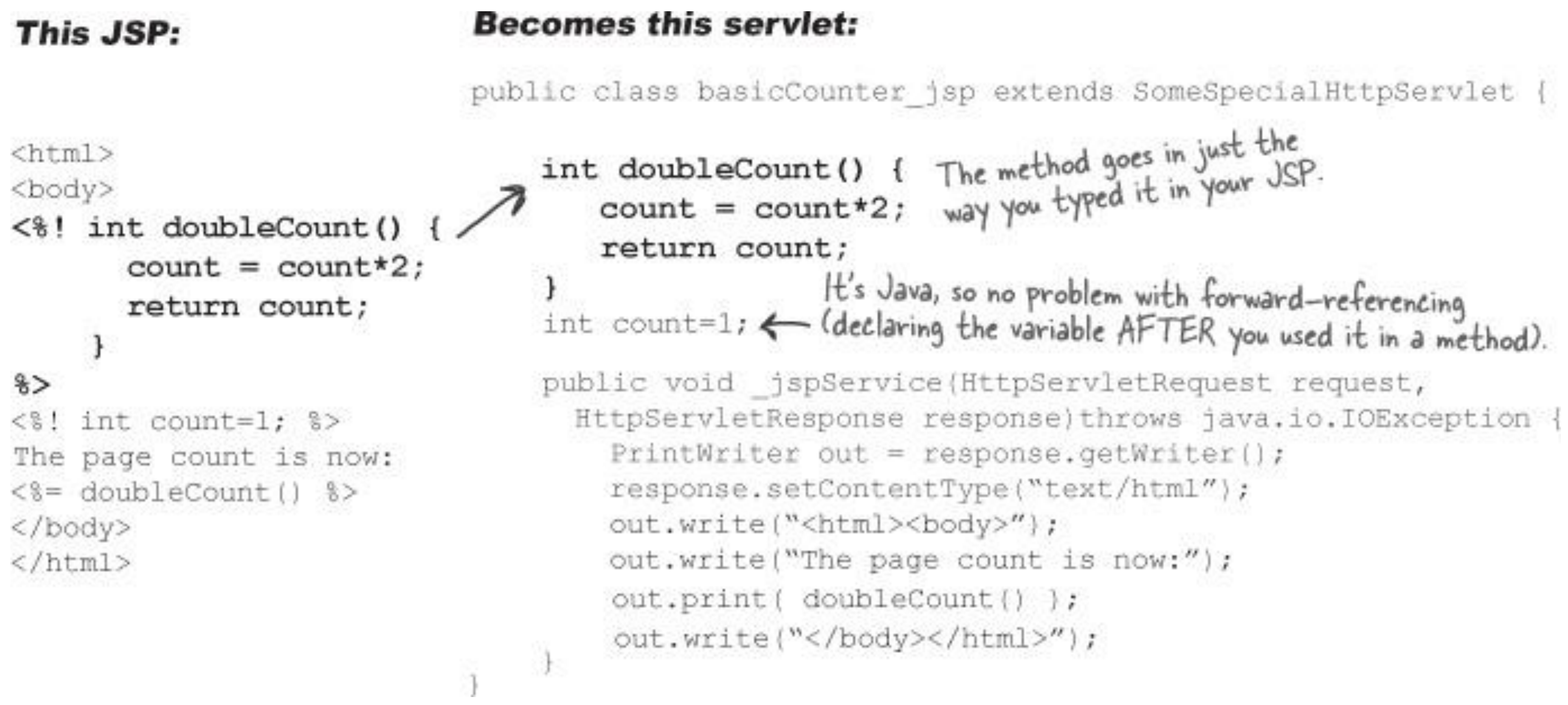
```
<html><body>
<%! int count=0; %>
The page count is now:
<%= ++count %>
</body></html>
```

Becomes this servlet:

```
public class basicCounter_jsp extends SomeSpecialHttpServlet {

    int count=0;

    public void _jspService(HttpServletRequest request,
      HttpServletResponse response)throws java.io.IOException {

        PrintWriter out = response.getWriter();
        response.setContentType("text/html");
        out.write("<html><body>");
        out.write("The page count is now:");
        out.print( ++count );
        out.write("</body></html>");
    }
}
```

This time, we're incrementing an instance variable instead of a local variable.

What REALLY happens to your JSP code?



This JSP:

```
<html>
<body>
<%! int doubleCount() {
      count = count*2;
      return count;
    }
%>
<%! int count=1; %>
The page count is now:
<%= doubleCount() %>
</body>
</html>
```

Becomes this servlet:

```
public class basicCounter_jsp extends SomeSpecialHttpServlet {

    int doubleCount() {
       count = count*2;
       return count;
    }
    int count=1;

    public void _jspService(HttpServletRequest request,
      HttpServletResponse response)throws java.io.IOException {
        PrintWriter out = response.getWriter();
        response.setContentType("text/html");
        out.write("<html><body>");
        out.write("The page count is now:");
        out.print( doubleCount() );
        out.write("</body></html>");
    }
}
```

The method goes in just the way you typed it in your JSP.

It's Java, so no problem with forward-referencing (declaring the variable AFTER you used it in a method).

Comment

A comment...

- <!-- HTML comment -->
- <%-- JSP comment --%>

Example

```
<HTML>
<HEAD>
<TITLE>Using Comments</TITLE>
</HEAD>
<BODY>

<!-- This is a HTML Style comment -->

<%-- This is a JSP Style comment --%>

</BODY>
</HTML>
```

JSP Directives

Overview

- JSP directives are used for controlling the processing of a JSP page. Directives provide information to the server on how the page should be processed.

- Syntax
  - <%@ directive name [attribute name=“value” attribute name=“value” ....]%>

- There are three types of Directives in JSP:
  - Page Directive
  - Include Directive
  - TagLib Directive

Page Directive

- There are several attributes, which are used along with Page Directives
  - import
  - session
  - isErrorPage
  - errorPage
  - ContentType
  - isThreadSafe
  - extends
  - info
  - language
  - autoflush
  - buffer

Page Directive

- Import
  - <%@page import="java.io.*%>
  - <%@page import="java.lang.*%>

- ContentType
  - <%@ page contentType="text/html"%>

Include Directive

- Include directive is used for merging external files to the current JSP page during translation phase

- Syntax
  - <%@include file ="value"%>

- Example
  - <%@include file="myJSP.jsp"%>

Taglib Directive

- This directive basically allows user to use Custom tags in JSP.
- Syntax
  - <%@taglib uri ="taglibURI" prefix="tag prefix"%>
- Example
  - <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

