Java Bean and MVC

Including Files & Applets in JSP Documents

- Including the output of JSP, HTML or plain text pages at the time the client requests the page
- Including JSP files at the time the main page is translated into a servlet
- Including applets that use the Java Plug-In

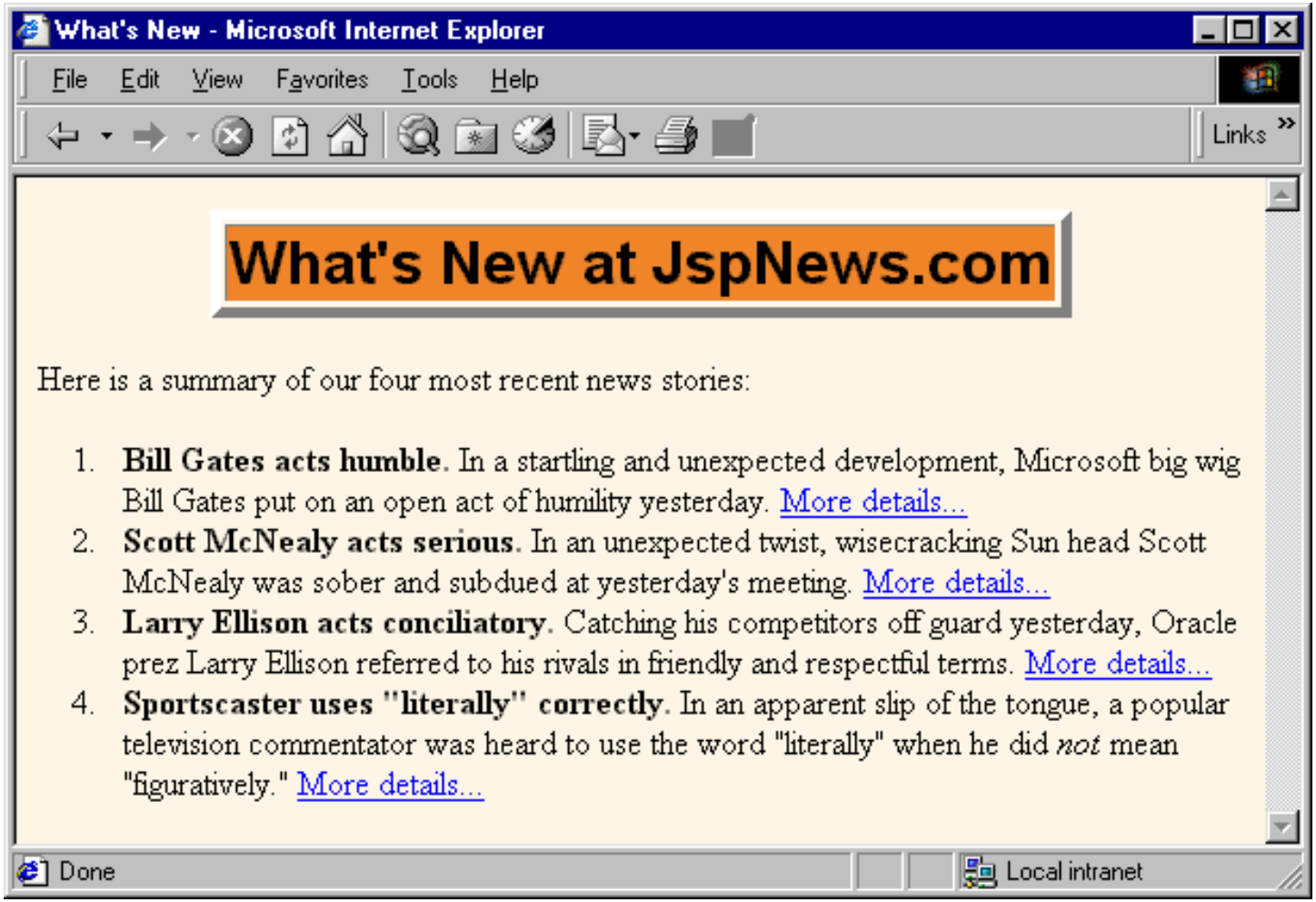
Including Files at Request Time

- **Format**
  - <jsp:include page="Relative URL" flush="true" />
- **Purpose**
  - To reuse JSP, HTML, or plain text content
  - JSP content cannot affect main page: only *output* of included JSP page is used
  - To permit updates to the included content without changing the main JSP page(s)
- **Notes**
  - You are not permitted to specify `false` for the flush attribute in JSP 1.1 and earlier
  - Don't forget that trailing slash

