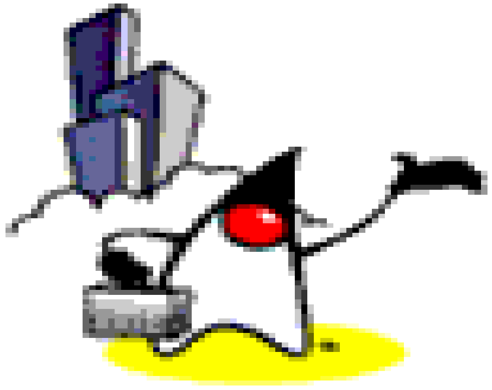
JSP Basics II

Sang Shin
www.JPassion.com
"Learn with JPassion!"

Agenda

- Dynamic contents generation techniques in JSP
- Invoking Java code using JSP scripting elements
- Including and forwarding to other JSP
- Redirecting
- Directives
- Scope objects
- Error handling



Dynamic Content Generation Techniques in JSP

Dynamic Contents Generation Techniques with JSP Technology

- Various techniques can be chosen depending on the following factors
  - Size and complexity of the project
  - Requirements on re usability of code, maintainability, degree of robustness
- Simple to incrementally complex techniques are available

Dynamic Contents Generation Techniques with JSP

a) Call Java code directly within JSP (covered in this presentation)

b) Call Java code indirectly within JSP

c) Use JavaBeans within JSP (covered in “jsp_javabean”)

d) Develop and use your own custom tags (“jsp_2.0customtags)

e) Leverage JSTL (JSP Standard Tag Library) and 3rd-party custom tags (“jsp_jstl”)

f) Follow MVC design pattern

g) Leverage proven MVC frameworks

(a) Call Java code directly

- Place all Java code in JSP page
- Suitable only for a very simple Web application because it is
  - hard to maintain
  - hard to reuse code
  - hard to understand for web page authors
- Not recommended for relatively sophisticated Web applications
  - Because it does not provide adequate separation between contents and presentation

(b) Call Java code indirectly

- Develop separate utility classes (external to JSP page)
- Insert into JSP page only the Java code needed to invoke the utility classes
- Better separation of contents generation from presentation logic than the previous method
- Better reusability and maintainability than the previous method
- Still not enough separation between contents and presentation, however

(c) Use JavaBeans

- Develop utility classes in the form of JavaBeans
- Leverage built-in JSP facility of creating JavaBeans instances, getting and setting JavaBeans properties
  - Use JSP element syntax
- Easier to use for web page authors
- Better reusability and maintainability than the previous methods

(d) Develop and Use Custom Tags

- Develop sophisticated components called custom tags
  - Custom tags are specifically designed for JSP
- More powerful than JavaBeans components
  - It provides more than just getter and setter methods
- Higher level of reusability, maintainability, robustness
- Downside: Development of custom tags are more difficult than creating JavaBeans, however (especially JSP 1.2 based ones)

(e) Use JSTL & 3rd-party Custom tags

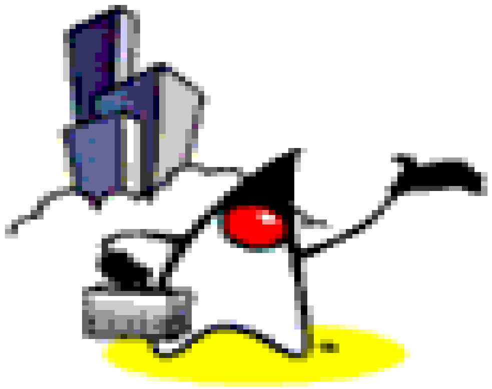
- JSTL (JSP Standard Tag Library) standardize the set of custom tags that should be available over Java EE platform at a minimum
  - As a developer or deployer, you can be assured that a standard set of custom tags are already present in Java EE compliant platform (J2EE 1.3 and after)
- There are many open source and commercial custom tags available
  - Apache Struts

(f) Design/Use MVC Design Pattern

- Follow MVC design pattern
  - Model using some model technologies
  - View using JSP
  - Controller using Servlet
- Creating and maintaining your own MVC framework is highly discouraged however

(g) Use Proven MVC Frameworks

- There are many to choose from
  - SpringMVC
  - JavaServer Faces (JSF)
  - Apache Struts 1 or Struts 2
  - Tapestry
  - Wicket



Invoking Java Code within JSP

JSP Scripting Elements

- You can insert Java code within a JSP page through JSP scripting elements
  - Minimize the usage of JSP scripting elements in your JSP pages if possible
- There are three forms of JSP scripting elements
  - Expressions: <%= Expressions %>
  - Scriptlets: <% Code %>
  - Declarations: <%! Declarations %>

Expressions

- During execution phase
  - Expression is evaluated and converted into a String
  - The String is then Inserted into the servlet's output stream directly
  - Results in something like out.println(expression)
  - Can use predefined variables (implicit objects) within the expression
- Format
  - <%= Expression %> or
  - <jsp:expression>Expression</jsp:expression>
  - Semi-colons are not allowed for expressions

