Java Server Pages (JSP)

Contents

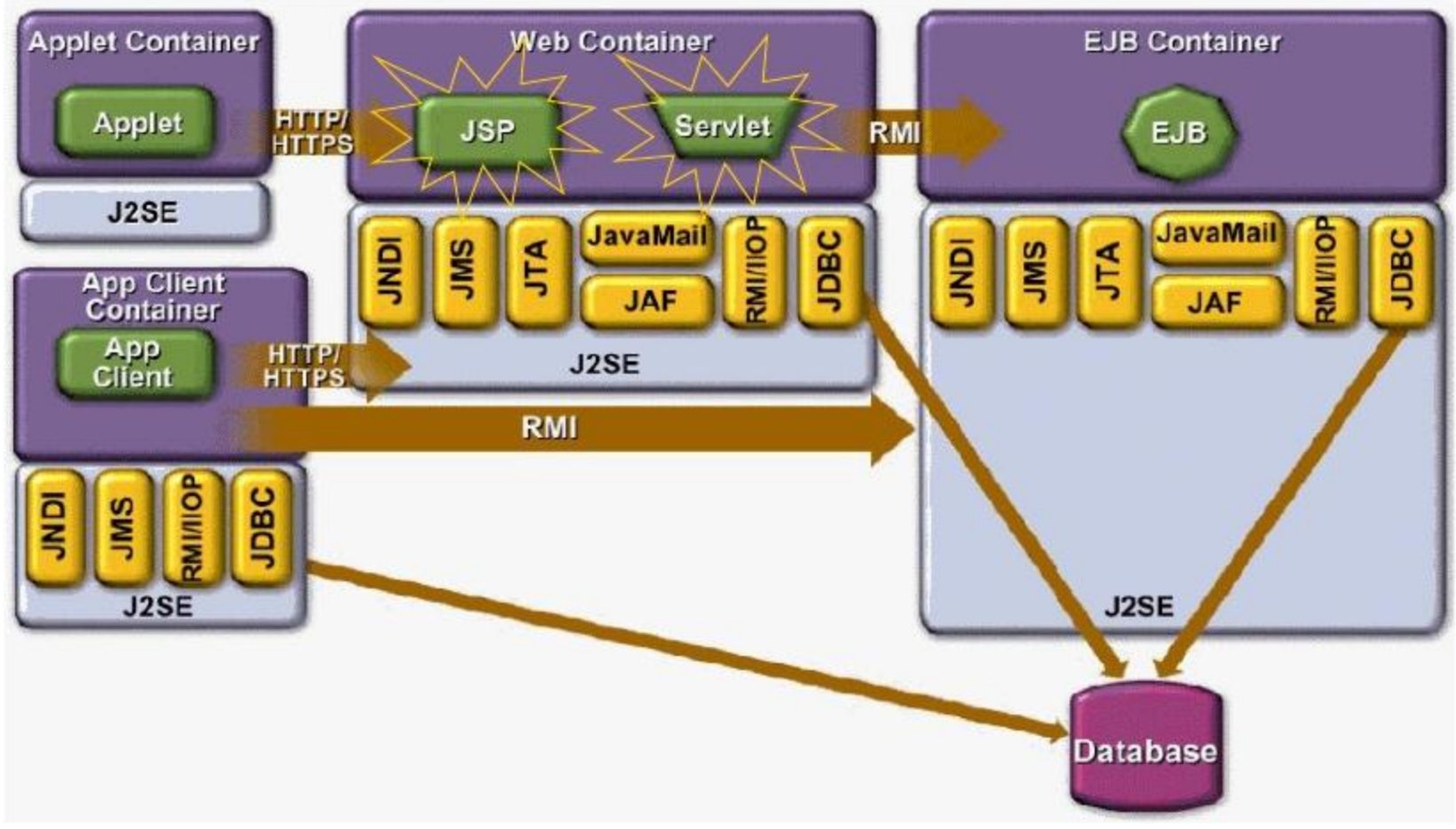
- JSP Introduction
- Life-cycle of JSP page
- Servlet & JSP code
- JSP Tags
- Including and Forwarding to Other Web Resources
- Error handling
- JavaBeans for JSP
- Dynamic contents generation techniques

JSP Introduction

What are Static & Dynamic Contents?

- Static contents
  - Typically static HTML page
  - Same display for everyone
- Dynamic contents
  - Dynamically generated based on conditions
  - Conditions could be
    - User identity
    - Time of the day
    - User entered values through forms and selections

JSP & Servlet as Web Components



What is JSP Page?

- JavaServer Pages (JSP) technology allows you to easily create web content that has both static and dynamic components.
- JSP technology makes available all the dynamic capabilities of Java Servlet technology but provides a more natural approach to creating static content.
- Static content and dynamic content can be intermixed
  - Static content
    - HTML, XML, Text
  - Dynamic content
    - Java code
    - Displaying properties of JavaBeans
    - Invoking business logic defined in Custom tags

What is JSP Technology?