Including Files: Example Code

```
...
<BODY>
<CENTER>
<TABLE BORDER=5>
  <TR><TH CLASS="TITLE">
      What's New at JspNews.com</TABLE>
</CENTER>
<P>
Here is a summary of our four most recent news stories:
<OL>
  <LI><jsp:include page="news/Item1.html" flush="true" />
  <LI><jsp:include page="news/Item2.html" flush="true" />
  <LI><jsp:include page="news/Item3.html" flush="true" />
  <LI><jsp:include page="news/Item4.html" flush="true" />
</OL>
</BODY></HTML>
```

Including Files: Example Result



Including Files at Page Translation Time

- **Format**
  - <%@ include file="Relative URL" %>
- **Purpose**
  - To reuse JSP content in multiple pages, *where JSP content affects main page*
- **Notes**
  - Servers are not required to detect changes to the included file, and in practice they don't.
  - Thus, you need to change the JSP files whenever the included file changes.
  - You can use OS-specific mechanisms such as the Unix "touch" command, or
    - <%@ include file=”Testing.jsp" %>

Differences Between jsp:include and @include

- **jsp:include includes the *output* of the designated page**
  - @include includes the actual *code*
- **jsp:include occurs at *request* time**
  - @include occurs at *page translation* time
- **With jsp:include, the main page and the included page become *two* separate servlets**
  - With @include, they become parts of a *single* servlet
- **jsp:include automatically handles changes to the included file**
  - @include might not (big maintenance problem!)

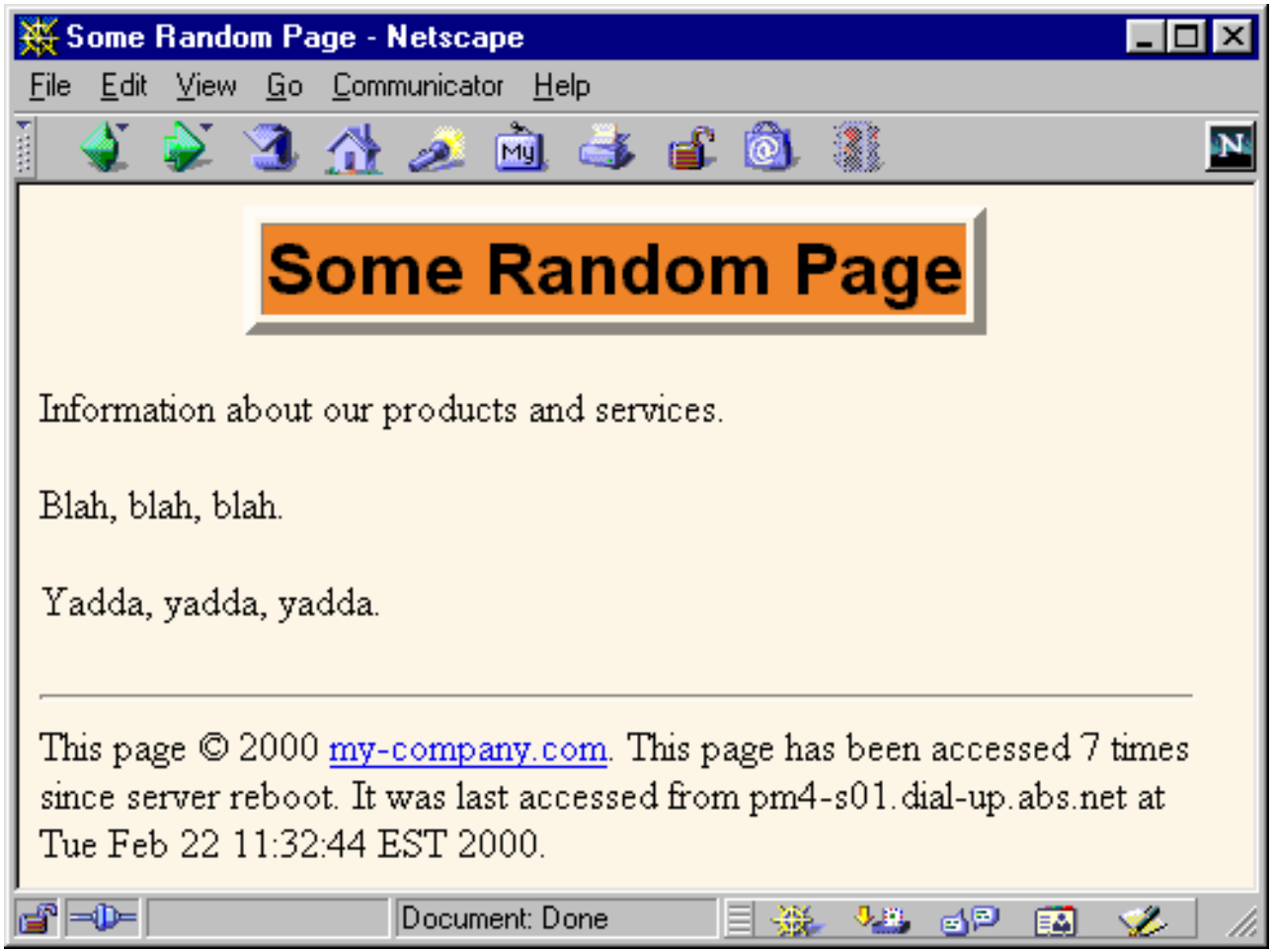
Reusable JSP Content: ContactSection.jsp

```
<%@ page import="java.util.Date" %>
<%-- The following become fields in each servlet that
     results from a JSP page that includes this file. --%>
<%!
private int accessCount = 0;
private Date accessDate = new Date();
private String accessHost = "<I>No previous access</I>";
%>
<P>
<HR>
This page &copy; 2000
<A HREF="http//www.my-company.com/">my-company.com</A>.
This page has been accessed <%= ++accessCount %>
times since server reboot. It was last accessed from
<%= accessHost %> at <%= accessDate %>.
<% accessHost = request.getRemoteHost(); %>
<% accessDate = new Date(); %>
```