Example: Expressions

- Display current time using Date class
  - Current time: <%= new java.util.Date() %>
- Display random number using Math class
  - Random number: <%= Math.random() %>
- Use implicit objects
  - Your hostname: <%= request.getRemoteHost() %>
  - Your parameter: <%= request. getParameter("yourParameter") %>
  - Server: <%= application.getServerInfo() %>
  - Session ID: <%= session.getId() %>

Scriptlets

- Used to insert arbitrary Java code into servlet's *jspService()* method
- Can do things expressions alone cannot do such as following
  - setting response headers and status codes
  - writing to a server log
  - updating database
  - executing code that contains loops, conditionals
- Can use predefined variables (implicit objects)
- Format:
  - <% Java code %> or
  - <jsp:scriptlet> Java code</jsp:scriptlet>

Example: Scriptlets

- Display query string

```
<%
String queryData = request.getQueryString();
out.println(“Attached GET data: “ + queryData);
%>
```

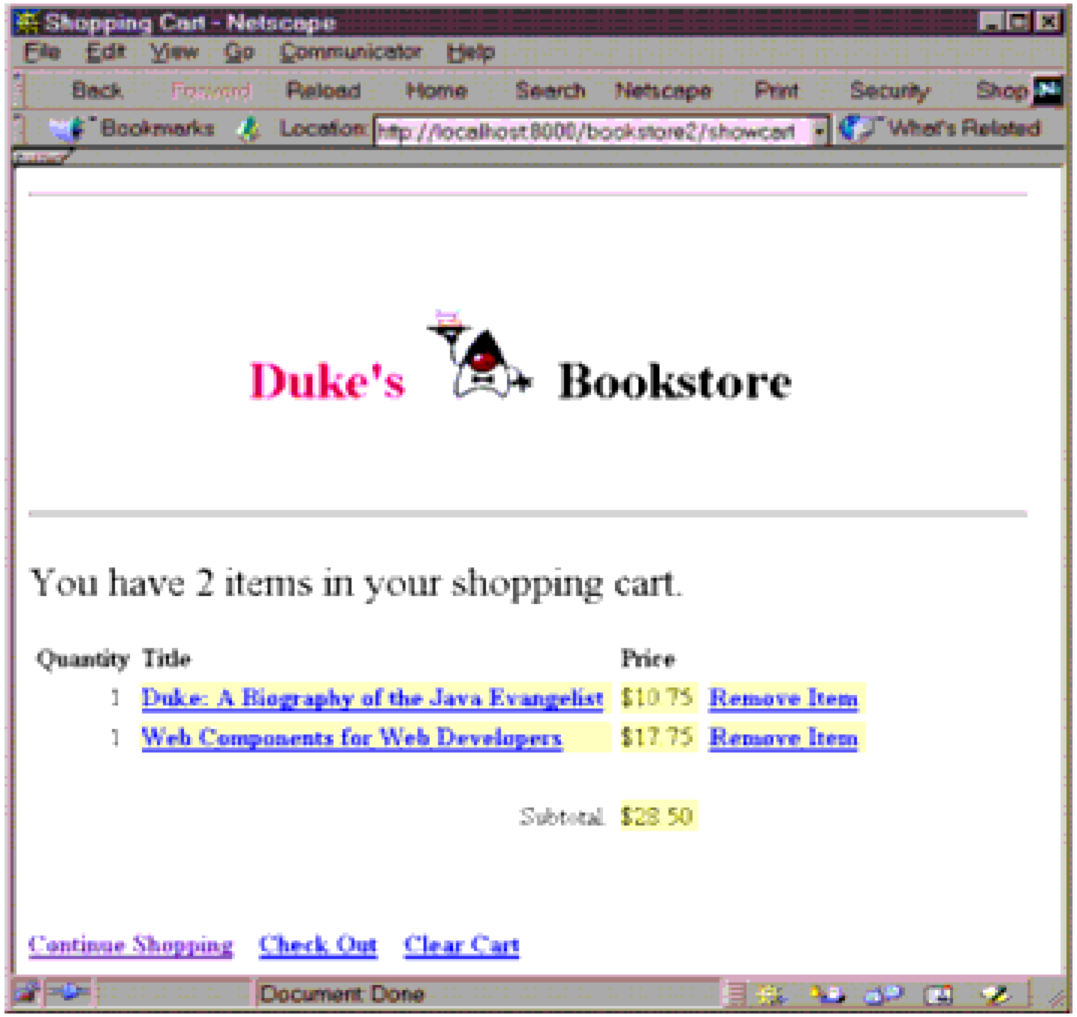
- Settng response type

```
<% response.setContentType(“text/plain”); %>
```