- Enables separation of business logic from presentation
  - Presentation is in the form of HTML or XML/XSLT
  - Business logic is implemented as Java Beans or custom tags
  - Better maintainability, reusability
- Extensible via custom tags
- Builds on Servlet technology

A Simple JSP Page

```
<html>

<body>
Hello World!
<br>
Current time is <%= new java.util.Date() %>

</body>

</html>
```

Output


Netscape
File Edit View Go Bookmarks Tools Window Help
http://localhost:8080/hello2/da
Search
Mail Home Radio Netscape Search Bookmarks
http://localhost:8080/hello2/date
Hello World!
Current time is Sun Jan 12 21:03:14 EST 2003
Document: Done (16.604 secs)

Servlets and JSP - Comparison

Servlets

- HTML code in Java
- Not easy to author

JSP

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

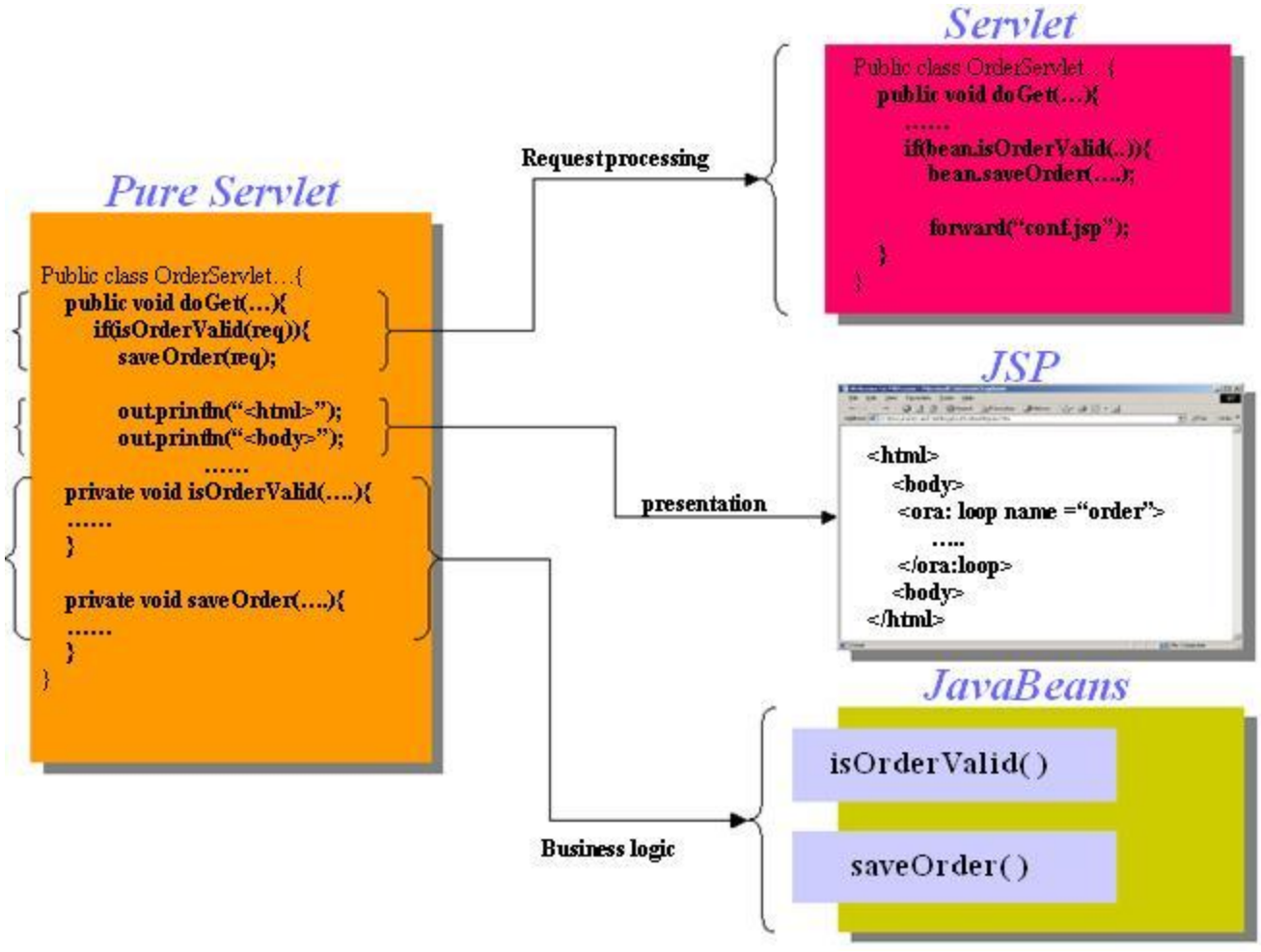
JSP Benefits

- Content and display logic are separated
- Supports software reuse
  - JavaBeans, Custom tags
- Automatic deployment
  - Recompile automatically when changes are made to JSP pages
- Easier to author web pages
- Platform-independent

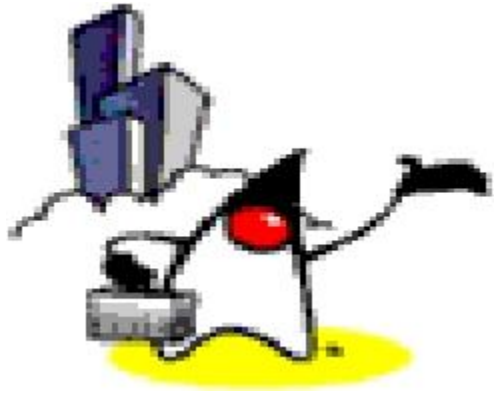
JSP or Servlet?

- Use both to leverage the strengths of each technology
  - Servlet's strength is “controlling and dispatching”
  - JSP's strength is “displaying”
- In practice, servlet and JSP are used together
  - via MVC (Model, View, Controller) architecture
  - Servlet handles Controller
  - JSP handles View