Using the JSP Content

```
...
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Some Random Page</TABLE>
<P>
Information about our products and services.
<P>
Blah, blah, blah.
<P>
Yadda, yadda, yadda.

<%@ include file="ContactSection.jsp" %>

</BODY>
</HTML>
```

Using the JSP Content: Result



Including Applets for the Java Plug-In

- **Neither Netscape 4 nor IE 5/6 support Java 2**
  - Netscape 6 supports JDK 1.3
- **Problems using applets**
  - In order to use Swing, you must send the Swing files over the network. This process is time consuming and fails in Internet Explorer 3 and Netscape 3.x and 4.01-4.05 (which only support JDK 1.02).
  - You cannot use Java 2D
  - You cannot use the Java 2 collections package
  - Your code runs more slowly, since most compilers for the Java 2 platform are significantly improved over their 1.1 predecessors

Solution (?)
Use the Java Plug-In

- **Only a viable option for intranets**
  - Requires *each* client to install large browser plug-in
- **Cannot use simple APPLET tag**
- **Need long and ugly OBJECT tag for Internet Explorer**
- **Need long and ugly EMBED tag for Netscape**
- **The jsp:plugin action lets you write a simple tag that gets translated into the OBJECT and EMBED tags that are required**
  - Warning: jsp:plugin fails in JRun 3 SP2

Using jsp:plugin

- **APPLET Tag**
  -
```
<APPLET CODE="MyApplet.class"
        WIDTH=475 HEIGHT=350>
</APPLET>
```
- **Equivalent jsp:plugin**
  -
```
<jsp:plugin type="applet"
            code="MyApplet.class"
            width="475" height="350">
</jsp:plugin>
```
- **Reminder**
  - JSP element and attribute names are case sensitive
  - All attribute values must be in single or double quotes
  - This is like XML but unlike HTML

Attributes of the jsp:plugin Element

- **type**
  - For applets, this should be "applet". Use "bean" to embed JavaBeans elements in Web pages.
- **code**
  - Used identically to CODE attribute of APPLET, specifying the top-level applet class file
- **width, height**
  - Used identically to WIDTH, HEIGHT in APPLET
- **codebase**
  - Used identically to CODEBASE attribute of APPLET
- **align**
  - Used identically to ALIGN in APPLET and IMG

Attributes of the jsp:plugin Element (Cont.)

- **hspace, vspace**
  - Used identically to HSPACE, VSPACE in APPLET,
- **archive**
  - Used identically to ARCHIVE attribute of APPLET, specifying a JAR file from which classes and images should be loaded
- **name**
  - Used identically to NAME attribute of APPLET, specifying a name to use for inter-applet communication or for identifying applet to scripting languages like JavaScript.
- **title**
  - Used identically to rarely used TITLE attribute

Attributes of the jsp:plugin Element (Cont.)

