

Lecture 15

Simplifying access to Java Code: The JSP 2 Expression Language

Lecture Agenda

- 1 ➤ Motivating use of the Expression Language (EL)
- 2 ➤ Understanding the Expression Language (EL) basic syntax
- 3 ➤ Understanding the relationship of EL to MVC architecture
- 4 ➤ Referencing scoped variables
- 5 ➤ Accessing bean properties, Array/List elements, and Map entries.
- 6 ➤ Using Expression Language operators.
- 7 ➤ Evaluating expressions conditionally.

EL Motivation: Simplifying MVC Output Pages

Servlets and JSP

Possibilities for Handling a Single HTTP Request

1. **Servlet Only:** Works well when ...

- Output is a binary type (ex. an image)
- There is no output (ex. you doing forwarding or redirection).
- Format/layout of page is highly variable (ex. portal)

2. **JSP only:** Works well when ...

- Output is mostly character data (ex. HTML)
- Format/layout is mostly fixed

3. **Combination (MVC Architecture):** Needed when ...

- A single request will result in multiple substantially different-looking output results.
- You have a large development team with different team members doing Web development and the Business Logic tasks.
- You perform complicated data processing, but have a relatively fixed layout.

Steps to Implement MVC with RequestDispatcher

1. Identify and define bean(s) to represent result data
 - ordinary Java classes with at least one get (accessor) method
2. Use servlet to handle requests
 - Servlet reads request parameters, checks for **missing and malformed data**, calls necessary business logic etc...
3. Obtain bean instances
 - The servlet invokes business logic (application-specific code) or data-access code to obtain the results.
4. Store the bean in the **Request**, **Session**, or **ServletContext**.
 - The servlet calls setAttribute on the request, session, or servlet context objects to store a reference to the bean(s) that represent the results of the request.

Steps to Implement MVC with RequestDispatcher ...

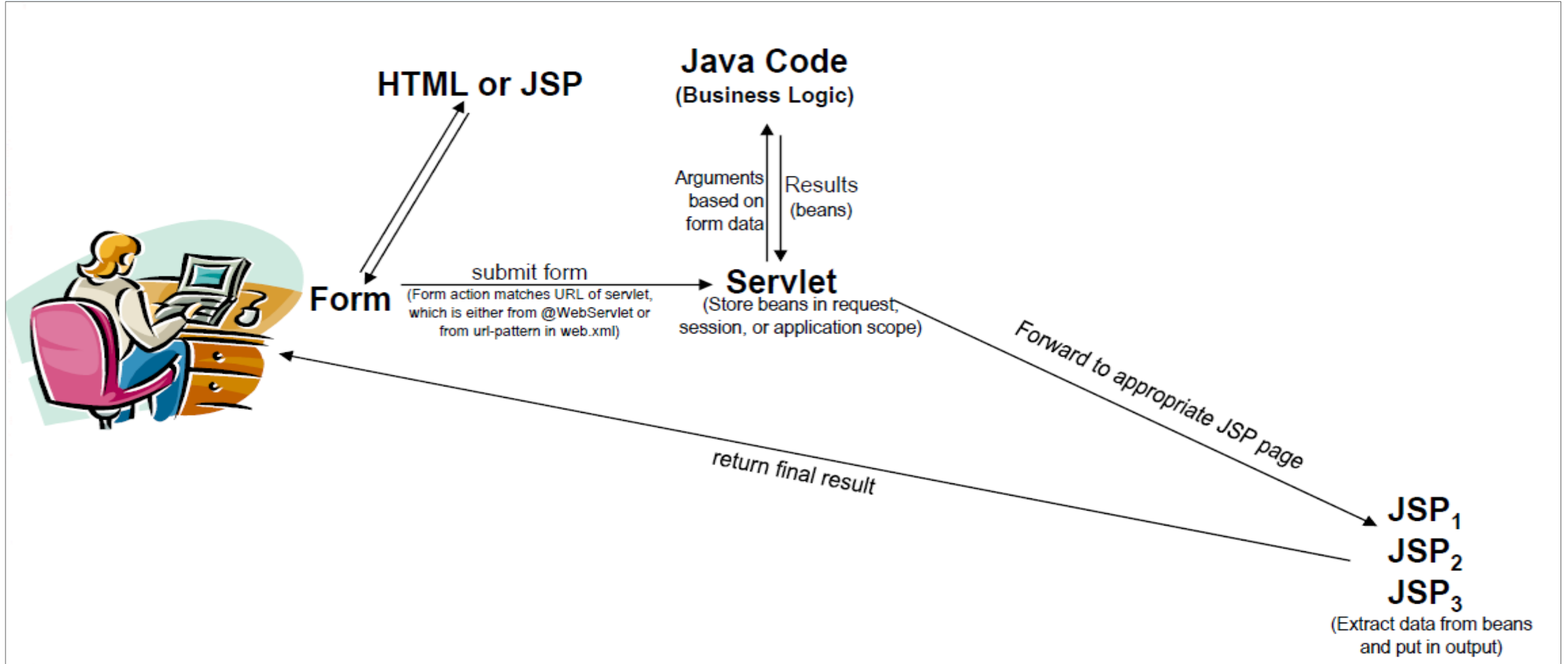
5. Forward the request to a JSP page

- The servlet determines which JSP page is appropriate to the situation and uses the forward method of **RequestDispatcher** to transfer control to that jsp page.

6. Extract the data from the beans

- The JSP page accesses beans with the `jsp:useBean` and a scope matching the location of step 4. The page then uses `jsp:getProperty` to output the bean properties.
- The JSP page does not create or modify a bean, it merely extracts and displays data that the servlet created.

MVC Flow of Control



Drawback of MVC

- **Main drawback is the final step: Presenting the results in the JSP page.**
 - `jsp:useBean` and the `jsp:getProperty`
 - Clumsy and verbose
 - `<jsp:useBean id="sampleBean" class="com.java.snippet.SampleBean" scope="session">`
 - `<jsp:getProperty name="sampleBean" property="param1" />`
 - **JSP scripting elements `<% %>`**
 - Result in hard-to-maintain code
 - Defeats the whole purpose behind MVC
- **Goal**
 - More concise and readable syntax.
 - More accessible to Web developers
 - Ability to access sub-properties
 - Ability to access collections.

Jab

Main Point of Expression Language

■ Bean

- `public String getFirstName(...)`

■ Servlet

- `Customer someCustomer = lookupService.findCustomer(...);`
- `request.setAttribute("customer", someCustomer);`
- Use `RequestDispatcher.forward()` to go to JSP page

■ JSP

- `<h1>First Name is ${customer.firstName}</h1>`
- *(FYI: If this is all you ever know about EL, you're still in pretty good shape).*

Main Point of Expression Language ...