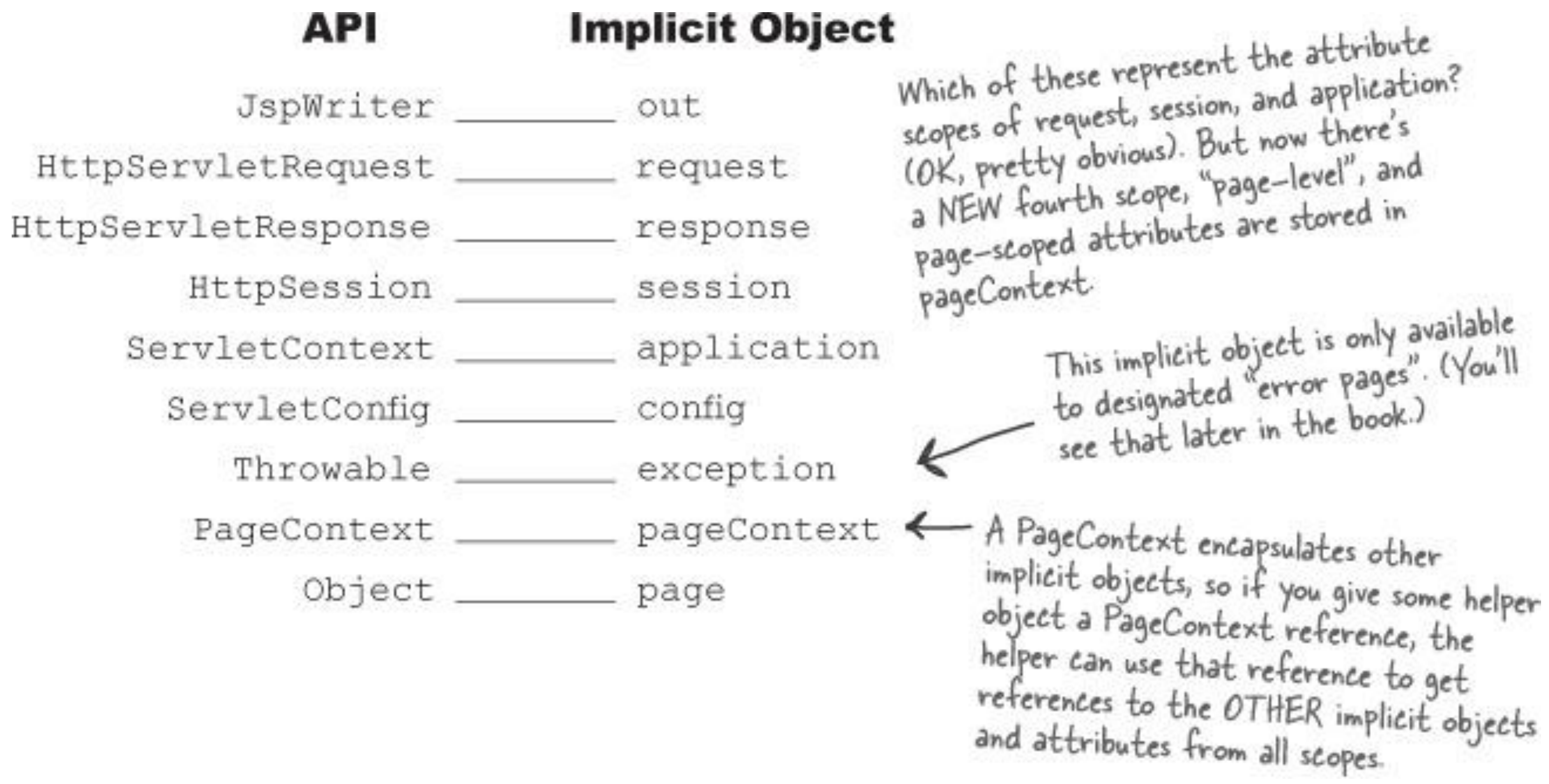
JSP Implicit Objects

Overview

- These objects are created by JSP Engine while translating the JSP page to Servlet.

- These objects are present inside service methods so we can directly use them without declaration.

Implicit Object



API	Implicit Object
JspWriter	out
HttpServletRequest	request
HttpServletResponse	response
HttpSession	session
ServletContext	application
ServletConfig	config
Throwable	exception
PageContext	pageContext
Object	page

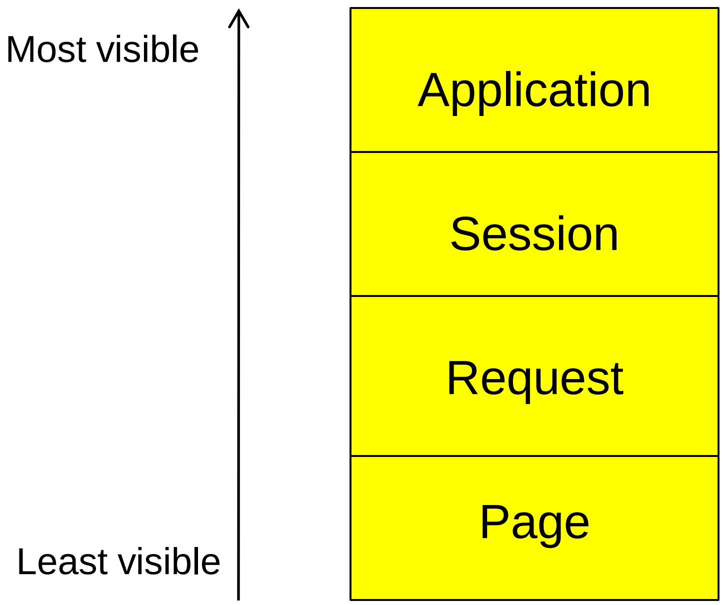
Which of these represent the attribute scopes of request, session, and application? (OK, pretty obvious). But now there's a NEW fourth scope, "page-level", and page-scoped attributes are stored in pageContext.

This implicit object is only available to designated "error pages". (You'll see that later in the book.)

A PageContext encapsulates other implicit objects, so if you give some helper object a PageContext reference, the helper can use that reference to get references to the OTHER implicit objects and attributes from all scopes.

CSC Proprietary and Confidential

Object Scope



Object accessible to pages that belong to the same application

Object accessible to pages that belong to the same session as the one in which they were created

Object accessible from pages processing the request where they were created

Object accessible only within pages where they were created

Attributes in a JSP

	In a servlet	In a JSP (using implicit objects)
Application	**getServletContext()**.setAttribute("foo", barObj);	**application**.setAttribute("foo", barObj);
Request	**request**.setAttribute("foo", barObj);	**request**.setAttribute("foo", barObj);
Session	**request.getSession()**.setAttribute("foo", barObj);	**session**.setAttribute("foo", barObj);
Page	Does not apply!	**pageContext**.setAttribute("foo", barObj);

Using PageContext for attributes

- Using this object you can find attribute, get attribute, set attribute and remove attribute at any of the below levels
  - JSP Page – Scope: PAGE_CONTEXT
  - HTTP Request – Scope: REQUEST_CONTEXT
  - HTTP Session – Scope: SESSION_CONTEXT
  - Application Level – Scope: APPLICATION_CONTEXT