Example: Scriptlet with Loop

```
<%
  Iterator i = cart.getItems().iterator();
  while (i.hasNext()) {
    ShoppingCartItem item =
      (ShoppingCartItem)i.next();
    BookDetails bd = (BookDetails)item.getItem();
%>

    <tr>
    <td align="right" bgcolor="#ffffff">
    <%=item.getQuantity()%>
    </td>
    <td bgcolor="#ffffaa">
    <strong><a href="
    <%=request.getContextPath()%>/bookdetails?bookId=
    <%=bd.getBookId()%>"><%=bd.getTitle()%></a></strong>
    </td>
    ...
<%
  // End of while
  }
%>
```

Example: Scriptlet Result


Shopping Cart - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop
Bookmarks Location: http://localhost:8000/bookstore2/showcart
What's Related
Duke's Bookstore
Document: Done

You have 2 items in your shopping cart.

Quantity	Title	Price	
1	Duke: A Biography of the Java Evangelist	$10.75	Remove Item
1	Web Components for Web Developers	$17.75	Remove Item
	Subtotal:	$28.50	

Continue Shopping Check Out Clear Cart

Example: JSP page fragment

- Suppose we have the following JSP page fragment
  - <H2> sangHTML </H2>
  - <%= sangExpression() %>
  - <% sangScriptletCode(); %>

Example: Resulting Servlet Code

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
                        throws ServletException, IOException {
    response.setContentType(“text/html”);
    HttpSession session = request.getSession(true);
    JSPWriter out = response.getWriter();

    // Static HTML fragment is sent to output stream  in “as is” form
    out.println(“<H2>sangHTML</H2>”);

    // Expression is converted into String and then sent to output
    out.println(sangExpression());

    // Scriptlet is inserted as Java code within _jspService()
    sangScriptletCode();
    ...
}
```

Declarations

- Used to define variables or methods that get inserted into the main body of servlet class
  - Outside of _jspSevice() method
  - Implicit objects are not accessible to declarations
- Usually used with Expressions or Scriptlets
- For initialization and cleanup in JSP pages, use declarations to override jspInit() and jspDestroy() methods
- Format:
  - <%! method or variable declaration code %>
  - <jsp:declaration> method or variable declaration code </jsp:declaration>

Example: JSP Page fragment

```
<H1>Some heading</H1>
<%!
   private String randomHeading() {
        return(“<H2>” + Math.random() + “</H2>”);
   }
%>
<%= randomHeading() %>
```

Example: Resulting Servlet Code

```
public class xxxx implements HttpJSPPage {
  private String randomHeading() {
     return(“<H2>” + Math.random() + “</H2>”);
  }

  public void _jspService(HttpServletRequest request,
                                   HttpServletResponse response)
                                   throws ServletException, IOException {
    response.setContentType(“text/html”);
    HttpSession session = request.getSession(true);
    JSPWriter out = response.getWriter();
    out.println(“<H1>Some heading</H1>”);
    out.println(randomHeading());
    ...
  }
  ...
}
```

Example: Declaration

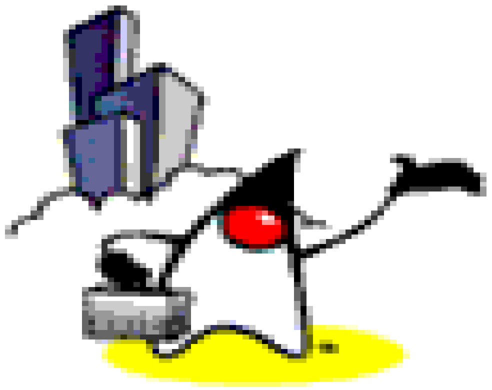
```
<%!
  private BookDBAO bookDBAO;