Upgrading from JSP 1.2 to JSP 2.0+

- **When in JSP 2.x-compliant server with current web.xml version, change:**

```
<jsp:useBean id="someName"  
             type="somePackage.someClass"  
             scope="request, session, or application"/>  
  
<jsp:getProperty name="someName"  
                property="someProperty"/>
```

- **To:**

```
${someName.someProperty}
```

- **Bean, servlet, business logic**
 - Remain exactly the same as before

Advantages of the Expression Language

- **Concise access to stored objects.**

- To output a “**scoped variable**” (object stored with `setAttribute` in the `PageContext`, `HttpServletRequest`, `HttpSession` or `ServletContext`) for example named **saleItem**, you use **`${saleItem}`**
- Shorthand notation for bean properties
 - To output the **companyName** property (ex: result of the **`getCompanyName()`** method) of a scoped variable named **company**, you use **`${company.companyName}`**. To access the **firstName** property of the **president** property of a scoped variable named **company**, you use **`${company.president.firstName}`**.

- **Simple**

- To access an element of **Array**, **List**, or **Map**, you use **`${variable[indexOrKey]}`**. Provided that the index or key is in a form that is legal for Java variable names, the dot notation for beans is interchangeable with the bracket notation for collections.

Advantages of the Expression Language Continued ...

- **Clear access to request parameters, cookies and other request data**
 - To access the standard types of request data, you can use one of the several **predefined implicit objects**.
- **A small but useful set of simple operators.**
 - To manipulate objects within EL expressions, you can use any of the several **arithmetic, relational, logical or empty-testing operators**.
- **Conditional output**
 - To choose among output options, you do not have to resort to using Java scripting elements. Instead, you can use **`${test ? option1: option2}`** (think Java ternary operator `? :`)
- **Automatic type conversion**
 - The expression language removes the need for most typecasts and for much of the code that parses strings as numbers.
- **Empty values instead of error messages**
 - In most cases, missing values or **`NullPointerExceptions`** result in empty strings, not thrown exceptions

Setup

Activating the Expression Language

- Available only in servers that support JSP 2.0 or higher (servlets 2.4+)
 - Ex: Tomcat 5 or greater
 - Ex: Tomcat 8 – **Java Servlet 3.1** – **JSP 2.3**
- You must use the JSP 2.x web.xml file

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee web-app_2_4.xsd" version="3.1">
```

Web.xml extract
with Servlet 3.1

Summary

- Use MVC (Model 2) approach when:
 - One submission will result in more than one basic look
 - Several pages have substantial common processing
 - Your application is moderately complex
- Approach
 - A servlet answers the original incoming HTTP request
 - Servlet calls business logic and stores the results in beans
 - Beans stored in `HttpServletRequest`, `HttpSession`, or `ServletContext`
 - Servlet invokes JSP page via `RequestDispatcher.forward()`
 - JSP page reads data from beans
 - Most modern servers (JSP 2.0 +): `${beanName.propertyName}`
 - Older Servers (JSP 1.2): `jsp:useBean` with appropriate scope (request, session, application) plus `jsp:getProperty`

Invoking the Expression Language

- Basic form: `${expression}`

- These EL elements can appear in ordinary text or in JSP tag attributes. For example:

``

`Name: ${expression1}`

`Address: ${expression2}`

``

`<jsp:include page="${expression3}" />`

- The EL in tag attributes

- You can use multiple expressions (possibly intermixed with static text) and the results are coerced to strings and concatenated. For example:
 - `<jsp:include page="${expr1}blah${expr2}" />`

Preventing Expression Language Evaluation

- Deactivating the EL in an entire Web application
 - Use a web.xml that refers to java servlets 2.3 (JSP 1.2) or earlier (EL is not supported)
- Deactivating the expression language in multiple pages
 - Use the `jsp-property-group` web.xml element

```
<jsp-config>
<jsp-property-group>
<url-pattern>/test.jsp</url-pattern>
<scripting-invalid>true</scripting-invalid>
</jsp-property-group>
</jsp-config>
```

disable by
page

```
<jsp-config>
<jsp-property-group>
<url-pattern>/noscriptlets/</url-pattern>
<scripting-invalid>true</scripting-invalid>
</jsp-property-group>
</jsp-config>
```

disable by
url-pattern

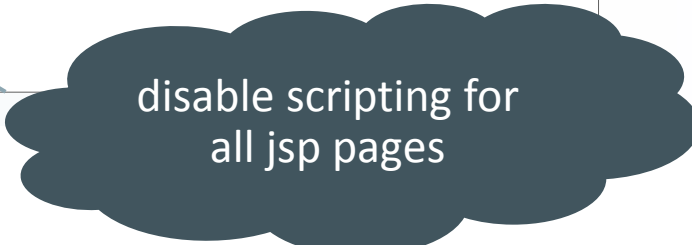
Preventing Expression Language Evaluation Continued ..

- Deactivating the expression language in individual pages
 - Use `<%@page isELIgnored="true" %>`
- Deactivating individual EL statements
 - In JSP 2.0+ pages that contain **both** EL statements and literal “`${`” Strings, you can use `\$` (where `\` *is the* escape sequence) when you want “`${`” String in the output.

Preventing Use of Standard Scripting Elements

- To enforce EL-only with no scripting, use scripting-invalid in web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi=
    "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://java.sun.com/xml/ns/j2ee web-app_2_4.xsd"
  version="2.4">
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <scripting-invalid>true</scripting-invalid>
  </jsp-property-group>
</web-app>
```



disable scripting for
all jsp pages

Downsides to Preventing Use of Scripting Elements

- Harder debugging

- `<% System.out.println(" ... "); %>`

- No redirects

- `<% response.sendRedirect("welcome.jsp")%>`

- Some techniques are hard to do with MVC

```
<%  
    If(outputShouldBeExcel()) {  
        response.setContentType("application/vnd.ms-excel")  
    }  
%>
```

- Just because scripting is usually bad does not mean it is always bad

EL Uses: Scoped variables, Bean properties,
collections

Accessing Scoped Variables

- **`${varName}`**

- Searches the **PageContext**, the **HttpServletRequest**, the **HttpSession** and the **ServletContext**, in that order and outputs the object with the specified attribute name. Please note **PageContext** does not apply in MVC.