- **jreversion**
  - Identifies version of the Java Runtime Environment (JRE) that is required. Default is 1.1.
- **iepluginurl**
  - Designates a URL from which plug-in for Internet Explorer can be downloaded. Users who don't already have the plug-in installed will be prompted to download it from this location. Default value will direct user to Sun site, but for intranet use you might want to direct user to a local copy.
- **nspluginurl**
  - Designates a URL from which plug-in for Netscape can be downloaded. Default value will direct user to Sun site, but for intranet use you might want local copy.

The jsp:param and jsp:params Elements

- **PARAM Tags**
  - ```
    <APPLET CODE="MyApplet.class"
            WIDTH=475 HEIGHT=350>
      <PARAM NAME="PARAM1" VALUE="VALUE1">
      <PARAM NAME="PARAM2" VALUE="VALUE2">
    </APPLET>
    ```
- **Equivalent jsp:param**
  - ```
    <jsp:plugin type="applet"
                code="MyApplet.class"
                width="475" height="350">
      <jsp:params>
        <jsp:param name="PARAM1" value="VALUE1" />
        <jsp:param name="PARAM2" value="VALUE2" />
      </jsp:params>
    </jsp:plugin>
    ```

The jsp:fallback Element

- **APPLET Tag**
  -
```
<APPLET CODE="MyApplet.class"
        WIDTH=475 HEIGHT=350>
  <B>Error: this example requires Java.</B>
</APPLET>
```
- **Equivalent jsp:plugin with jsp:fallback**
  -
```
<jsp:plugin type="applet"
            code="MyApplet.class"
            width="475" height="350">
  <jsp:fallback>
    <B>Error: this example requires Java.</B>
  </jsp:fallback>
</jsp:plugin>
```

Example Using Plugin

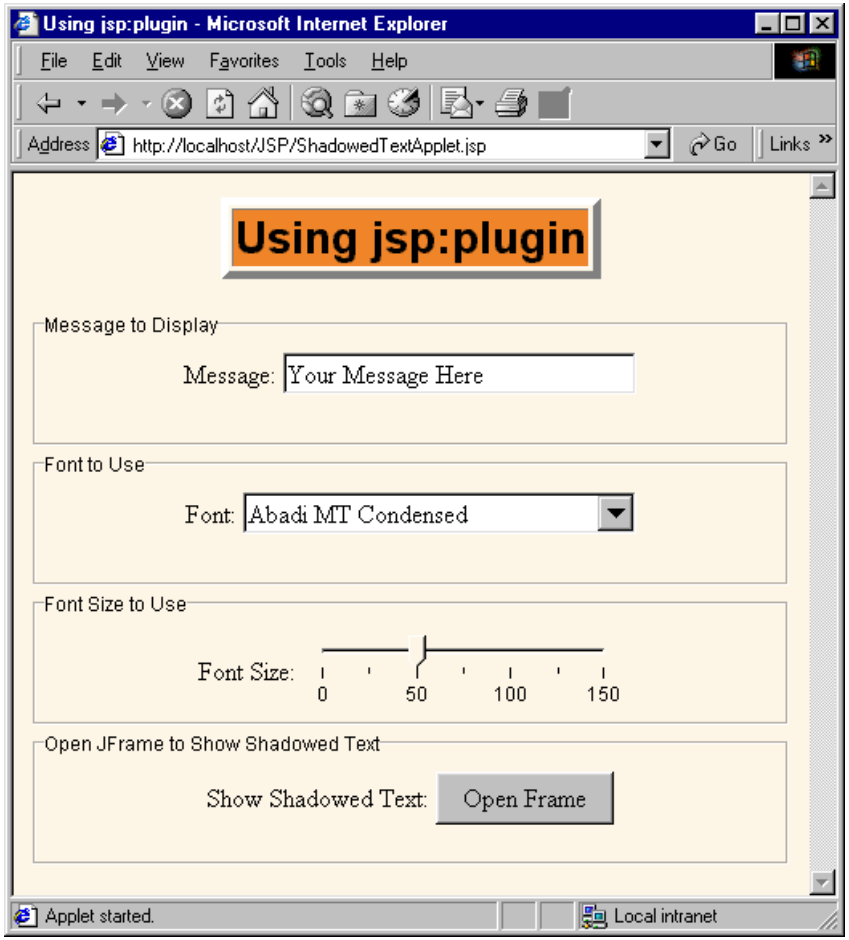
```
<DOCTYPE ...>...
<jsp:plugin
      type="applet"
      code="coreservlets.ShadowedTextApplet.class"
      width="475" height="350">
  <jsp:params>
    <jsp:param name="MESSAGE"
               value="Your Message Here" />
  </jsp:params>
</jsp:plugin>
...
</BODY></HTML>
```