Using pageContext to get and set attributes

- Setting a page-scoped attribute

```
<% Float one = new Float(42.5); %>
<% pageContext.setAttribute("foo", one); %>
```

- Getting a page-scoped attribute

```
<%= pageContext.getAttribute("foo") %>
```

Using pageContext to get and set attributes

- Using the pageContext to set a session-scoped attribute

```
<% Float two = new Float(22.4); %>
<% pageContext.setAttribute("foo", two, PageContext.SESSION_SCOPE); %>
```

- Using the pageContext to get a session-scoped attribute

```
<%= pageContext.getAttribute("foo", PageContext.SESSION_SCOPE) %>
(Which is identical to: <%= session.getAttribute("foo") %> )
```

- Using the pageContext to find an attribute when you don’t know the scope

```
<%= pageContext.findAttribute("foo") %>
```

Example - index.html

```
<html>
<head>
<title> User Login Page – Enter details</title>
</head>
<body>
<form action="validation.jsp">
        Enter User-Id: <input type="text" name="uid"><br>
        Enter Password: <input type="text" name="upass"><br>
        <input type="submit" value="Login">
</form>
</body>
</html>
```

Example - validation.jsp

```
<html>
<head> <title> Validation JSP Page</title>
</head>
<body>
<%
    String id=request.getParameter("uid");
    String pass=request.getParameter("upass");
    out.println("hello "+id);
    pageContext.setAttribute("UName", id, PageContext.SESSION_SCOPE);
    pageContext.setAttribute("UPassword", pass, PageContext.SESSION_SCOPE);
%>
<a href="display.jsp">Click here to see what you have entered </a>
</body>
</html>
```

Example - display.jsp

```
<html>
<head>
<title>Displaying User Details</title>
</head>
<body>
<%
        String username= (String) pageContext.getAttribute("UName",
                                                                    PageContext.SESSION_SCOPE);
        String userpassword= (String) pageContext.getAttribute("UPassword",
                                                                    PageContext.SESSION_SCOPE);
        out.println("Hi "+username);
        out.println("Your Password is: "+userpassword);
%>
</body>
</html>
```

JavaBean

Overview

- A JavaBean is a specially constructed Java class and coded according to the JavaBeans API specifications.
  - It provides a default, no-argument constructor.
  - It should be serializable and implement the Serializable interface.
  - It may have a number of "getter" and "setter" methods for the properties.

Example

```
public class StudentsBean implements java.io.Serializable
{
   private String firstName = null;
   private String lastName = null;
   public StudentsBean() {
   }
   public String getFirstName(){
      return firstName;
   }
   public String getLastName(){
      return lastName;
   }
public void setFirstName(String firstName){
      this.firstName = firstName;
   }
   public void setLastName(String lastName){
      this.lastName = lastName;
   }
}
```

JSP Action Tags

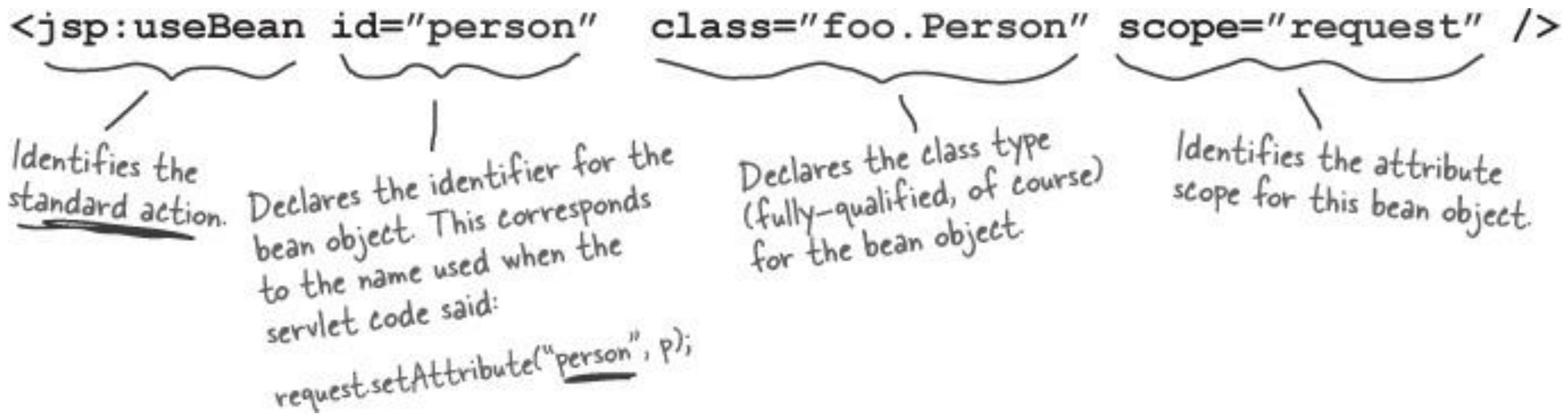
<jsp:useBean>

- This action is useful when you want to use Beans in a JSP page

- Syntax

```
<jsp:useBean id= "instanceName"
                    scope= "page | request | session | application"
                    class= "packageName.className"
                    type= "packageName.className"
beanName="packageName.className | <%= expression >" >
</jsp:useBean>
```

<jsp:useBean>


<jsp:useBean id="person" class="foo.Person" scope="request" />

Identifies the standard action.

Declares the identifier for the bean object. This corresponds to the name used when the servlet code said:

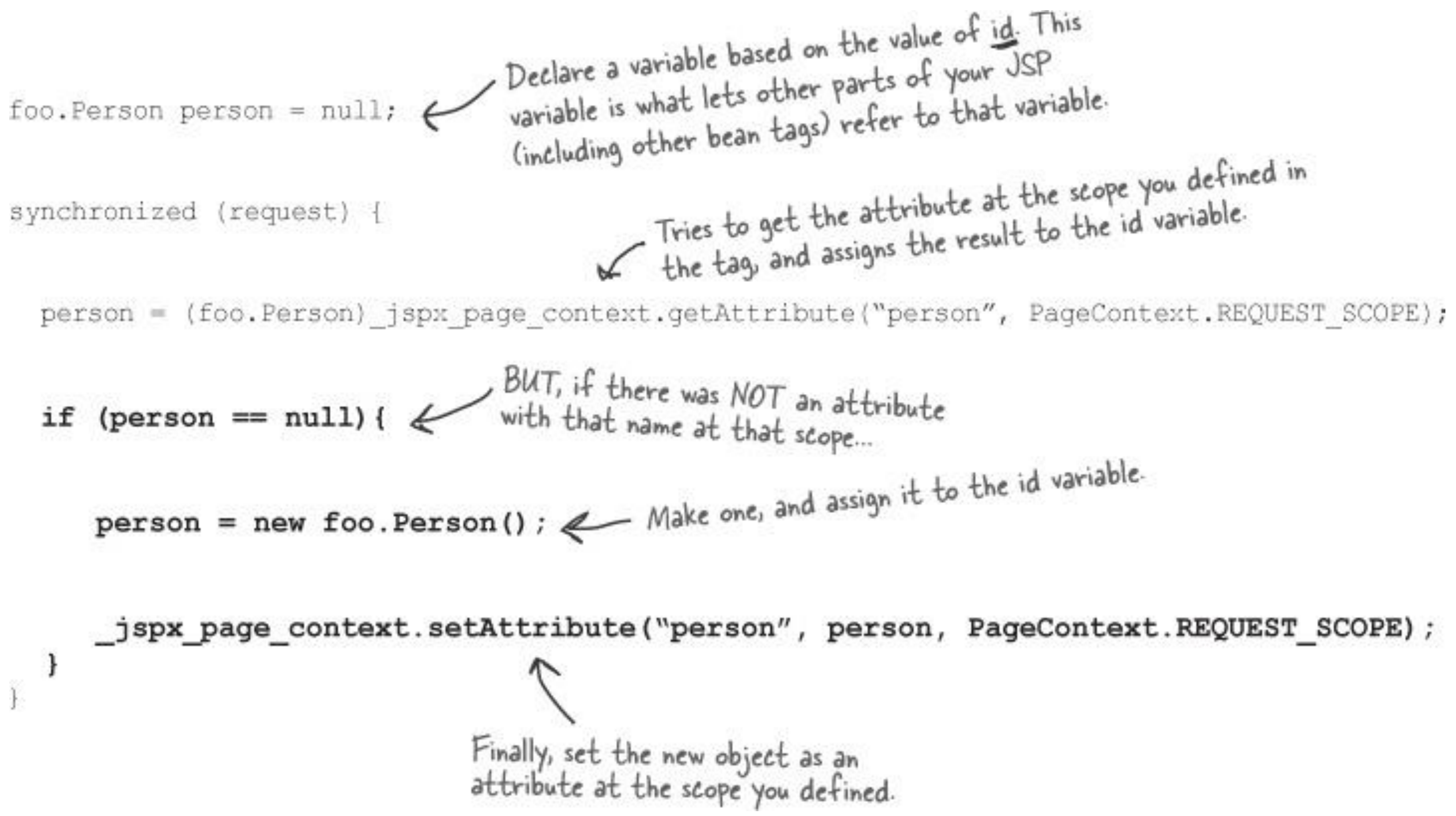
request.setAttribute("person", p);

Declares the class type (fully-qualified, of course) for the bean object.

Identifies the attribute scope for this bean object.

<jsp:useBean> can also CREATE a bean!

```
<jsp:useBean id="person"  class="foo.Person" scope="request" />
```



```
foo.Person person = null;

synchronized (request) {

  person = (foo.Person)_jspx_page_context.getAttribute("person", PageContext.REQUEST_SCOPE);

  if (person == null){

    person = new foo.Person();

    _jspx_page_context.setAttribute("person", person, PageContext.REQUEST_SCOPE);
  }
}
```

Declare a variable based on the value of id. This variable is what lets other parts of your JSP (including other bean tags) refer to that variable.

Tries to get the attribute at the scope you defined in the tag, and assigns the result to the id variable.

BUT, if there was NOT an attribute with that name at that scope...

Make one, and assign it to the id variable.

Finally, set the new object as an attribute at the scope you defined.

<jsp:getProperty>

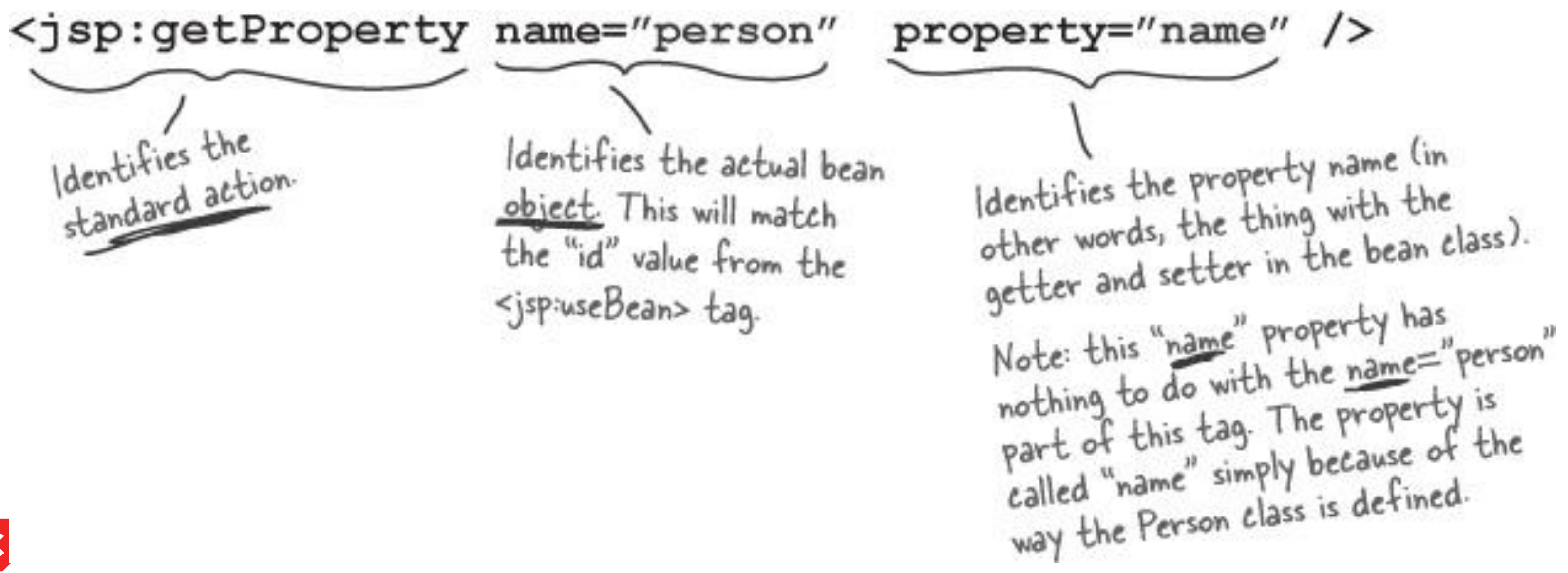
- Get a bean attribute’s property value with <jsp:getProperty>

- Syntax

  <jsp: useBean id="unique_name_to_identify_bean" class="package_name.class_name" />

  <jsp:getProperty name="unique_name_to_identify_bean" property="property_name" />

- Example



```
<jsp:getProperty name="person"  property="name"  />
```

Identifies the standard action.

Identifies the actual bean object. This will match the "id" value from the <jsp:useBean> tag.

Identifies the property name (in other words, the thing with the getter and setter in the bean class).