- **Equivalent forms**

- I. **`${name}`**
- II. **`<%= PageContext.findAttribute("name") %>`**
- III. **`<jsp:useBean id="name" type="somePackage.SomeClass" scope="..." >`
`<%= name %>`**

Example: Accessing Scoped Variables

```
@WebServlet("/scoped-vars")
public class ScopedVars extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        request.setAttribute("attribute1", "First Value");
        HttpSession session = request.getSession();
        session.setAttribute("attribute2", "Second Value");
        ServletContext application = getServletContext();
        application.setAttribute("attribute3",
                                new java.util.Date());
        request.setAttribute("repeated", "Request");
        session.setAttribute("repeated", "Session");
        application.setAttribute("repeated", "ServletContext");
        RequestDispatcher dispatcher =
            request.getRequestDispatcher
                ("/WEB-INF/results/scoped-vars.jsp");
        dispatcher.forward(request, response);
    }
}
```

Example: Accessing Scoped Variables Continued ...

```
<!DOCTYPE ...>
```

```
...
```

```
<TABLE BORDER=5 ALIGN="CENTER">
```

```
  <TR><TH CLASS="TITLE">
```

```
    Accessing Scoped Variables
```

```
</TABLE>
```

```
<P>
```

```
<UL>
```

```
  <LI><B>attribute1:</B> ${attribute1}
```

```
  <LI><B>attribute2:</B> ${attribute2}
```

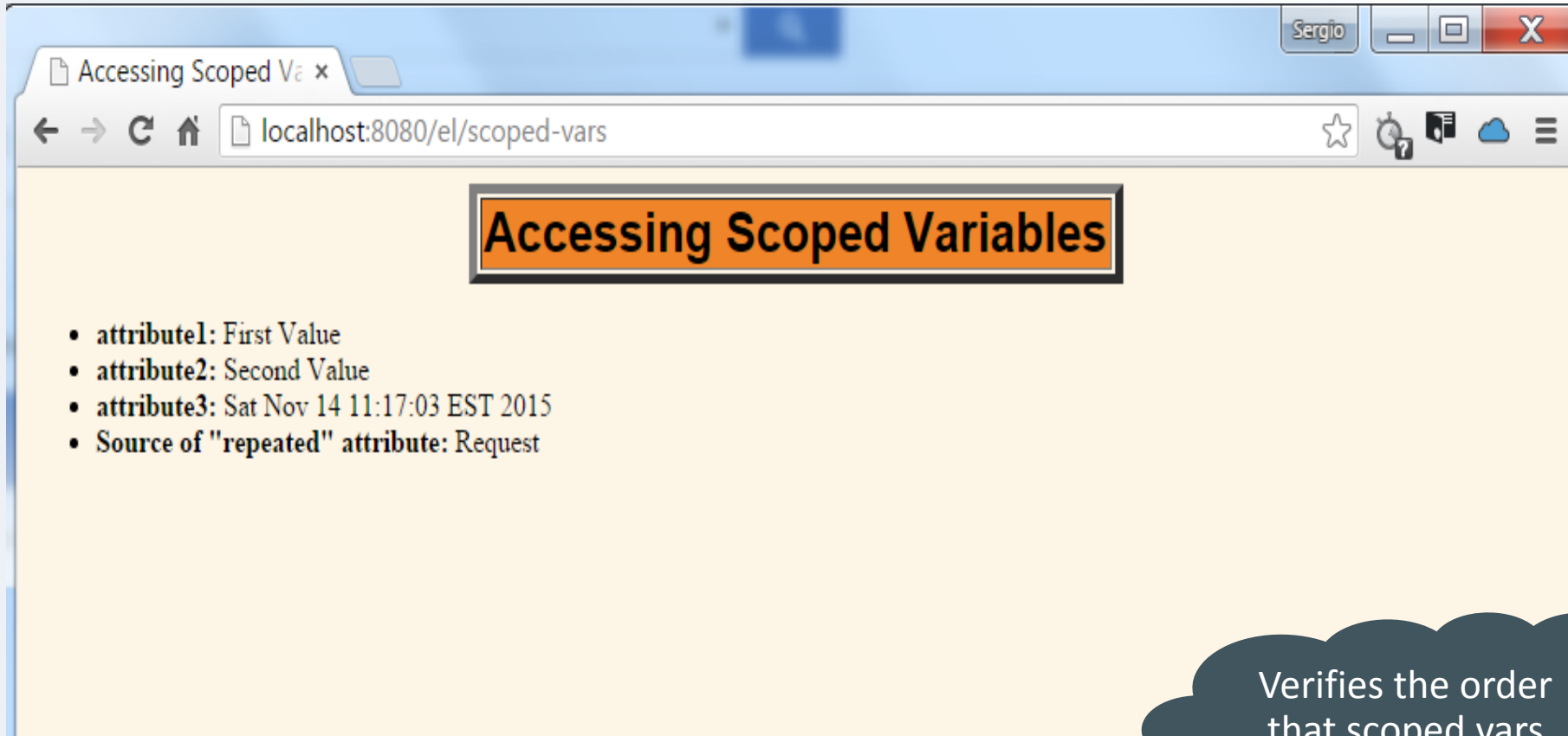
```
  <LI><B>attribute3:</B> ${attribute3}
```

```
  <LI><B>Source of "repeated" attribute:</B>  
    ${repeated}
```

```
</UL>
```

```
</BODY></HTML>
```


Example: Accessing Scoped Variables Result



Verifies the order
that scoped vars
are accessed.

Accessing Bean Properties

- **`${varName.propertyName}`**

- Means to find scoped variable of given name and output the specified bean property
 - Remember from MVC lecture that bean property corresponds to getter method name, not instance var.

- **Equivalent forms**

- `${customer.firstName}`
- ```
<%@ page import="coreservlets.NameBean" %>
<%
NameBean person =
 (NameBean)pageContext.findAttribute("customer");
%>
<%= person.getFirstName() %>
```



Entire Scriptlet

# Example: Accessing Bean Properties

```
@WebServlet("/bean-properties")
public class BeanProperties extends HttpServlet {
 public void doGet(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException {
 Name name = new Name("Marty", "Hall");
 Company company =
 new Company("coreservlets.com",
 "Customized Java EE and Ajax Training");
 Employee employee =
 new Employee(name, company);
 request.setAttribute("employee", employee);
 RequestDispatcher dispatcher =
 request.getRequestDispatcher
 ("/WEB-INF/results/bean-properties.jsp");
 dispatcher.forward(request, response);
 }
}
```

# Example: Accessing Bean Properties Continued ...

```
public class Employee {
 private Name name;
 private Company company;

 public Employee(Name name, Company company) {
 setName(name);
 setCompany(company);
 }

 public Name getName() { return(name); }

 public void setName(Name name) {
 this.name = name;
 }

 public CompanyBean getCompany() { return(company); }

 public void setCompany(Company company) {
 this.company = company;
 }
}
```