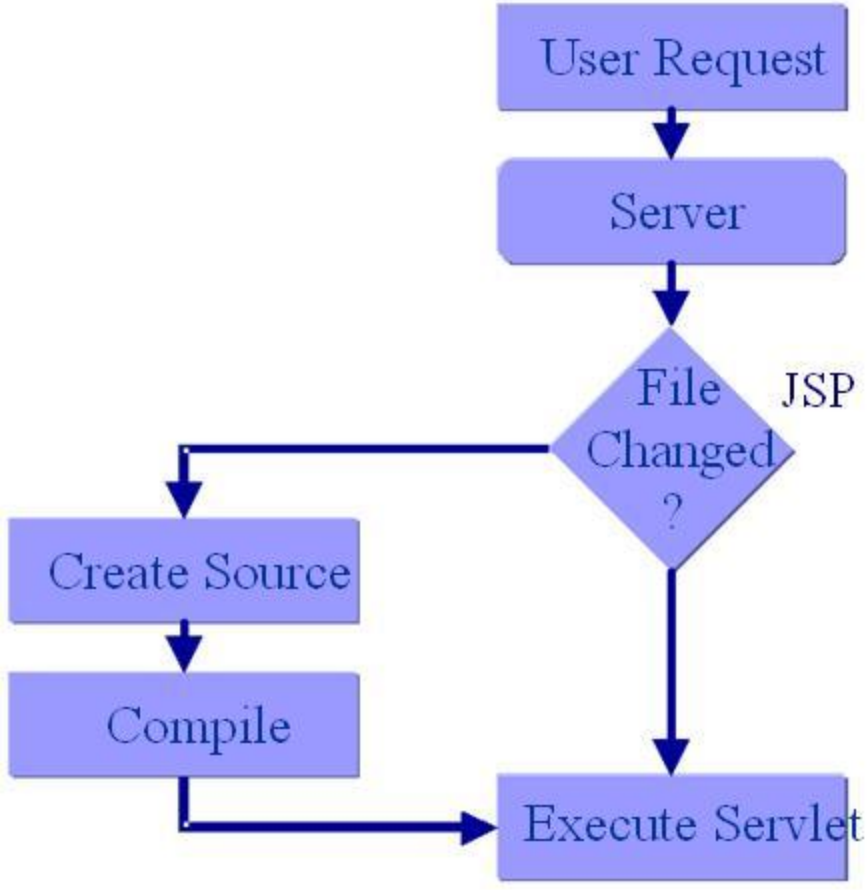
Separate Request processing From Presentation



Life-Cycle of a JSP Page



How Does JSP Work?



JSP Page Lifecycle Phases

- Translation phase
- Compile phase
- Execution phase

Translation/Compilation Phase

- JSP files get translated into servlet source code, which is then compiled
- Done by the container automatically
- The first time JSP page is accessed after it is deployed (or modified and redeployed)
- Static data is transformed into code that will emit data into the stream
- Scriptlet (Java code) within JSP page ends up being inserted into jspService() method of resulting servlet

javax.servlet.jsp package

- *javax.servlet.jsp* package defines two interfaces:
  - *JSPPage*
  - *HttpJspPage*
- These interfaces defines the three methods for the compiled JSP page. These methods are:
  - *jspInit()*
  - *jspDestroy()*
  - *_jspService(HttpServletRequest request,HttpServletResponse response)*
- In the compiled JSP file these methods are present.

javax.servlet.jsp package

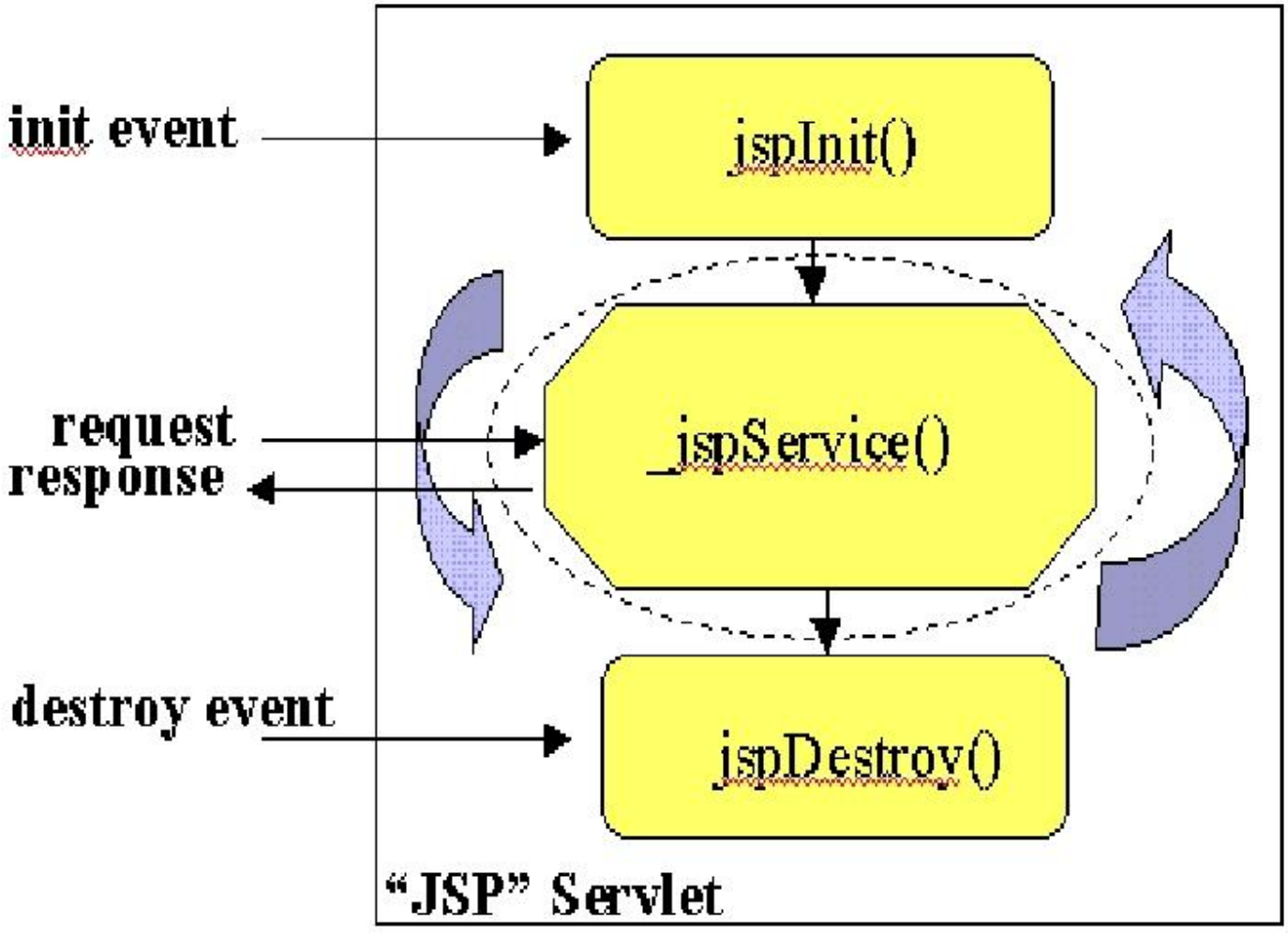
- Programmer can define *jspInit()* and *jspDestroy()* methods
- *_jspService(HttpServletRequest request,HttpServletResponse response)* method is generated by the JSP engine.