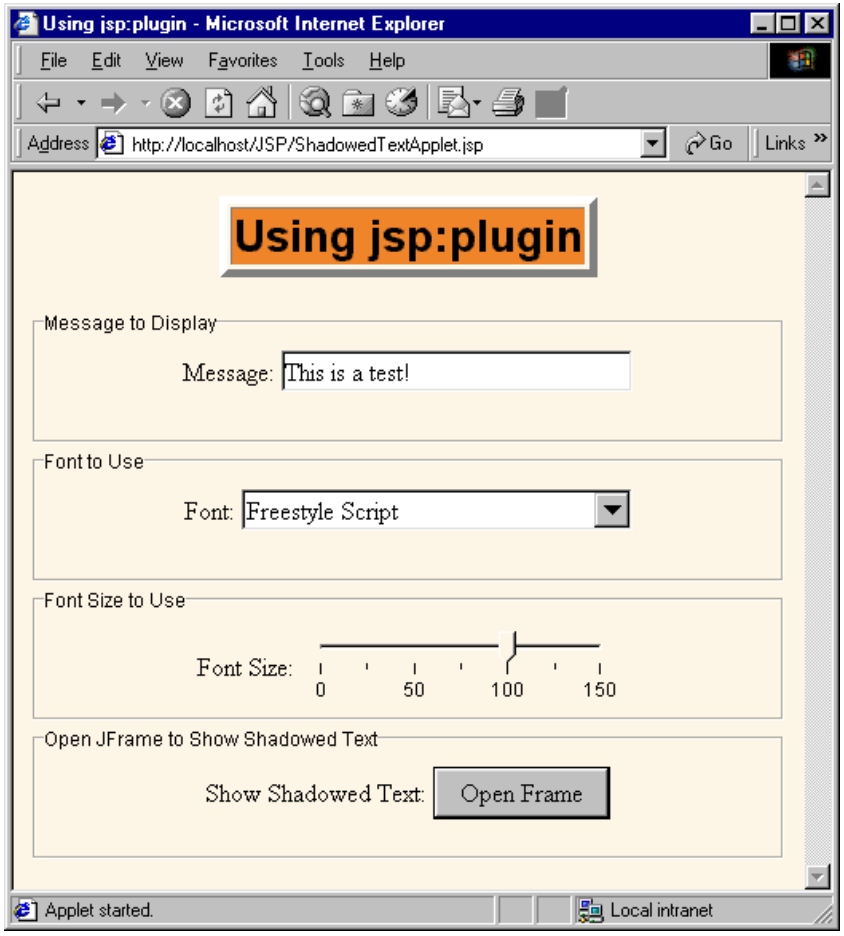
- **See book and Web site for applet code**

Example Using Plugin: Resultant HTML

```
<DOCTYPE ...>...
<OBJECT
  classid=clsid:8AD9C840-044E-11D1-B3E9-00805F499D93
  width="370" height="420"
   codebase="http://java.sun.com/products/plugin/1.2.2/jinstall-
   1_2_2-win.cab#Version=1,2,2,0">
<PARAM name="java_code" value="PluginApplet.class">
<PARAM name="type" value="application/x-java-applet;">
<COMMENT>
<EMBED type="application/x-java-applet;"
       width="370" height="420"
     pluginspage="http://java.sun.com/products/plugin/"
>
<NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>
...
</BODY></HTML>
```

Example Using Plugin (Continued)





Example Using Plugin (Continued)

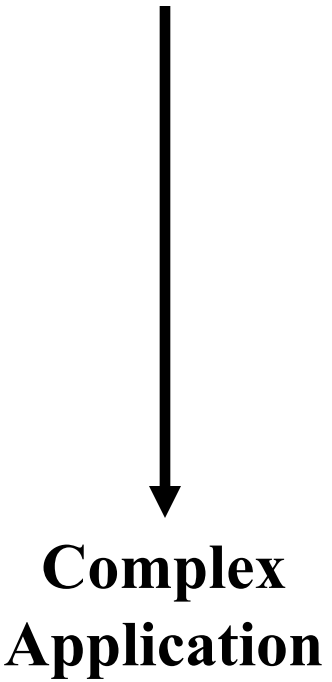
Summary

- **<jsp:include page="Relative URL" flush="true" />**
  - Output of URL inserted into JSP page at request time
  - Cannot contain JSP content that affects entire page
  - Changes to included file do not necessitate changes to pages that use it
- **<%@ include file="Relative URL" %>**
  - File gets inserted into JSP page prior to page translation
  - Thus, file can contain JSP content that affects entire page (e.g., import statements, declarations)
  - Changes to included file require you to manually update pages that use it
- **<jsp:plugin ...> simplifies applets for plugin**