Note: this "name" property has nothing to do with the name="person" part of this tag. The property is called "name" simply because of the way the Person class is defined.

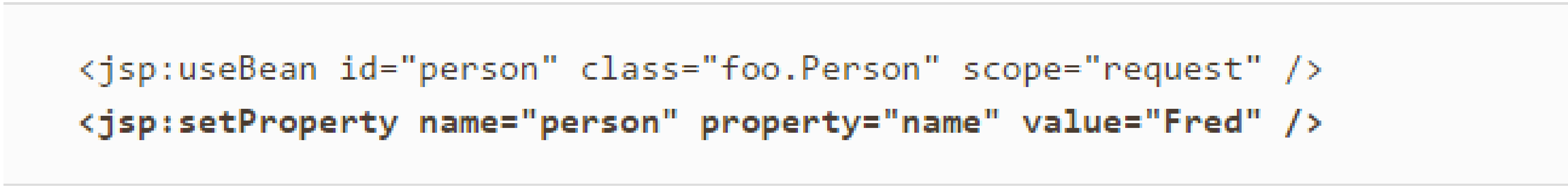
<jsp:setProperty>

- This action tag is used to set the property of a Bean

- Syntax

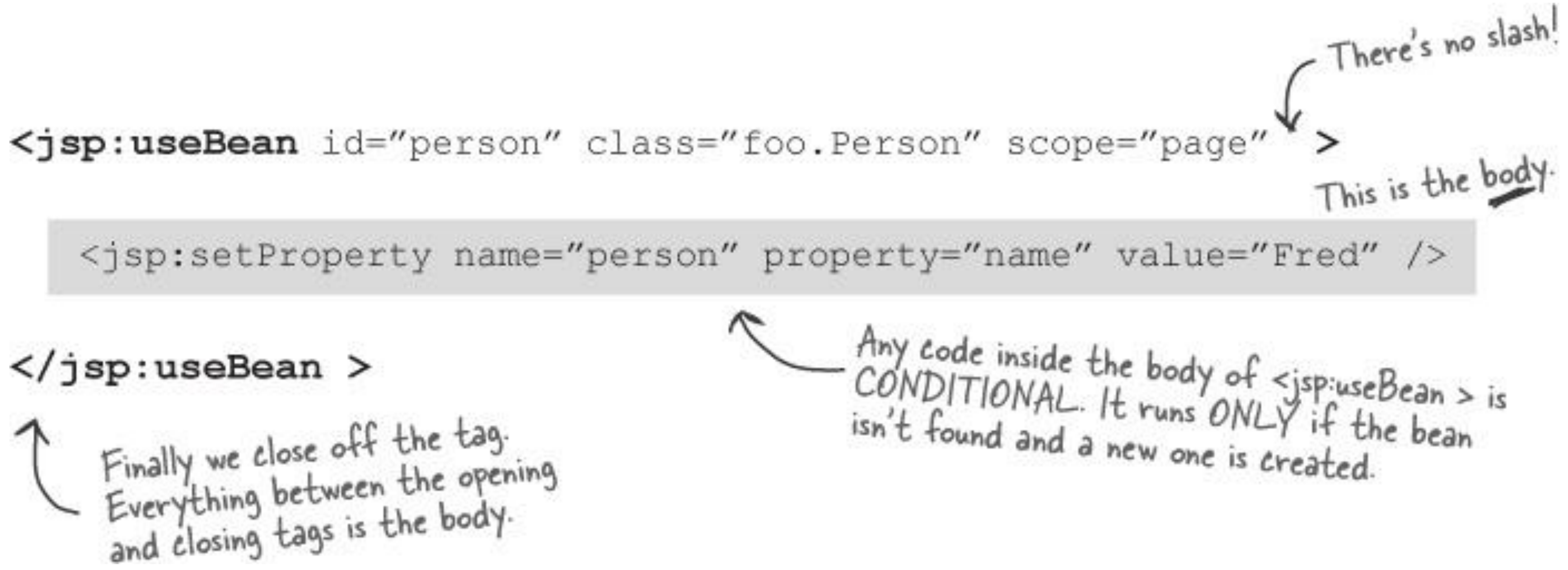
```
<jsp: useBean id="unique_name_to_identify_bean"  class="package_name.class_name" />
<jsp:setProperty name="unique_name_to_identify_bean" property="property_name" />
   OR
<jsp: useBean id="unique_name_to_identify_bean"  class="package_name.class_name">
                <jsp:setProperty name="unique_name_to_identify_bean" property="property_name" />
</jsp:useBean>
```

- Example



```
<jsp:useBean id="person" class="foo.Person" scope="request" />
<jsp:setProperty name="person" property="name" value="Fred" />
```

<jsp:useBean> can have a body!



There's no slash!

```
<jsp:useBean id="person" class="foo.Person" scope="page" >

    <jsp:setProperty name="person" property="name" value="Fred" />

</jsp:useBean >
```

This is the body.

Any code inside the body of <jsp:useBean > is CONDITIONAL. It runs ONLY if the bean isn't found and a new one is created.

Finally we close off the tag. Everything between the opening and closing tags is the body.

Example

```
<jsp:useBean id="students"
                    class="com.tutorialspoint.StudentsBean">
          <jsp:setProperty name="students" property="firstName"
                                                  value="Zara"/>
          <jsp:setProperty name="students" property="lastName"
                                                  value="Ali"/>
</jsp:useBean>

<p>Student First Name:
                    <jsp:getProperty name="students" property="firstName"/>
</p>
<p>Student Last Name:
                    <jsp:getProperty name="students" property="lastName"/>
</p>
```

<jsp:include>

- The include action can be used to insert the output of both static and dynamic pages into the current page.
- Syntax
  - <jsp:include page="page URL" flush="Boolean Value" />
  - Page: The relative URL of the page to be included.
  - Flush: The flush attribute determines whether the included resource has its buffer flushed before it is included.
- Example
  - <jsp:include page="date.jsp" flush="true" />

Example - index.jsp

```
<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="name" value="Chaitanya Pratap Singh" />
        <jsp:param name="age" value="27" />
</jsp:include>

</body>
</html>
```

Example - display.jsp

```
<html>
<head>
<title>Display Page</title>
</head>
<body>
<h2>Hello this is a display.jsp Page</h2>
        User Name: <%=request.getParameter("name") %><br>
        Age: <%=request.getParameter("age") %>
</body>
</html>
```

<jsp:forward>

- The forward action terminates the action of the current page and forwards the request to another resource, such as a static page, another JSP page, or a Java Servlet.

- Syntax
  - <jsp:forward page="URL of the another static, JSP OR Servlet page" />

- Example
  - <jsp:forward page="second.jsp" />

<jsp:param>

- This action is useful for passing the parameters to Other JSP action tags such as JSP include & JSP forward tag.