Example: initdestroy.jsp

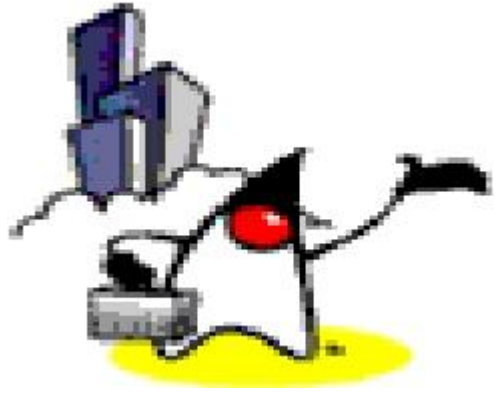
```
<%@ page import="database.*" %>
<%@ page errorPage="errorpage.jsp" %>
<%-- Declare initialization and finalization methods using JSP declaration --%>
<%!
    private BookDBAO bookDBAO;
    public void jspInit() {
        // retrieve database access object, which was set once per web
        //application
        bookDBAO =(BookDBAO)getServletContext().getAttribute("bookDB");
        if (bookDBAO == null)
                System.out.println("Couldn't get database.");
        }

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

JSP Lifecycle Methods during Execution Phase



Servlet & JSP code



GreetingServlet.java (Hello2)

```
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
/**
  * This is a simple example of an HTTP Servlet.  It responds to the GET
  * method of the HTTP protocol.
  */
public class GreetingServlet extends HttpServlet {
    public void doGet (HttpServletRequest
    request,HttpServletResponse response)
                    throws ServletException, IOException{
        response.setContentType("text/html");
        response.setBufferSize(8192);
        PrintWriter out = response.getWriter();
        // then write the data of the response
        out.println("<html>" +"<head><title>Hello</title></head>");
```

GreetingServlet.java

```
    // then write the data of the response
    out.println("<body  bgcolor=\"#ffffff\">" +
            "<img src=\"duKe.waving.gif\">" +
            "<h2>Hello, my name is DuKe. What's yours?</h2>" +
            "<form method=\"get\">" +
            "<input type=\"text\" name=\"username\" size=\"25\">" +
            "<p></p>" +
            "<input type=\"submit\" value=\"Submit\">" +
            "<input type=\"reset\" value=\"Reset\">" +
            "</form>");

    String username = request.getParameter("username");

    // dispatch to another web resource
    if ( username != null && username.length() > 0 ) {
            RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/response");
            if (dispatcher != null)
                    dispatcher.include(request, response);
    }
    out.println("</body></html>");
    out.close();
    }
    public String getServletInfo() {            return "The Hello servlet says hello.";         }
}
```

greeting.jsp

```
<html>
<head><title>Hello</title></head>
<body bgcolor="white">
<img src="duKe.waving.gif">
<h2>My name is DuKe. What is yours?</h2>

<form method="get">
<input type="text" name="username" size="25">
<p></p>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form>
<%
    String username = request.getParameter("username");
    if ( username != null && username.length() > 0 ) {
%>
    <%@include file="response.jsp" %>
<%
    }
%>
</body>
</html>
```

ResponseServlet.java

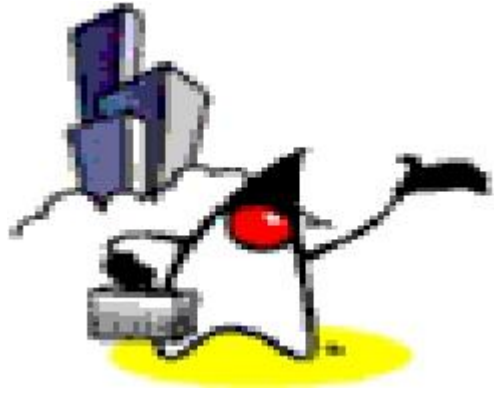
```
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

// This is a simple example of an HTTP Servlet.  It responds to the GET
// method of the HTTP protocol.
public class ResponseServlet extends HttpServlet {
      public void doGet (HttpServletRequest request, HttpServletResponse response)
                              throws ServletException, IOException{
            PrintWriter out = response.getWriter();
            // then write the data of the response
            String username = request.getParameter("username");
            if ( username != null && username.length() > 0 )
                              out.println("<h2>Hello, " + username + "!</h2>");
      }
      public String getServletInfo() {
            return "The Response servlet says hello.";
      }
}
```

response.jsp

```
<h2><font color="black">Hello,<%=username%>!</font></h2>
```

JSP tags



JSP tags

- In JSP tags can be devided into 4 different types:
- **Declarations**
  This tag is used for defining the functions and variables to be used in the JSP.
- **Scriplets**
  In this tag we can insert any amount of valid java code and these codes are placed in *_jspService* method by the JSP engine.
- **Expressions**
  We can use this tag to output any data on the generated page. These data are automatically converted to string and printed on the output stream.
- **Directives**
  In the directives we can import packages, define error handling pages or the session information of the JSP page.

JSP comments

- JSP comments:

  <%--comments--%>

JSP DECLARATION

- Syntax of JSP Declarations are:

  <%! //java codes %>
- We can embed any amount of java code in the JSP Declarations.
- Variables and functions defined in the declarations are class level and can be used anywhere in the JSP page.
- Declare.jsp

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

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() {
    ...
  }
%>
```