  public void jspInit() {
    ...
  }
  public void jspDestroy() {
    ...
  }
%>
```

Why XML Syntax?

- From JSP 1.2
- Examples
  - <jsp:expression>Expression</jsp:expression>
  - <jsp:scriptlet> Java code</jsp:scriptlet>
  - <jsp:declaration> declaration code </jsp:declaration>
- You can leverage
  - XML validation (via XML schema)
  - Many other XML tools
    - editor
    - transformer
    - Java APIs

Demo:

jsp_scripting
4009_jsp_basics2.zip



Including and Forwarding to JSP

Including Contents in a JSP Page

- Two mechanisms for including another Web resource in a JSP page
  - include directive
  - jsp:include element

Include Directive

- Is processed when the JSP page is translated into a servlet class
- Effect of the directive is to insert the text contained in another file-- either static content or another JSP page--in the including JSP page
- Used to include banner content, copyright information, or any chunk of content that you might want to reuse in another page
- Syntax and Example
  - <%@ include file="filename" %>
  - <%@ include file="banner.jsp" %>

jsp:include Element

- Is processed when a JSP page is executed
- Allows you to include either a static or dynamic resource in a JSP file
  - static: its content is inserted into the calling JSP file
  - dynamic: the request is sent to the included resource, the included page is executed, and then the result is included in the response from the calling JSP page
- Syntax and example
  - <jsp:include page="includedPage" />
  - <jsp:include page="date.jsp"/>

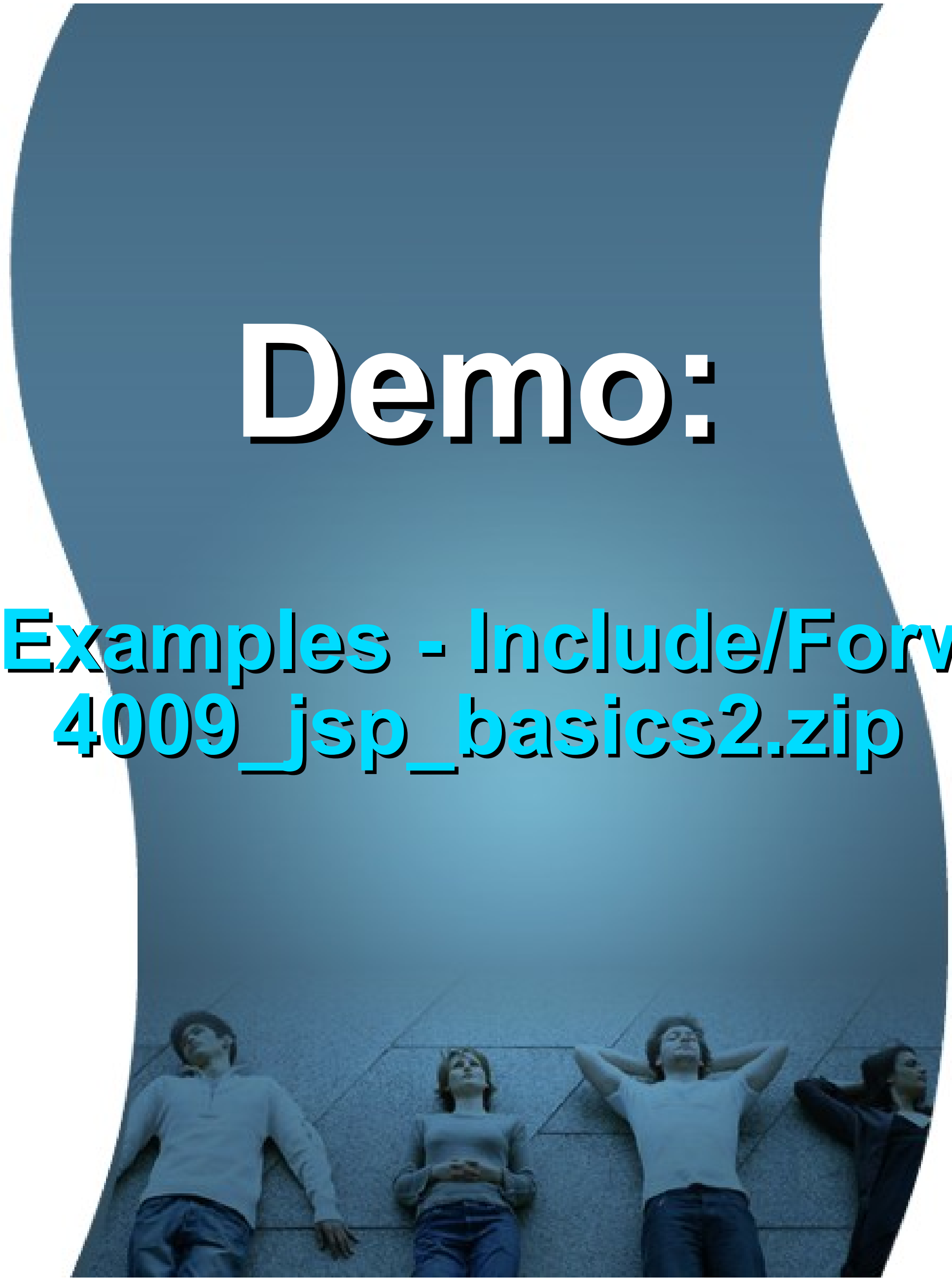
Which One to Use it?

- Use include directive if the file changes rarely
  - It is faster than *jsp:include*
- Use *jsp:include* for content that changes often
- Use *jsp:include* if which page to include cannot be decided until the main page is requested

Forwarding to another Web component

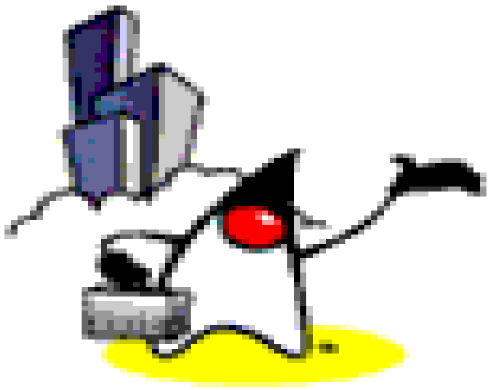
- Same mechanism as in Servlet
- Syntax
  - <jsp:forward page="/main.jsp" />
- Original request object is provided to the target page via jsp:parameter element