- Syntax
  – <jsp: param name="param_name_here" value="value_of_parameter_here" />

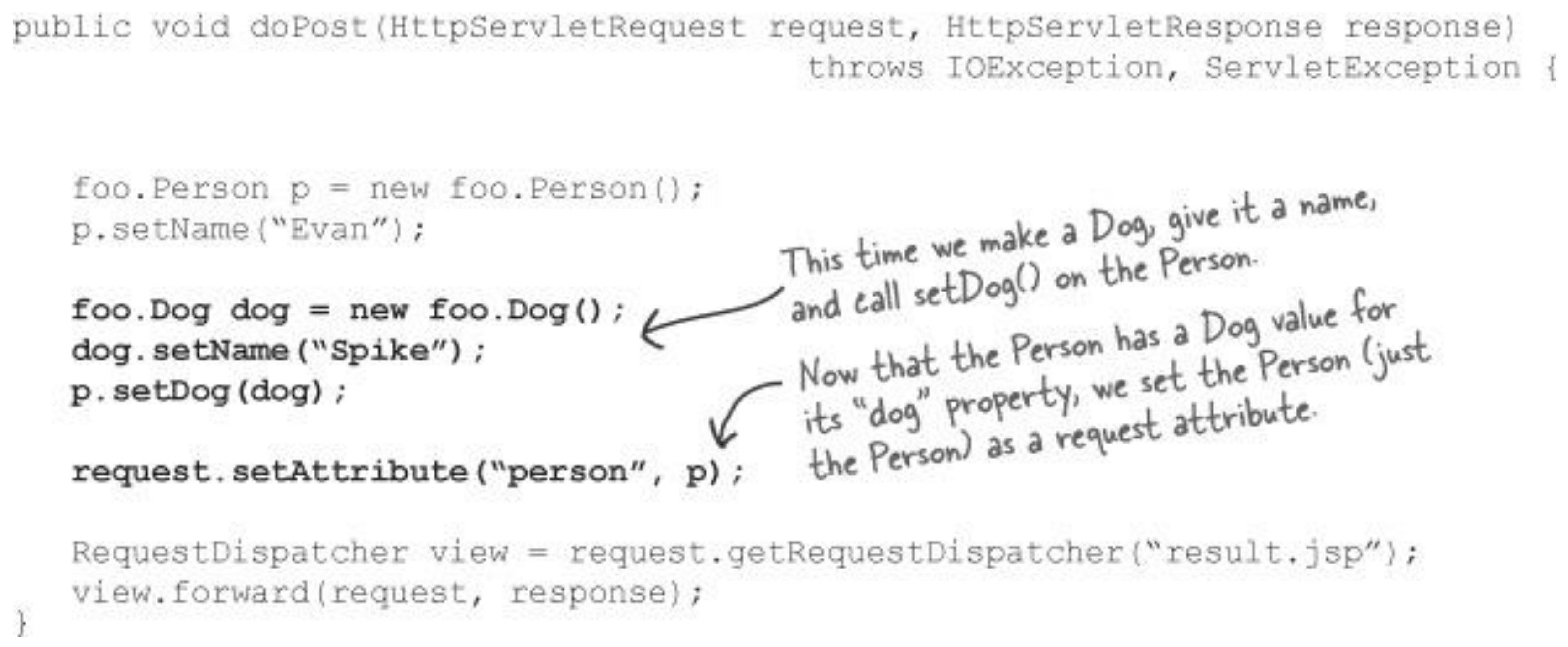
- Example

```
        <jsp:forward page="second.jsp">
                <jsp:param name ="data" value="ABC" />
        </jsp:forward>

        My Data:<%= request.getParameter("data") %>
```

JSP Expression Language (EL)

JSP Expression Language (EL)

- Servlet code



```
public void doPost(HttpServletRequest request, HttpServletResponse response)
                                        throws IOException, ServletException {

   foo.Person p = new foo.Person();
   p.setName("Evan");

   foo.Dog dog = new foo.Dog();
   dog.setName("Spike");
   p.setDog(dog);

   request.setAttribute("person", p);

   RequestDispatcher view = request.getRequestDispatcher("result.jsp");
   view.forward(request, response);
}
```

This time we make a Dog, give it a name, and call setDog() on the Person.

Now that the Person has a Dog value for its "dog" property, we set the Person (just the Person) as a request attribute.

JSP Expression Language (EL)

- Using scripting

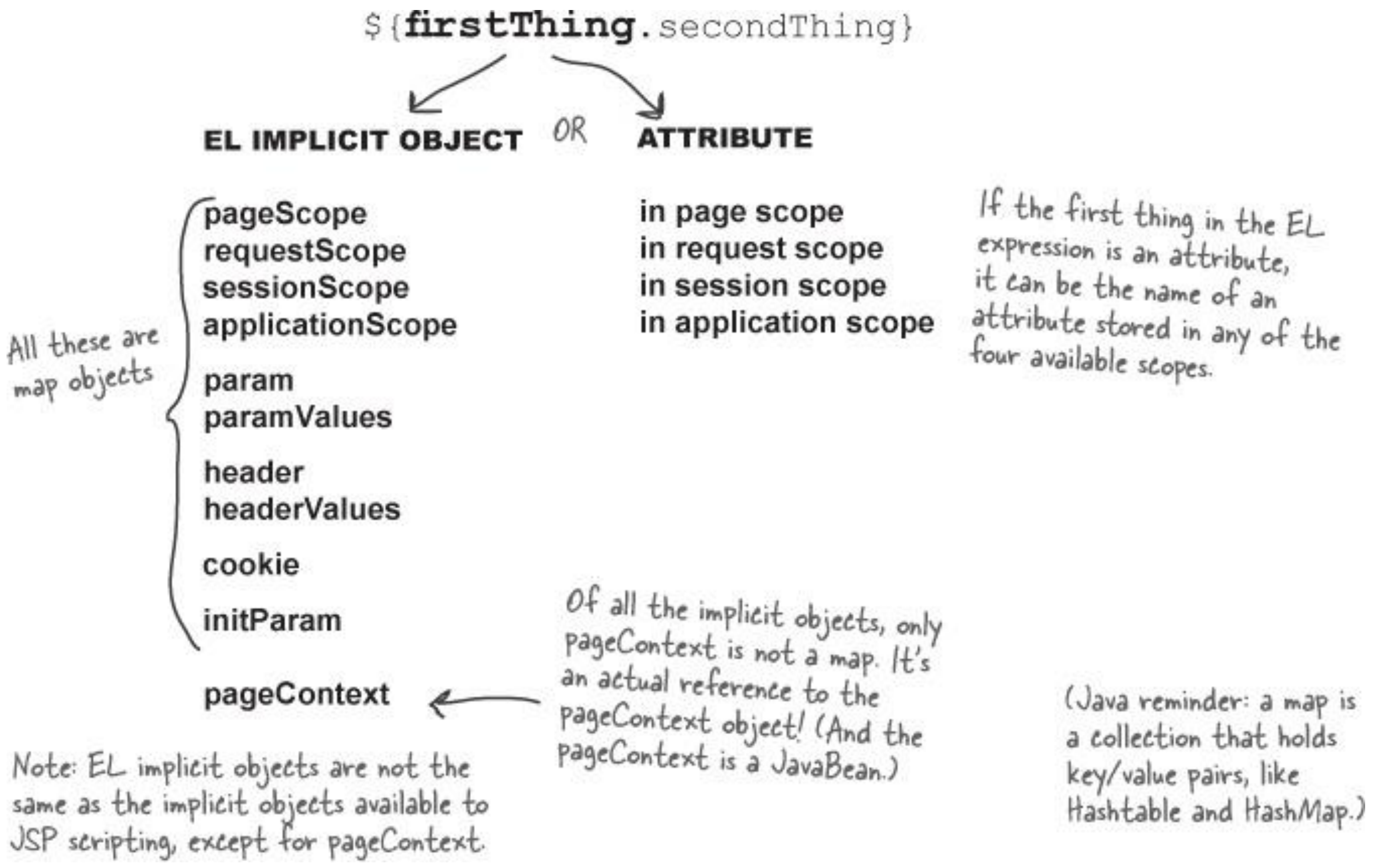
This works perfectly... but we had to use scripting

```
<html><body>
<%=  ((foo.Person) request.getAttribute("person")).getDog().getName() %>
</body></html>
```