# Example: Accessing Bean Properties Continued ...

```
public class Name {
 private String firstName;
 private String lastName;

 public Name(String firstName, String lastName) {
 setFirstName(firstName);
 setLastName(lastName);
 }

 public String getFirstName() {
 return (firstName);
 }
 public void setFirstName(String firstName) {
 this.firstName = firstName;
 }
 public String getLastName() {
 return (lastName);
 }
 public void setLastName(String lastName) {
 this.lastName = lastName;
 }
}
```

# Example: Accessing Bean Properties Continued ...

```
public class Company {
 private String companyName;
 private String business;

 public Company(String companyName, String business) {
 setCompanyName(companyName);
 setBusiness(business);
 }

 public String getCompanyName() { return(companyName); }

 public void setCompanyName(String companyName) {
 this.companyName = companyName;
 }

 public String getBusiness() { return(business); }

 public void setBusiness(String business) {
 this.business = business;
 }
}
```

# Example: Accessing Bean Properties Continued ...

```
<!DOCTYPE ...>
```

```
...
```

```

```

```
First Name:
```

```
 ${employee.name.firstName}
```

```
Last Name:
```

```
 ${employee.name.lastName}
```

```
Company Name:
```

```
 ${employee.company.companyName}
```


```
Company Business:
```

```
 ${employee.company.business}
```

```

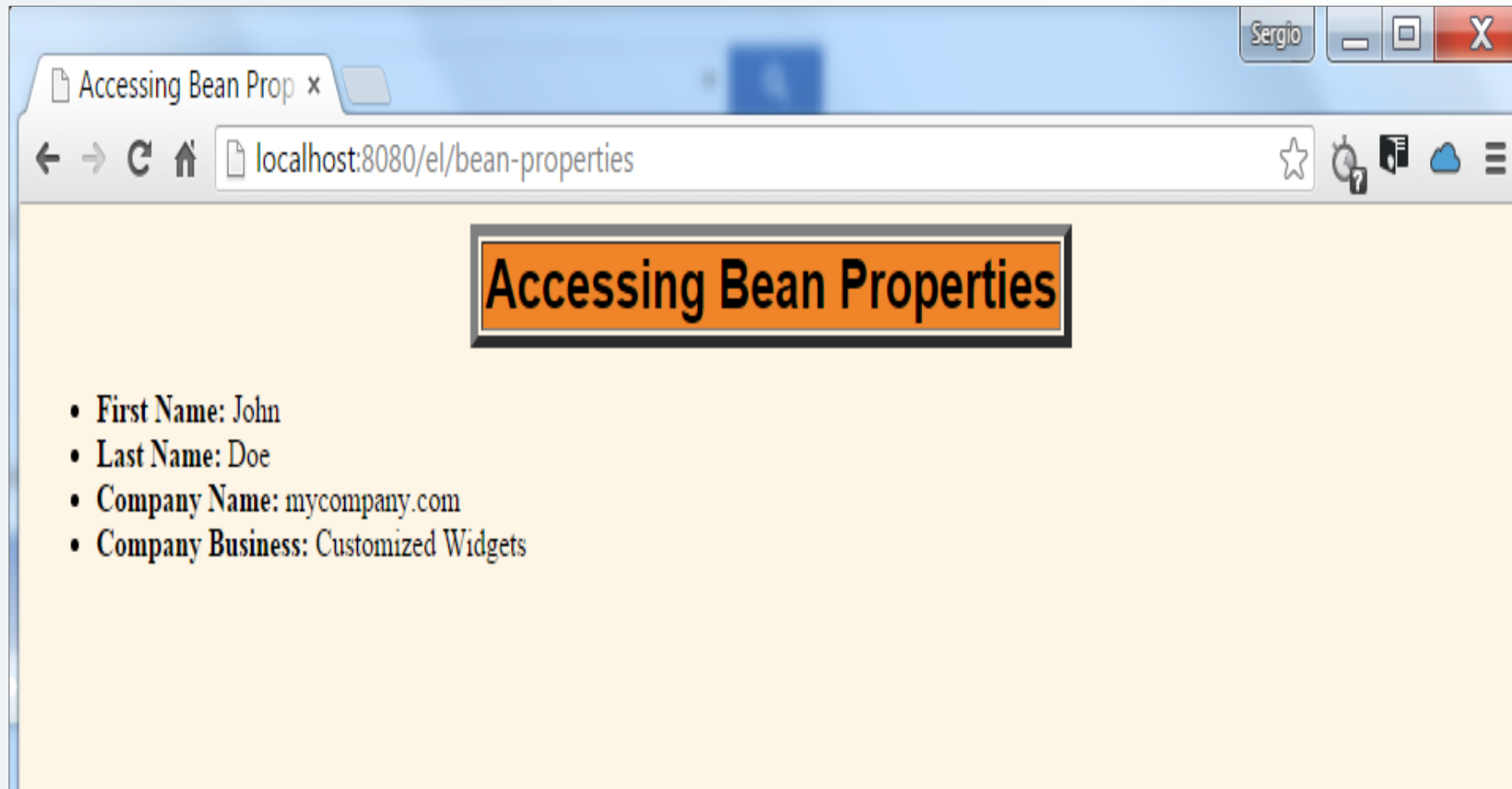
```

```
</BODY></HTML>
```



Notice how we  
access the objects  
sub properties  
using EL

# Example: Accessing Bean Properties Continued ...





# Equivalence of Dot and Array Notations

- **Equivalent forms**

- `${name.property}`
- `${name["property"]}`

- **Reasons for using array notation**

- To access arrays, lists, and other collections
  - See upcoming slides
- To calculate the property name at request time.
  - `{name1[name2]}` (no quotes around name2)
- To use names that are illegal as Java variable names
  - `{foo["bar-baz"]}`
  - `{foo["bar.baz"]}`

# Accessing Collections

- **`${attributeName[entryName]}`**

- **Works for**

- Array. Equivalent to
  - `theArray[index]`
- List. Equivalent to
  - `theList.get(index)`
- Map. Equivalent to
  - `theMap.get(keyName)`