```
<jsp:forward page="..." >
  <jsp:param name="param1" value="value1"/>
</jsp:forward>
```



Demo:

JSPExamples - Include/Forward
4009_jsp_basics2.zip



Redirecting

Forwarding vs. Redirecting

- Forwarding
  - The browser is unaware of what has happened in the server side at the web container.
  - So it still thinks it is tending to the original request and displays the original URL in its address bar.
  - However, the page content displayed is from the second page.
- Redirecting
  - Instructs the client browser (via HTTP response header) to fetch another URL.
  - So the browser fetches entirely a new URL and displays the second URL in its address bar.
  - This could cause slight performance delay

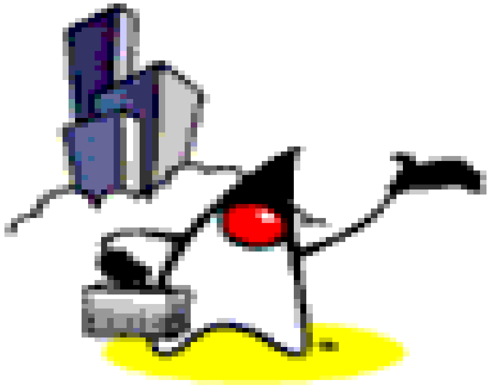
Redirecting to another Web component

- You can add scriptlet code in JSP page as following – this is a bad practice but there is no other way

```
<%
   String redirectURL = "http://jpassion.com/redirect";
   response.sendRedirect(redirectURL);
%>
```

- Or you do the following

```
<%
response.setStatus(HttpServletResponse.SC_MOVED_PERMANENTLY);
String newLocn = "/newpath/index.html";
response.setHeader("Location",newLocn);
%>
```



Directives

Directives

- Directives are messages to the JSP container in order to affect overall structure of the servlet
- Do not produce output into the current output stream
- Syntax
  - <%@ directive {attr=value}* %>

Three Types of Directives

- page: Communicate page dependent attributes and communicate these to the JSP container
  - <%@ page import="java.util.* %>
- include: Used to include text and/or code at JSP page translation-time
  - <%@ include file="header.html" %>
- Taglib: Indicates a tag library that the JSP container should interpret
  - <%@ taglib uri="mytags" prefix="codecamp" %>

Page Directives

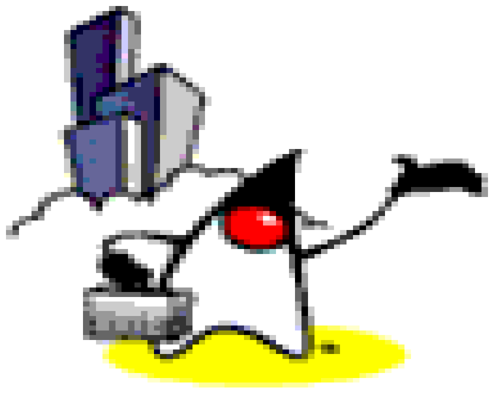
- Give high-level information about the servlet that results from the JSP page.
- Control
  - Which classes are imported
    - <%@ page import="java.util.* %>
  - What MIME type is generated
    - <%@ page contentType="MIME-Type" %>
  - How multithreading is handled
    - <%@ page isThreadSafe="true" %> <%!--Default --%>
    - <%@ page isThreadSafe="false" %>
  - What page handles unexpected errors
    - <%@ page errorPage="errorpage.jsp" %>

Implicit Objects

- A JSP page has access to certain implicit objects that are always available, without being declared first
- Created by container
- Corresponds to classes defined in Servlet

Implicit Objects

- request (HttpServletRequest)
- response (HttpServletRepsonse)
- session (HttpSession)
- application(ServletContext)
- out (of type JspWriter)
- config (ServletConfig)
- pageContext



Error Handling

Creating An Exception Error Page

- Determine the exception thrown
- In each of your JSP, include the name of the error page
  - `<%@ page errorPage="errorpage.jsp" %>`
- Develop an error page, it should include
  - `<%@ page isErrorPage="true" %>`
- In the error page, use the exception reference to display exception information
  - `<%= exception.toString() %>`

Example: initdestroy.jsp

```
<%@ page import="database.*" %>
<%@ page errorPage="errorpage.jsp" %>
<%!

  private BookDBAO bookDBAO;
  public void jspInit() {

    // retrieve database access object, which was set once per web
     application
    bookDBAO =
      (BookDBAO)getServletContext().getAttribute("bookDB");
    if (bookDBAO == null)
        System.out.println("Couldn't get database.");
  }

  public void jspDestroy() {
    bookDBAO = null;
  }
%>
```

Example: errorpage.jsp