- Using EL

```
<html><body>
Dog's name is: ${person.dog.name}
</body></html>
```

JSP Expression Language (EL)



`${firstThing.secondThing}`

EL IMPLICIT OBJECT OR **ATTRIBUTE**

EL IMPLICIT OBJECT

All these are map objects

- pageScope
- requestScope
- sessionScope
- applicationScope
- param
- paramValues
- header
- headerValues
- cookie
- initParam

- pageContext

ATTRIBUTE

- in page scope
- in request scope
- in session scope
- in application scope

If the first thing in the EL expression is an attribute, it can be the name of an attribute stored in any of the four available scopes.

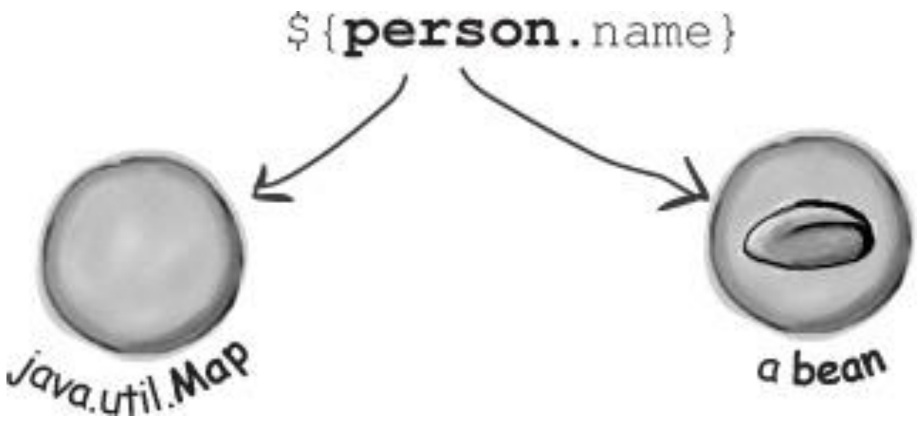
Of all the implicit objects, only pageContext is not a map. It's an actual reference to the pageContext object! (And the pageContext is a JavaBean.)

Note: EL implicit objects are not the same as the implicit objects available to JSP scripting, except for pageContext.

(Java reminder: a map is a collection that holds key/value pairs, like Hashtable and HashMap.)

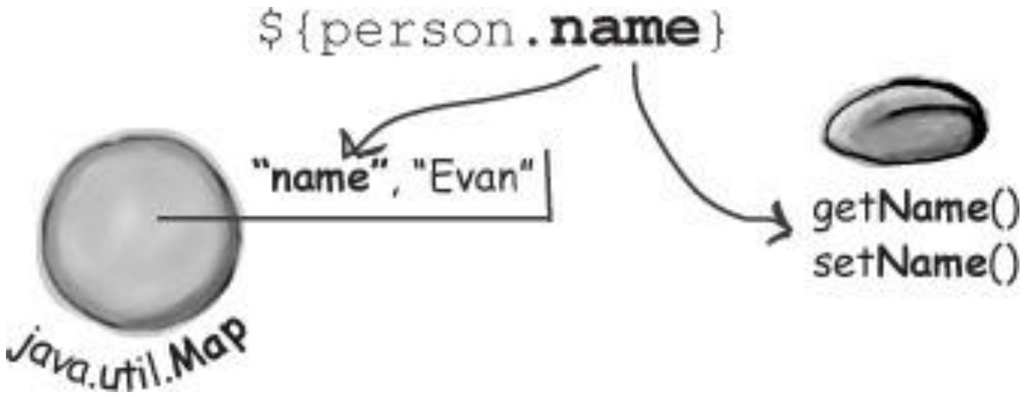
JSP Expression Language (EL)

- If the expression has a variable followed by a dot, the left-hand variable MUST be a Map or a bean.



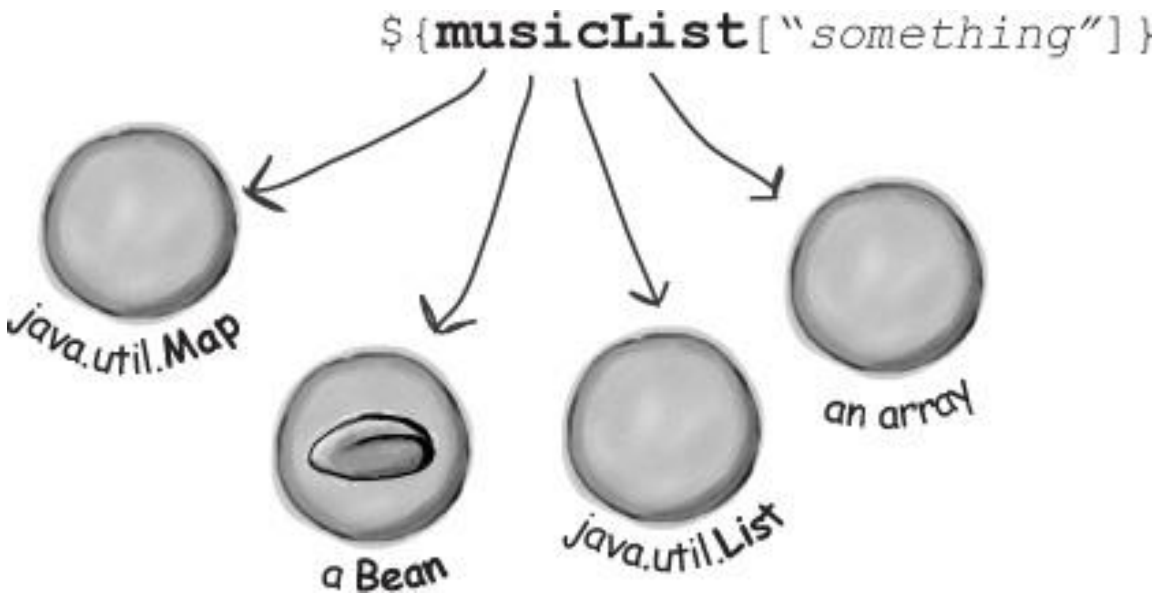
JSP Expression Language (EL)

- The thing to the right of the dot MUST be a Map key or a bean property.