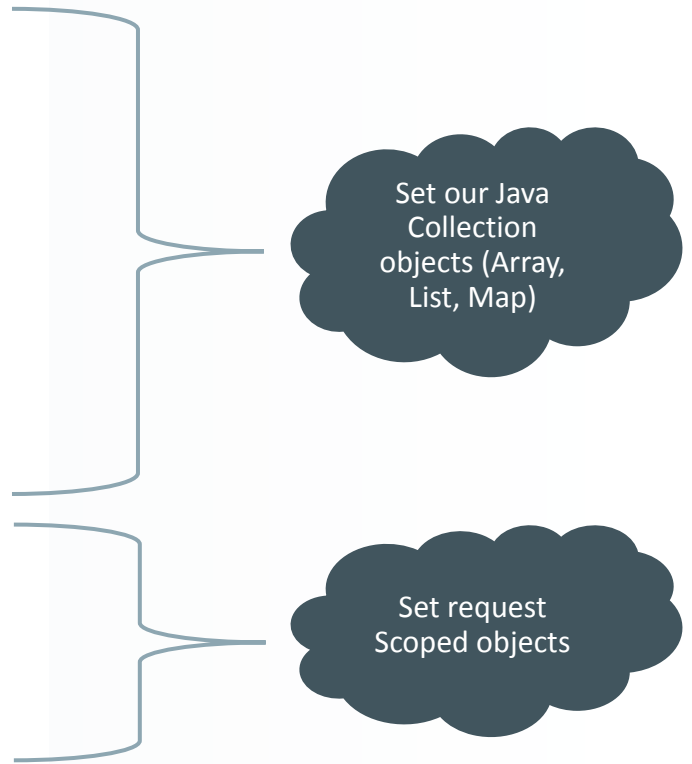
Java Code

- **Equivalent forms (for HashMap)**

- `${stateCapitals["maryland"]}`
- `${stateCapitals.maryland}`
- But the following is illegal since 2 is not a legal var name
  - `${listVar.2}`

# Example: Accessing Collections

```
public class Collections extends HttpServlet {
 public void doGet(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException {
 String[] firstNames = { "Bill", "Scott", "Larry" };
 List<String> lastNames = new ArrayList<String>();
 lastNames.add("Ellison");
 lastNames.add("Gates");
 lastNames.add("McNealy");
 Map<String,String> companyNames =
 new HashMap<String,String>();
 companyNames.put("Ellison", "Sun");
 companyNames.put("Gates", "Oracle");
 companyNames.put("McNealy", "Microsoft");
 request.setAttribute("first", firstNames);
 request.setAttribute("last", lastNames);
 request.setAttribute("company", companyNames);
 RequestDispatcher dispatcher =
 request.getRequestDispatcher
 ("/WEB-INF/results/collections.jsp");
 dispatcher.forward(request, response);
 }
}
```



Set our Java  
Collection  
objects (Array,  
List, Map)

Set request  
Scoped objects

# Example: Accessing Collections (Continued)

```
<!DOCTYPE ...>
```

```
...
```

```
<BODY>
```

```
<TABLE BORDER=5 ALIGN="CENTER">
```

```
<TR><TH CLASS="TITLE">
```

```
Accessing Collections
```

```
</TABLE>
```

```
<P>
```

```

```

```
${first[0]} ${last[0]} (${company["Ellison"]})
```

```
${first[1]} ${last[1]} (${company["Gates"]})
```

```
${first[2]} ${last[2]} (${company["McNealy"]})
```

```

```

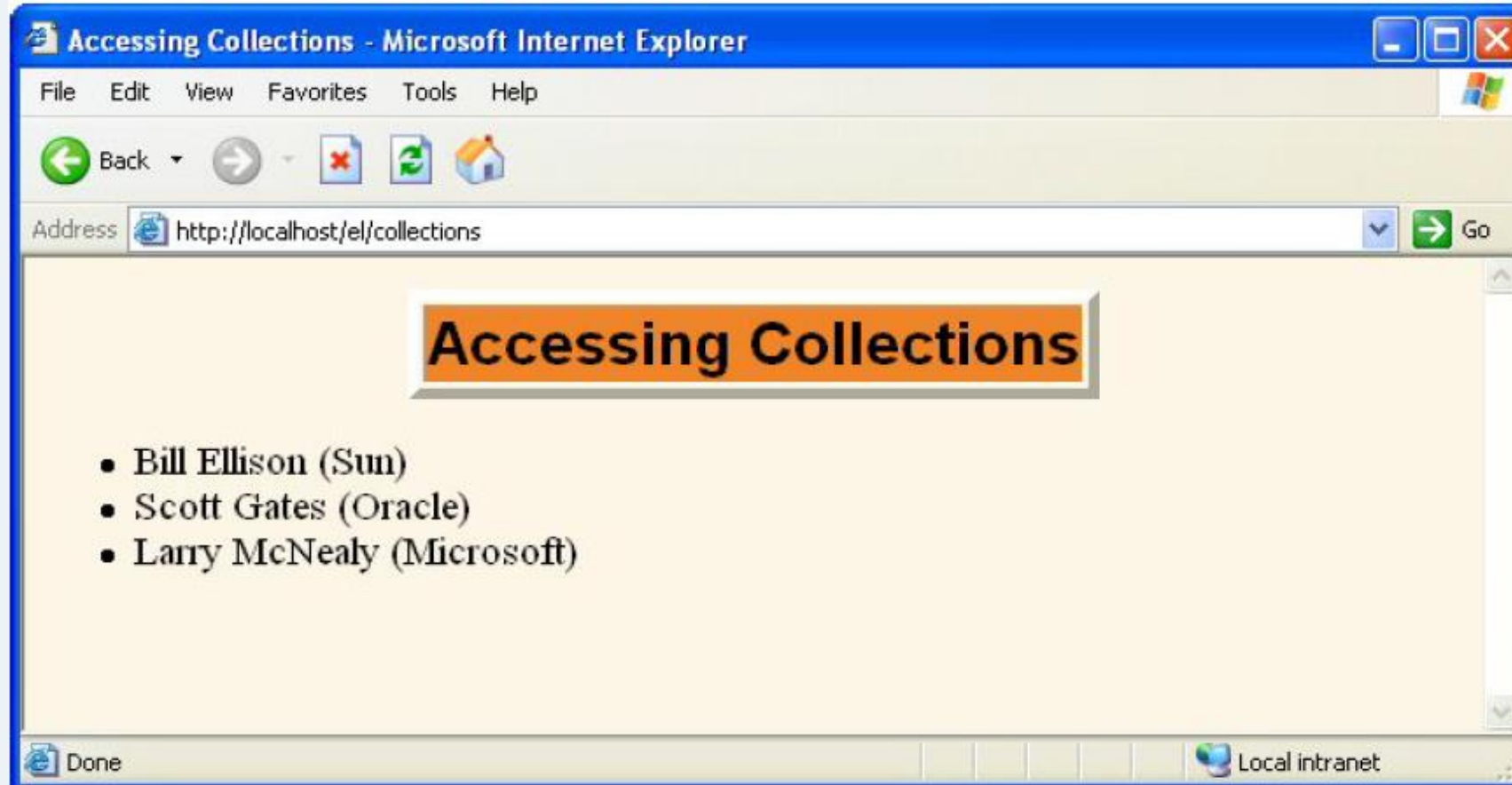
```
</BODY></HTML>
```

Accessing Array  
using indices

Accessing Map  
using key

Accessing  
ArrayList  
using indices

# Example: Accessing Collections (Result)



# Implicit Objects and Operators

# Referencing Implicit Objects

## Predefined Variable Names

JSP EL Implicit Objects	Type	Description
pageScope	Map	A map that contains the attributes set with page scope.
requestScope	Map	Used to get the attribute value with request scope.
sessionScope	Map	Used to get the attribute value with session scope.
applicationScope	Map	Used to get the attributes value from application scope.
param	Map	Used to get the request parameter value, returns a single value
paramValues	Map	Used to get the request param values in an array, useful when request parameter contain multiple values.
header	Map	Used to get request header information.
headerValues	Map	Used to get header values in an array.
cookie	Map	Used to get the cookie value in the JSP
initParam	Map	Used to get the context init params, we can't use it for servlet init params
pageContext	pageContext	Same as JSP implicit pageContext object, used to get the request, session references etc. example usage is getting request HTTP Method name.