JSP Scriptles

- Syntax of JSP Scriptles are:

  <% //java codes %>
- We can embed any amount of java code in the JSP Scriptlets.
- JSP Engine places these code in the *_jspService()* method.
- Can do things expressions alone cannot do
  - updating database
  - executing code that contains loops, conditionals
- Can use predefined variables (implicit objects)

Example: Scriptlets

- Display query string

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

- Setting 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
}
%>
```

- LoopExample

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();
    ...
}
```

9 implicit objects

- A JSP page has access to certain implicit objects that are always available, without being declared first.
- Created by container
- Variables available to the JSP Scriptlets are:
  - application : an instance of javax.servlet.ServletContext
  - config
    An instance of javax.servlet.ServletConfig
  - exception
    an instance of java.lang.Throwable
  - Request: an instance of javax.servlet.http.HttpServletRequest

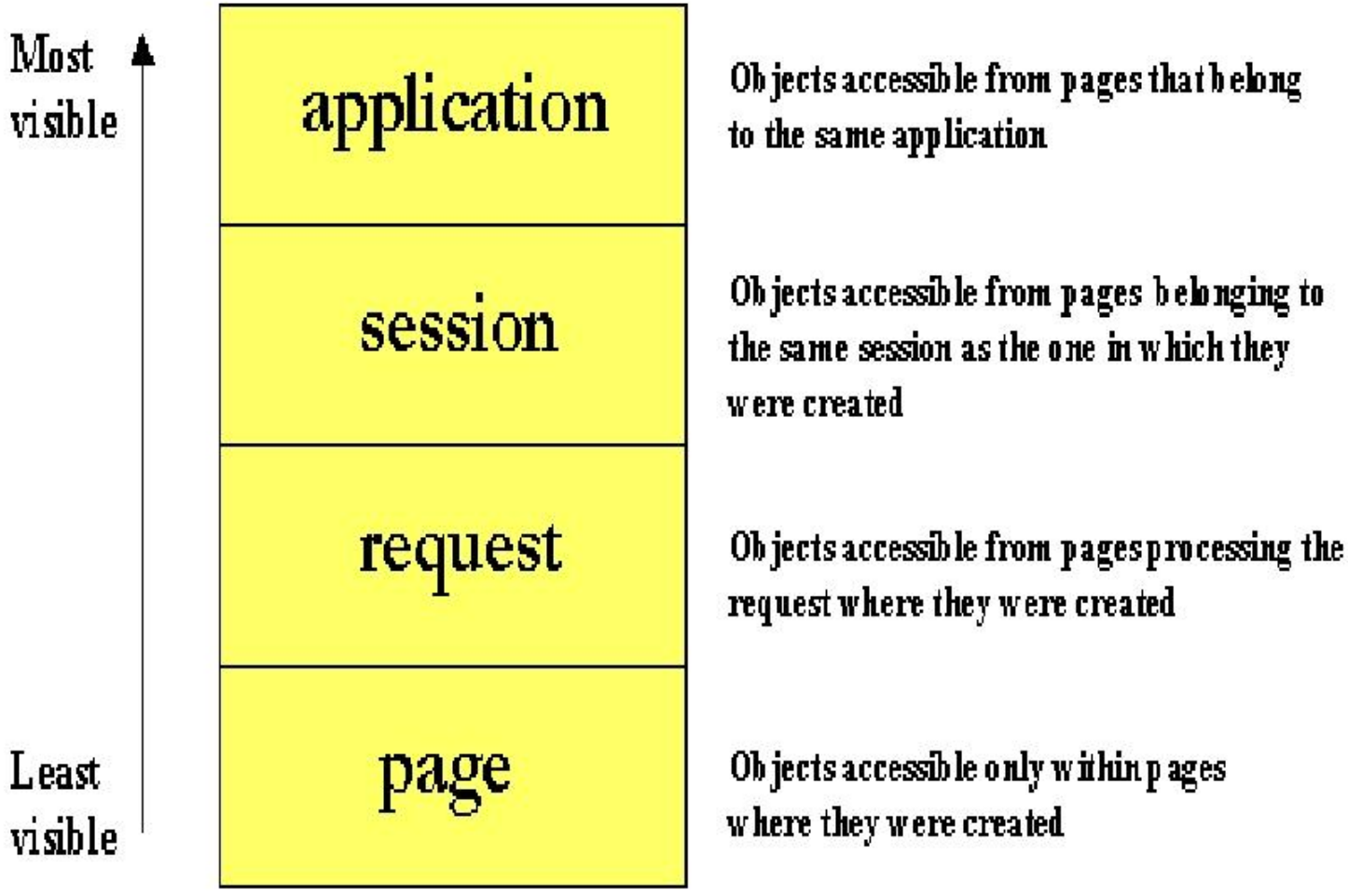
9 implicit objects

- response:
  an instance of
  javax.servlet.http.*HttpServletResponse*.
- session: (SessionExample)
  an instance of javax.servlet.http.HTTPSession
- out:
  an object of output stream and is used to send any output to the client.
- pageContext