```
<%@ page isErrorPage="true" %>
<%@ page import="java.util.*" %>
<%
   ResourceBundle messages =
     (ResourceBundle)session.getAttribute("messages");
   if (messages == null) {
      Locale locale=null;
      String language = request.getParameter("language");

      if (language != null) {
         if (language.equals("English")) {
          locale=new Locale("en", "");
         } else {
          locale=new Locale("es", "");
         }
      } else
         locale=new Locale("en", "");

      messages = ResourceBundle.getBundle("BookStoreMessages", locale);
      session.setAttribute("messages", messages);
   }
%> ...
```

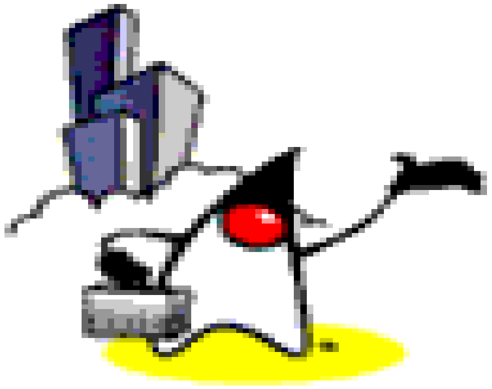
Example: errorpage.jsp

... (continued)

```
<html>
<head>
<title><%=messages.getString("ServerError")%></title>
</head>
<body bgcolor="white">
<h3>
<%=messages.getString("ServerError")%>
</h3>
<p>
<%= exception.getMessage() %>
</body>
</html>
```

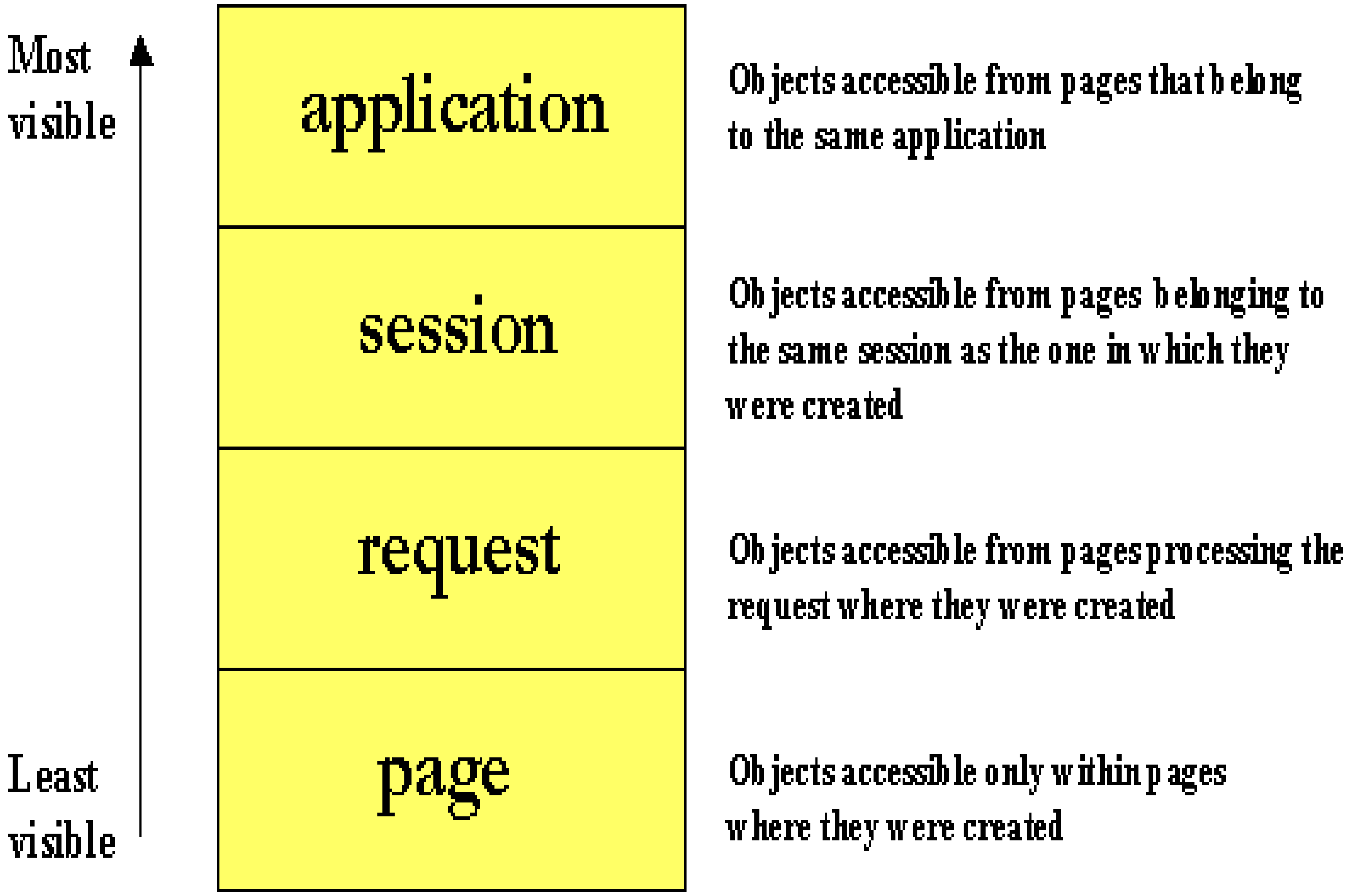
Demo:

JSPExamples – Error page
4009_jsp_basics2.zip



Scope Objects

Different Scope



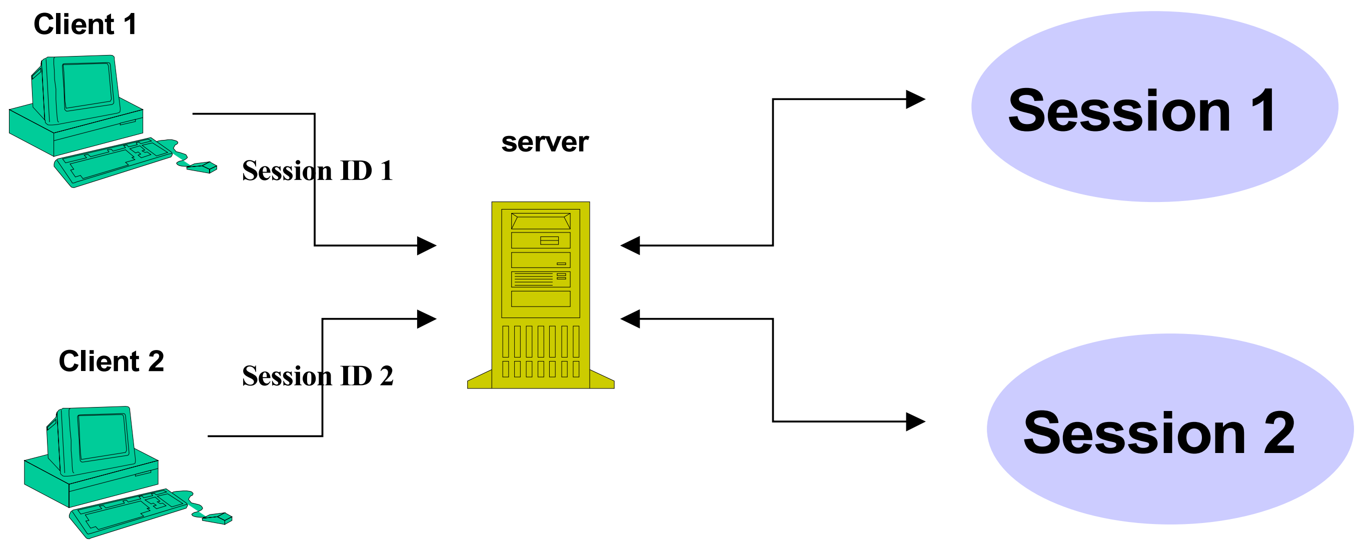
Objects accessible from pages that belong to the same application