# Referencing Implicit Objects Continued ...

## Predefined Variable Names

- **pageContext.** The PageContext object.
  - E.g. `${pageContext.session.id}`
- **param and paramValues.** Request params
  - E.g. `${param.custID}` or `${param['custID']}`
- **header and headerValues.** Request headers
  - E.g. `${header.Accept}` or `${header["Accept"]}`
  - `${header["Accept-Encoding"]}`
- **cookie.** Cookie object
  - E.g. `${cookie.userCookie.value}` or `${cookie["userCookie"].value}`
- **pageScope, requestScope, sessionScope, applicationScope**
  - For searching different scopes example: `${sessionScope['cart']}`



# Example: Implicit Object

## Predefined Variable Names

```
<!DOCTYPE ...>
```

```
...
```

```
<P>
```

```

```

```
test Request Parameter:
 ${param.test}
```

```
User-Agent Header:
 ${header["User-Agent"]}
```

```
JSESSIONID Cookie Value:
 ${cookie.JSESSIONID.value}
```

```
Server:
 ${pageContext.servletContext.serverInfo}
```

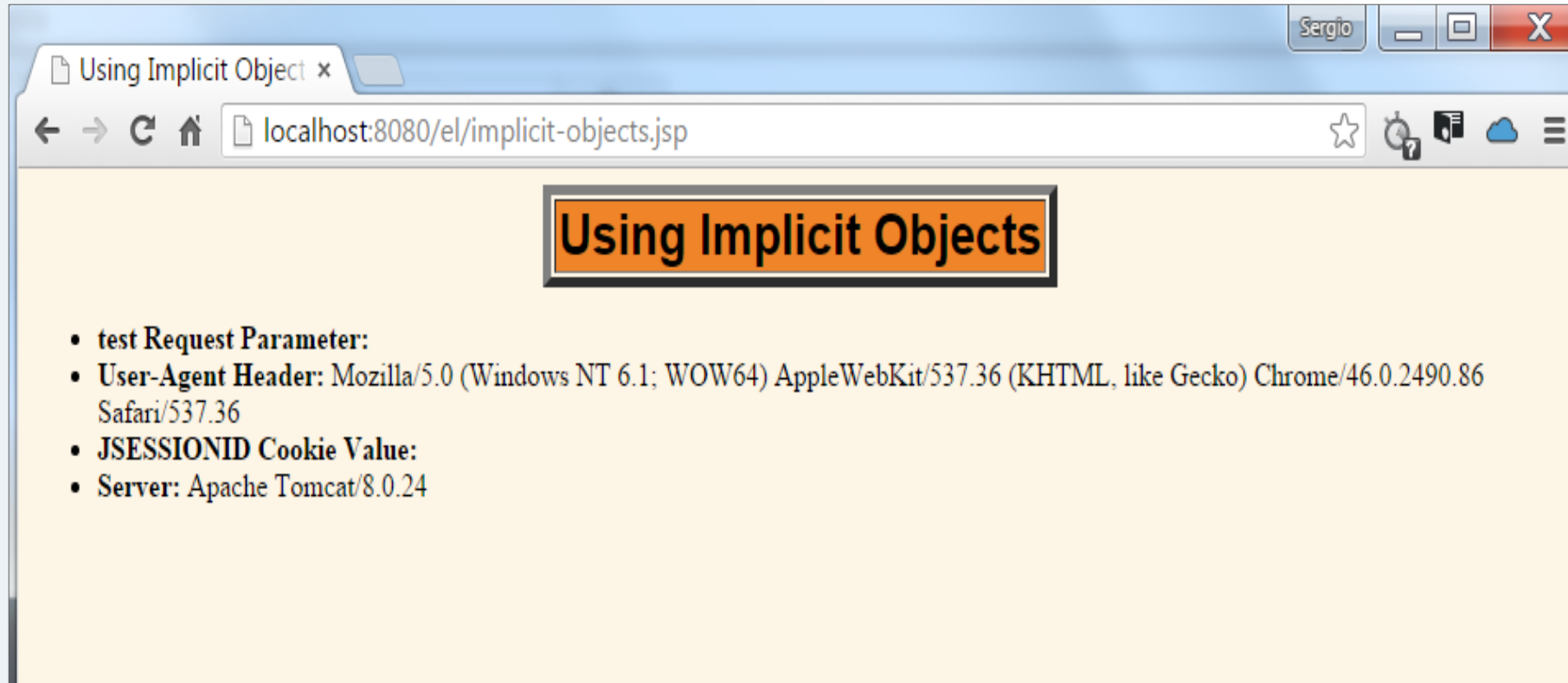
```

```

```
</BODY></HTML>
```

# Example: Implicit Object

## Result



# Expression Language Operators

# Expression Language Operators

## Summary

- **Arithmetic**

- `+ - * / div % mod`

- **Relational**

- `== eq != ne < lt > gt <= le >= ge`

- **Logical**

- `&& and || or ! Not`

- **Empty**

- `Empty`

- True for null, empty string, empty array, empty list, empty map. False otherwise.

# Example: Expression Language Operators

...