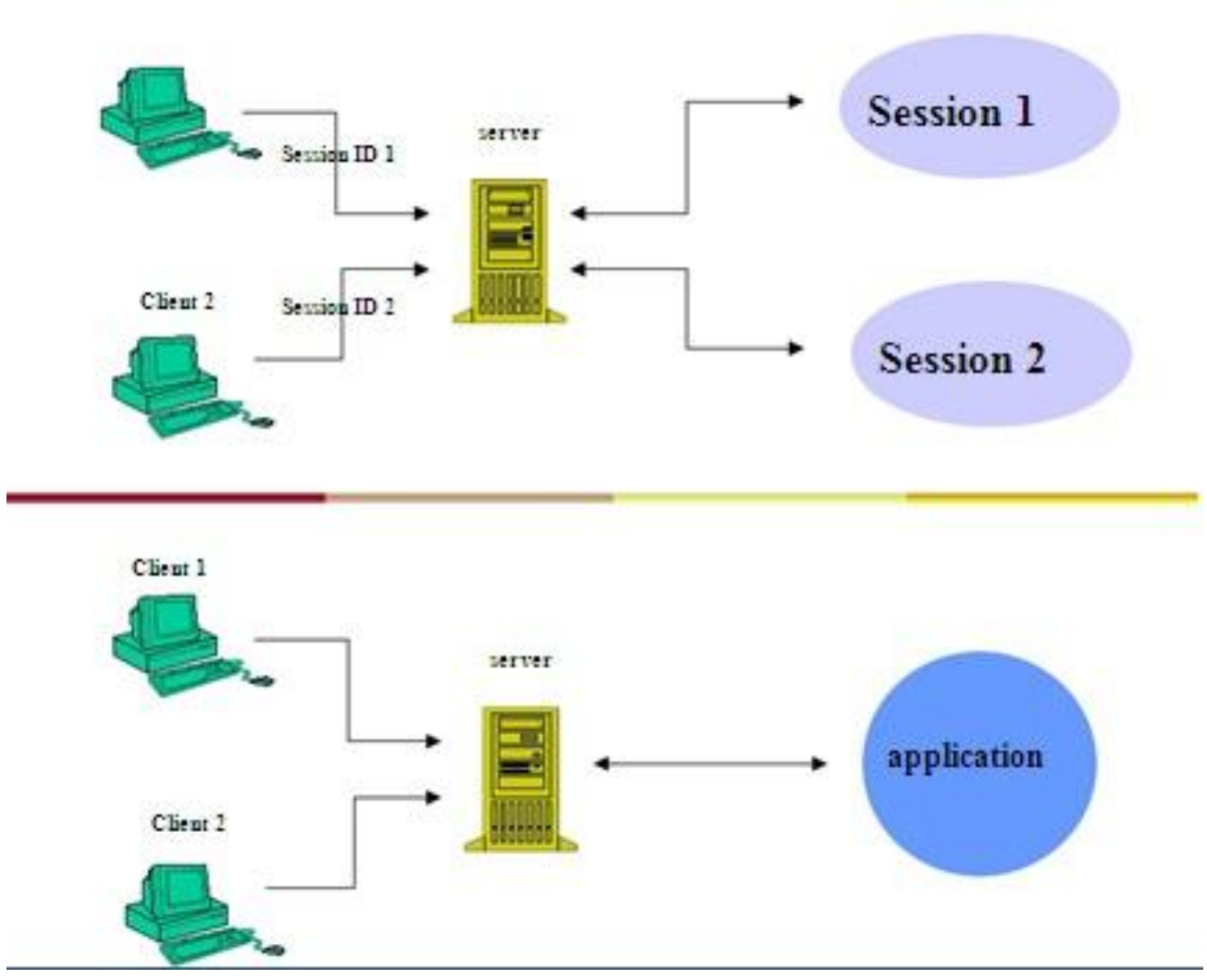
  an instance of javax.servlet.jsp.PageContext.
- Pages