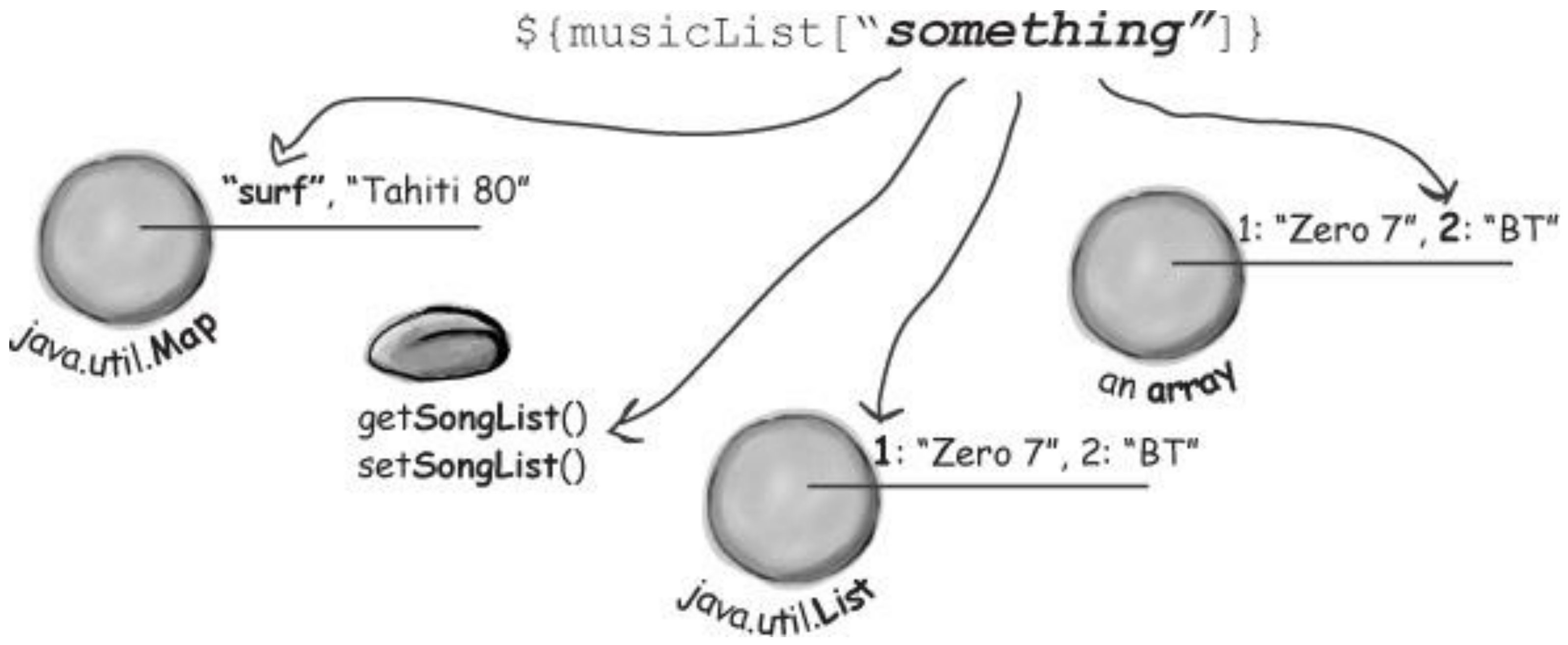
The [] gives you more options...

- If the expression has a variable followed by a bracket [ ], the left-hand variable can be a Map, a bean, a List, or an array.



The [] gives you more options...

- If the thing inside the brackets is a String literal (i.e., in quotes), it can be a Map key or a bean property, or an index into a List or array.



JSTL

Overview

- JSTL stands for JSP standard tag Library which is a collection of very useful core tags and functions.

- These tags and functions will help you write JSP code efficiently.

JSTL Core Tags

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

- Some useful tags
  - <c:out> tag
  - <c:set> tag
  - <c: if> tag
  - <c:forEach> tag

JSTL <c:out> Core Tag

- index.jsp

```
<form action="process.jsp" method="post">
      FirstName:<input type="text" name="fname"/><br/>
      LastName:<input type="text" name="lname"/><br/>
      <input type="submit" value="submit"/>
</form>
```

- process.jsp

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

  First Name:<c:out value="${param.fname}"></c:out><br/>
  Last Name:<c:out value="${param.lname}"></c:out>
```

<c:set> core JSTL tag

- Index.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="name" scope="application" value="Chaitanya Pratap Singh"/>
<a href="display.jsp">Display</a>
```

- display.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:out value="${name}"/>
```

JSTL <c:if> Core Tag

```
<c:set var="age" value="26"/>
<c:if test="${age >= 18}">
        <c:out value="You are eligible for voting!"/>
</c:if>

 <c:if test="${age < 18}">
         <c:out value="You are not eligible for voting!"/>
 </c:if>
```

JSTL <c:forEach> Tag

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Example c:forEach tag in JSTL</title>
</head>
<body>
        <c:forEach var="counter" begin="1" end="10">
                <c:out value="${counter}"/>
        </c:forEach>
</body>
</html>
```

JSTL Functions

- <%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

- Some useful functions
  - fn:length()
  - fn:startsWith()
  - fn:endsWith()
  - fn:substring()
  - fn:toUpperCase()
  - fn:toLowerCase()

Example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
<head>
<title>fn:trim() example in JSTL</title>
</head>
<body>
        <c:set var="mymsg" value=" This is the test String       "/>
        ${fn:trim(mymsg)}
</body>
</html>
```

Example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
<head>
<title>fn:startsWith example</title>
</head>
<body>
        <c:set var="mymsg" value="Example of JSTL function"/>

        The string starts with "Example": ${fn:startsWith(mymsg, 'Example')}
        <br>The string starts with "JSTL": ${fn:startsWith(mymsg, 'JSTL')}
</body>
</html>
```

Points to Remember

Q&A

Thank You

Client Logo

Revision History

Date	Version	Description	Updated by	Reviewed and Approved By
12/13/2015	1.0	Initial Document	Kien Tran	

CSC

BUSINESS SOLUTIONS
TECHNOLOGY
OUTSOURCING