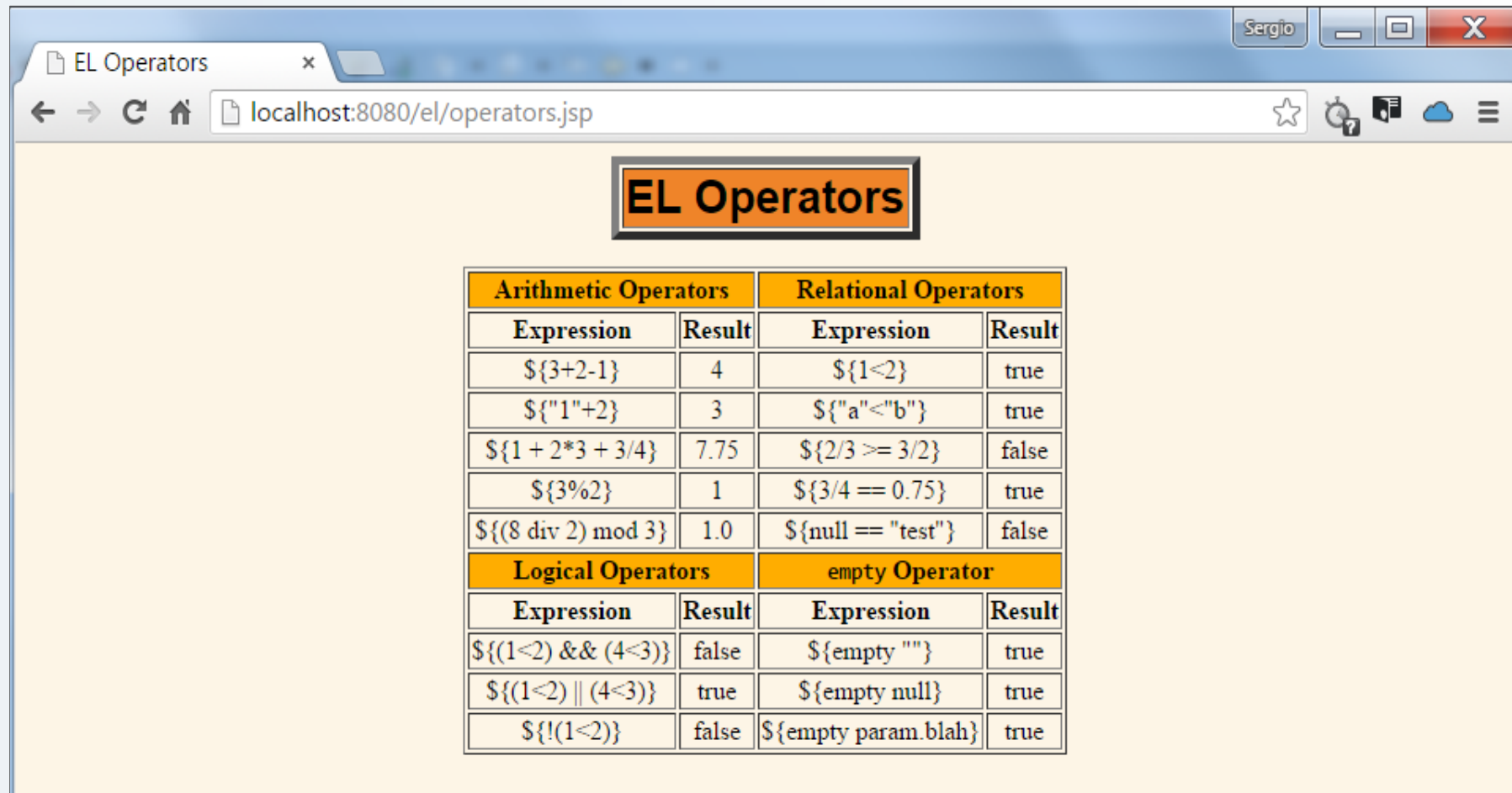
```
<TABLE BORDER=1 ALIGN="CENTER">
 <TR><TH CLASS="COLORED" COLSPAN=2>Arithmetic Operators
 <TH CLASS="COLORED" COLSPAN=2>Relational Operators
 <TR><TH>Expression<TH>Result<TH>Expression<TH>Result
 <TR ALIGN="CENTER">
 <TD>\${3+2-1}<TD>${3+2-1}
 <TD>\${1<2}<TD>${1<2}
 <TR ALIGN="CENTER">
 <TD>\${"1"+2}<TD>${"1"+2}
 <TD>\${"a"<"b"}<TD>${"a"<"b"}
 <TR ALIGN="CENTER">
 <TD>\${1 + 2*3 + 3/4}<TD>${1 + 2*3 + 3/4}
 <TD>\${2/3 >= 3/2}<TD>${2/3 >= 3/2}
 <TR ALIGN="CENTER">
 <TD>\${3%2}<TD>${3%2}
 <TD>\${3/4 == 0.75}<TD>${3/4 == 0.75}
```

...

Note escape  
character

Evaluated

# Example: Expression Language Operators



The screenshot shows a web browser window with the title 'EL Operators' and the address bar displaying 'localhost:8080/el/operators.jsp'. The page content includes a title 'EL Operators' in a stylized orange box. Below the title, there are two tables of EL operators. The first table is divided into 'Arithmetic Operators' and 'Relational Operators'. The second table is divided into 'Logical Operators' and 'empty Operator'.

Arithmetic Operators		Relational Operators	
Expression	Result	Expression	Result
<code>\${3+2-1}</code>	4	<code>\${1&lt;2}</code>	true
<code>\${"1"+2}</code>	3	<code>\${"a"&lt;"b"}</code>	true
<code>\${1 + 2*3 + 3/4}</code>	7.75	<code>\${2/3 &gt;= 3/2}</code>	false
<code>\${3%2}</code>	1	<code>\${3/4 == 0.75}</code>	true
<code>\${(8 div 2) mod 3}</code>	1.0	<code>\${null == "test"}</code>	false

Logical Operators		empty Operator	
Expression	Result	Expression	Result
<code>\${(1&lt;2) &amp;&amp; (4&lt;3)}</code>	false	<code>\${empty ""}</code>	true
<code>\${(1&lt;2)    (4&lt;3)}</code>	true	<code>\${empty null}</code>	true
<code>\${!(1&lt;2)}</code>	false	<code>\${empty param.blah}</code>	true

# Evaluating Expressions Conditionally

- **`${test ? expression 1 : expression 2}`**
  - Evaluates test and outputs either `expression1` or `expression2`
  - Similar in theory to the Java Ternary operator:  
`boolean var = (condition) ? true : false;`
- Concerns
  - JSTL is much better for conditional testing (`c:if` and `c:choose`)
  - Tempts you to put business/processing logic in JSP page
  - JSP, ideally should only be used for presentation logic.