  this

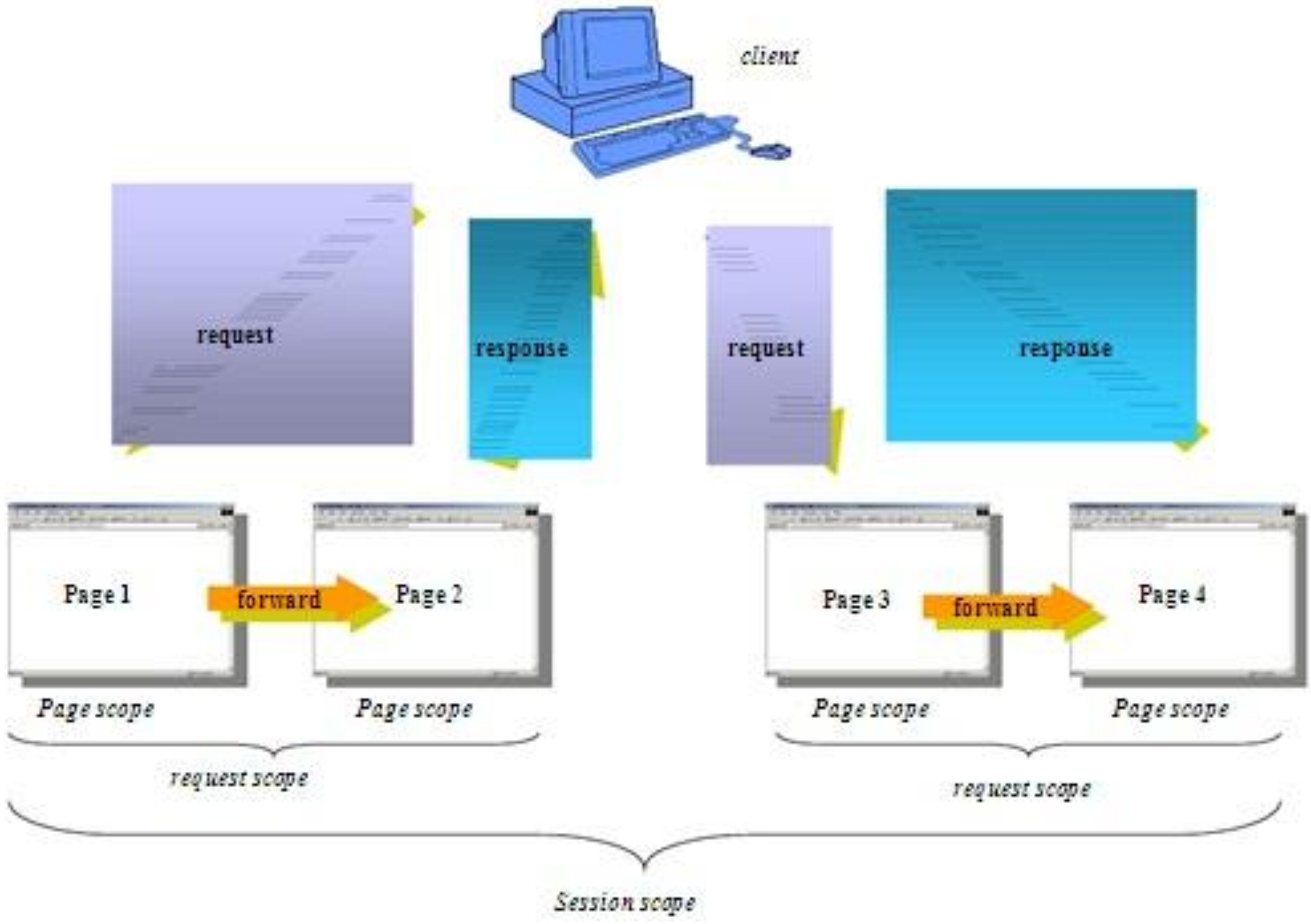
Different Scope



Session, Application Scope



Session, Request, Page Scope



JSP Expressions

- Syntax of JSP Expressions are:

  <%="Any thing" %>

- “Any thing” means anything that will be converted to the String and be displayed.
- Example:

  <%="Hello World!" %>

JSP 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 expression

Example: Expressions

- Display current time using Date class
  - Current time: <%= new java.util.Date() %>
- Display random number using Math class
  - Random number: <%= Math.random() %>

Example: Expressions

- Use implicit objects
  - Your hostname: <%= request.getRemoteHost() %>
  - Your parameter: <%= request.getParameter("yourParameter") %>
  - Server: <%= application.getServerInfo() %>
  - Session ID: <%= session.getId() %>

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 of JSP directives is:

```
<%@directive attribute="value" %>
```

Directives

- Where directive may be:
  - page: page is used to provide the information about it. Example: <%@page language="java" %>
  - include: include is used to include a file in the JSP page. Example: <%@ include file="/header.jsp" %>
  - taglib: taglib is used to use the custom tags in the JSP pages (custom tags allows us to defined our own tags). Example: <%@ taglib uri="tlds/taglib.tld" prefix="mytag" %>

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=“text/html" %>
  - What page handles unexpected errors
    - <%@ page errorPage="errorpage.jsp" %>

JSP Action Tags

- The JSP Actions tags enables the programmer to use the functions built in Servlet container.
- **jsp:include**
  The **jsp:include** action work as a subroutine, the Java servlet temporarily passes the request and response to the specified JSP/Servlet. Control is then returned back to the current JSP page.

- **jsp:param**
  The **jsp:param** action is used to add the specific parameter to current request. The jsp:param tag can be used inside a jsp:include, jsp:forward or jsp:params block.

JSP Action Tags

- **jsp:forward**
  The **jsp:forward** tag is used to hand off the request and response to another JSP or servlet. In this case the request never return to the calling JSP page.
- **jsp:useBean**
  The **jsp:useBean** tag is used to instantiate an object of Java Bean or it can re-use existing java bean object.

JSP Action Tags

- **jsp:getProperty**
  The **jsp:getPropertyB** is used to get specified property from the JavaBean object.

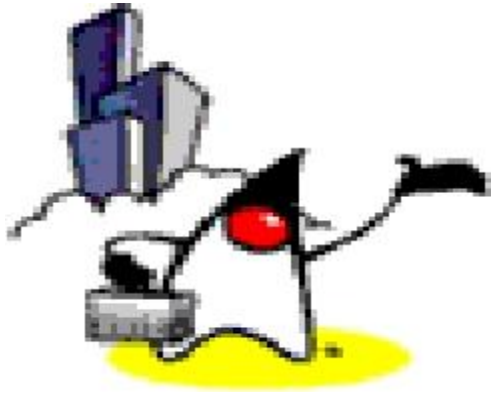
- **jsp:setProperty**
  The **jsp:setProperty** tag is used to set a property in the JavaBean object.

JSP Action Tags

- **jsp:plugin**
  The **jsp:plugin** tag actually generates the appropriate HTML code the embed the Applets correctly.
- **jsp:fallback**
  The **jsp:fallback** tag specify the message to be shown on the browser if applets is not supported by browser.
  Example:

```
<jsp:fallback>
    <p>Unable to load applet</p>
</jsp:fallback>
```

Including and Forwarding to Other Web Resource



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
- 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

Forwarding to another Web component

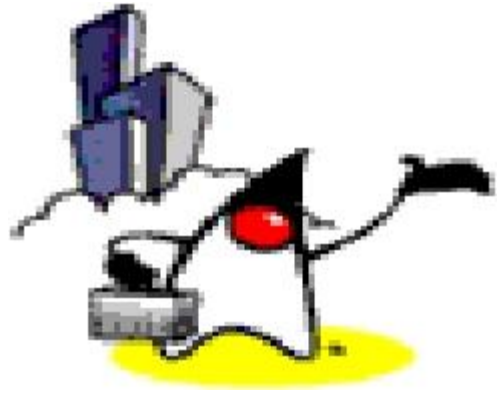
- Same mechanism as in Servlet
- Syntax

```
<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>
```

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() %>
- ErrorExample

JavaBeans for JSP

What are JavaBeans?