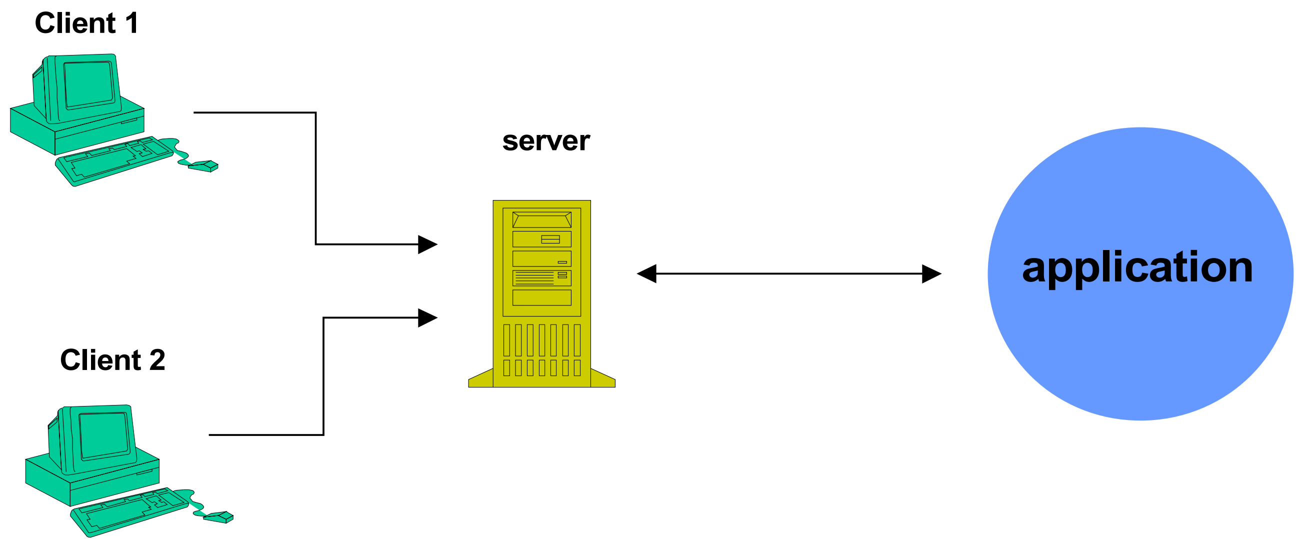
Objects accessible from pages belonging to the same session as the one in which they were created

Objects accessible from pages processing the request where they were created

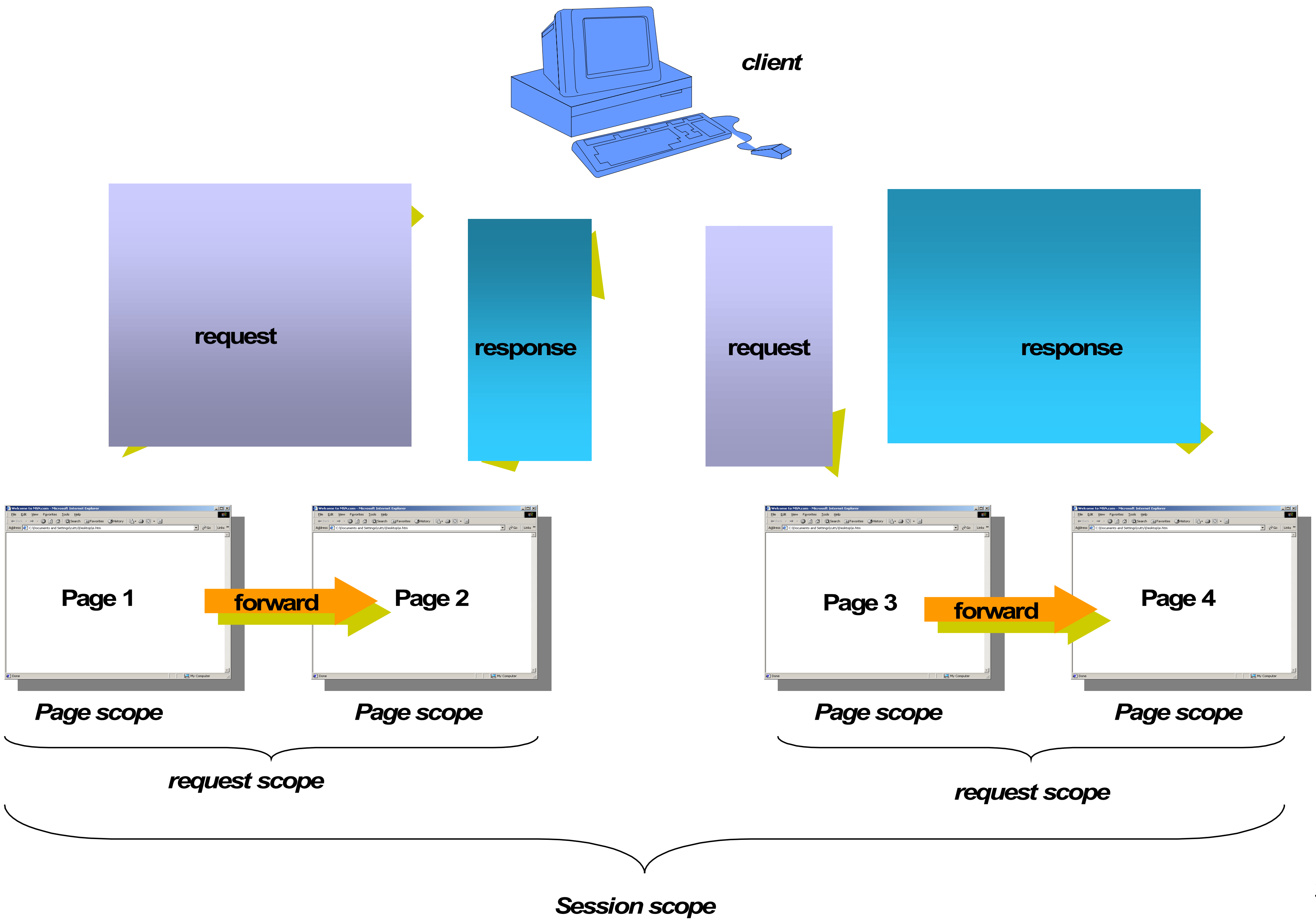
Objects accessible only within pages where they were created

Session, Application Scope





Session, Request, Page Scope



Demo:

jsp_scope_objects_include, jsp_scope_objects_forward 4009_jsp_basics2.zip

Thank you!

Sang Shin
http://www.javapassion.com
"Learn with JPassion!"