Jab

# Example: Conditional Expression

```
@WebServlet("/conditionals")
public class Conditionals extends HttpServlet {
 public void doGet(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException {
 SalesBean apples =
 new SalesBean(150.25, -75.25, 22.25, -33.57);
 SalesBean oranges =
 new SalesBean(-220.25, -49.57, 138.25, 12.25);
 request.setAttribute("apples", apples);
 request.setAttribute("oranges", oranges);
 RequestDispatcher dispatcher =
 request.getRequestDispatcher
 ("/WEB-INF/results/conditionals.jsp");
 dispatcher.forward(request, response);
 }
}
```



## Example: Conditional Expression Continued ...

```
public class SalesBean {
 private double q1, q2, q3, q4;

 public SalesBean(double q1Sales,
 double q2Sales,
 double q3Sales,
 double q4Sales) {
 q1 = q1Sales; q2 = q2Sales;
 q3 = q3Sales; q4 = q4Sales;
 }

 public double getQ1() { return(q1); }
 public double getQ2() { return(q2); }
 public double getQ3() { return(q3); }
 public double getQ4() { return(q4); }
 public double getTotal() {
 return(q1 + q2 + q3 + q4); }
}
```

# Example: Conditional Expression Continued ...

...

```
<TABLE BORDER=1 ALIGN="CENTER">
```

```
<TR><TH>
```

```
<TH CLASS="COLORED">Apples
```

```
<TH CLASS="COLORED">Oranges
```

```
<TR><TH CLASS="COLORED">First Quarter
```

```
<TD ALIGN="RIGHT">${apples.q1}
```

```
<TD ALIGN="RIGHT">${oranges.q1}
```

```
<TR><TH CLASS="COLORED">Second Quarter
```

```
<TD ALIGN="RIGHT">${apples.q2}
```

```
<TD ALIGN="RIGHT">${oranges.q2}
```

...

```
<TR><TH CLASS="COLORED">Total
```

```
<TD ALIGN="RIGHT"
```

```
BGCOLOR="${ (apples.total < 0) ? "RED" : "WHITE" }">
```

```
${apples.total}
```

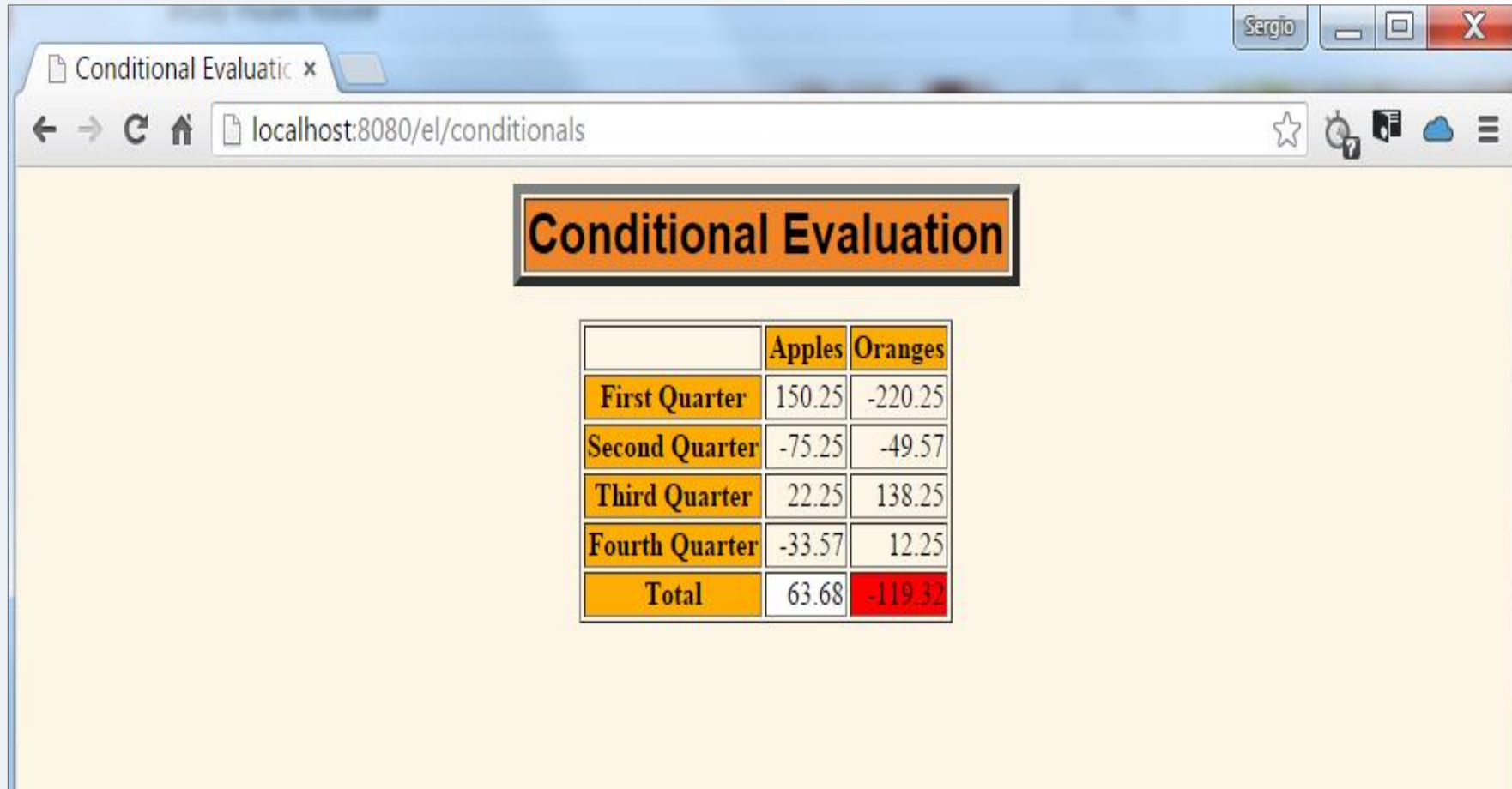
```
<TD ALIGN="RIGHT"
```

```
BGCOLOR="${ (oranges.total < 0) ? "RED" : "WHITE" }">
```

```
${oranges.total}
```

```
</TABLE>...
```

# Example: Conditional Expression (Result)



The screenshot shows a web browser window with a single tab titled 'Conditional Evaluatic'. The address bar displays 'localhost:8080/el/conditionals'. The page content features a title 'Conditional Evaluation' in a bold, black font within an orange-bordered box. Below the title is a table with three columns: an empty header cell, 'Apples', and 'Oranges'. The table contains five rows of data, with the 'Total' row highlighted in red.

	Apples	Oranges
First Quarter	150.25	-220.25
Second Quarter	-75.25	-49.57
Third Quarter	22.25	138.25
Fourth Quarter	-33.57	12.25
Total	63.68	-119.32

# Summary

- **The JSP 2 EL provides concise, easy-to-read access to:**
  - Scoped variables
  - Bean properties
  - Collection elements
  - Standard HTTP elements such as request parameters, request headers, and cookies
- **The JSP EL works best with MVC**
  - Use only output values created by separate Java code.
- **Resist use of EL for business logic**
  - Use EL operators and conditionals sparingly, if at all.

Questions?