- Java classes that can be easily reused and composed together into an application
- Any Java class that follows certain design conventions can be a JavaBeans component
  - properties of the class
  - public methods to get and set properties
- Within a JSP page, you can create and initialize beans and get and set the values of their properties
- JavaBeans can contain business logic or data base access logic

JavaBeans Design Conventions

- JavaBeans maintain internal properties
- A property can be
  - Read/write, read-only, or write-only
- Properties should be accessed and set via getXxx and setXxx methods
  - PropertyClass getProperty() { ... }
  - PropertyClass setProperty() { ... }
- JavaBeans must have a zero-argument (empty) constructor

Example: JavaBeans

```
public class Currency {
        private Locale locale;
        private double amount;
        public Currency() {
                locale = null;
                amount = 0.0;                   }
        public void setLocale(Locale l) {
                        locale = l;             }
        public void setAmount(double a) {
                        amount = a;     }
        public String getFormat() {
                NumberFormat nf =
                        NumberFormat.getCurrencyInstance(locale);
                        return nf.format(amount);   }
 }
```

Why Use JavaBeans in JSP Page?

- A JSP page can create and use any type of Java programming language object within a declaration or scriptlet like following:

```
<%
    ShoppingCart cart =
(ShoppingCart)session.getAttribute("cart");
    // If the user has no cart, create a new one
    if (cart == null) {
        cart = new ShoppingCart();
        session.setAttribute("cart", cart);
    }
%>
```

Compare the Two

```
<%
        ShoppingCart cart =
    (ShoppingCart)session.getAttribute("cart");
        // If the user has no cart object as an attribute in Session scope
        // object, then create a new one. Otherwise, use the existing
        // instance.
        if (cart == null) {
                cart = new ShoppingCart();
                session.setAttribute("cart", cart);
        }
    %>
```

versus

```
    <jsp:useBean id="cart" class="cart.ShoppingCart"
scope="session"/>
```

Why Use JavaBeans in JSP Page?

- No need to learn Java programming language for page designers
- Stronger separation between content and presentation
- Higher reusability of code
- Simpler object sharing through built-in sharing mechanism
- Convenient matching between request parameters and object properties

Creating a JavaBeans

- Declare that the page will use a bean that is stored within and accessible from the specified scope by jsp:useBean element

```
<jsp:useBean id="beanName"
   class="fully_qualified_classname" scope="scope"/>
      or
 <jsp:useBean id="beanName"
   class="fully_qualified_classname" scope="scope">
    <jsp:setProperty .../>
 </jsp:useBean>
```

Setting JavaBeans Properties

- Via JSP:setProperty
  - <jsp:setProperty name="beanName“ property="propName" value="string constant"/>
  - “beanName” must be the same as that specified for the id attribute in a useBean element
  - There must be a setPropName method in the bean

Example: jsp:setProperty with Expression

```
<jsp:useBean id="currency"
   class="util.Currency" scope="session">
<jsp:setProperty name="currency"
   property="locale" value="<%=
   request.getLocale() %>"/>
</jsp:useBean>
<jsp:setProperty name="currency"
   property="amount"value="<%=cart.getTotal(
   )%>"/>
```

Getting JavaBeans Properties

- via JSP:setProperty
  - <jsp:getProperty name="beanName“ property="propName"/>
  - “beanName” must be the same as that specified for the id attribute in a useBean element
  - There must be a “getPropName()” method in a bean

Getting JavaBeans Properties without Converting it to String

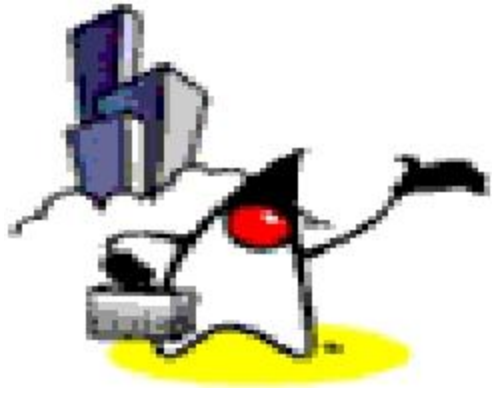
- Must use a scriptlet
- Format

```
<% Object o = beanName.getPropName(); %>
```

- Example

```
<%
  // Print a summary of the shopping cart
  int num = cart.getNumberOfItems();
  if (num > 0) {
%>
```

Dynamic Content Generation Techniques



Dynamic Contents Generation Techniques

- Various techniques can be chosen depending on the following factors
  - Size and complexity of the project
  - Requirements on reusability of code, maintainability, degree of robustness

Dynamic Contents Generation Techniques with JSP

- Call Java code directly within JSP
- Call Java code indirectly within JSP
- Use JavaBeans within JSP
- Develop and use your own custom tags
- Leverage 3rd-party custom tags or JSTL

Call Java code directly

- Place all Java code in JSP page
- Suitable only for a very simple Web application
  - hard to maintain
  - hard to reuse code
  - hard to understand for web page authors
- Not recommended for relatively sophisticated Web applications
  - weak separation between contents and presentation

Call Java code indirectly

- Develop separate utility classes
- 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 weak separation between contents and presentation, however.

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 method

Develop and Use Custom Tags

- Develop sophisticated components called custom tags
  - Custom tags are specifically designed for JSP
- More powerful than JavaBeans component
  - More than just getter and setter methods
- reusability, maintainability, robustness
- Development of custom tags are more difficult than creating JavaBeans, however

Use 3rd-party Custom tags or JSTL

- There are many open source and commercial custom tags available
  - Apache Struts
- JSTL (JSP Standard Tag Library) standardize the set of custom tags that should be available over Java EE platform at a minimum

The End!