Using JavaBeans with JSP

- Overview of beans in Java
- Basic use of beans in JSP
- Creating and accessing beans
- Setting bean properties explicitly
- Associating individual bean properties with request parameters
- Associating all bean properties with request parameters
- Conditional bean operations
- Sharing beans among multiple JSP pages and servlets

Uses of JSP Constructs



- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Custom tags
- Servlet/JSP combo (MVC), with beans and possibly custom tags

Background: What Are Beans?

- **Java classes that follow certain conventions**
  - Must have a zero-argument (empty) constructor
    - You can satisfy this requirement either by explicitly defining such a constructor or by omitting all constructors
  - Should have no public instance variables (fields)
    - I hope you already follow this practice and use accessor methods instead of allowing direct access to fields
  - Persistent values should be accessed through methods called get*Xxx* and set*Xxx*
    - If class has method getTitle that returns a String, class is said to have a String *property* named title
    - Boolean properties use is*Xxx* instead of get*Xxx*
  - For more on beans, see http://java.sun.com/beans/docs/

Why You Should Use Accessors, Not Public Fields

- To be a bean, you cannot have public fields
- So, you should replace

```
public double speed;
```

- with

```
private double a;

public double getSpeed() {
  return(a);
}
public void setSpeed(double newSpeed) {
  a = newSpeed;
}
```

- You should do this in *all* your Java code anyhow. Why?

Why You Should Use Accessors, Not Public Fields

- **1) You can put constraints on values**

```
public void setSpeed(double newSpeed) {
  if (newSpeed < 0) {
     sendErrorMessage(...);
     newSpeed = Math.abs(newSpeed);
   }
   speed = newSpeed;
}
```

  - If users of your class accessed the fields directly, then they would each be responsible for checking constraints.

Why You Should Use Accessors, Not Public Fields

- **2) You can change your internal representation without changing interface**

```
// Now using metric units (kph, not mph)

public void setSpeed(double newSpeed) {
  setSpeedInKPH = convert(newSpeed);
}

public void setSpeedInKPH(double newSpeed) {
  speedInKPH = newSpeed;
}
```

Why You Should Use Accessors, Not Public Fields

- **3) You can perform arbitrary side effects**

```
public double setSpeed(double newSpeed) {
  speed = newSpeed;
  updateSpeedometerDisplay();
}
```

  - If users of your class accessed the fields directly, then they would each be responsible for executing side effects. Too much work and runs huge risk of having display inconsistent from actual values.

Basic Bean Use in JSP

- **Format**
  - <jsp:useBean id="“my*name*" class="*package.myClass*" />
- **Purpose**
  - Allow instantiation of Java classes without explicit Java programming (XML-compatible syntax)
- **Notes**
  - Simple interpretation: JSP action
    <jsp:useBean id="book1" class="coreservlets.Book" />
    can be thought of as equivalent to the scriptlet
    <% coreservlets.Book book1 = new coreservlets.Book(); %>
  - But useBean has two additional advantages:
    - It is easier to derive object values from request parameters
    - It is easier to share objects among pages or servlets

Accessing Bean Properties

- **Format**
  - <jsp:getProperty name="*name*" property="*property*" />
- **Purpose**
  - Allow access to bean properties (i.e., calls to get*Xxx* methods) without explicit Java programming
- **Notes**
  - <jsp:getProperty name="book1" property="title" />
    is equivalent to the following JSP expression
    <%= book1.getTitle() %>

Setting Bean Properties: Simple Case

- **Format**
  - <jsp:setProperty name="*name*"
property="*property*"
value="*value*" />
- **Purpose**
  - Allow setting of bean properties (i.e., calls to set*Xxx* methods) without explicit Java programming
- **Notes**
  - <jsp:setProperty name="book1"
property="title"
value="Core Servlets and JavaServer Pages" />
is equivalent to the following scriptlet
<% book1.setTitle("Core Servlets and JavaServer Pages"); %>

Example: StringBean

```
package coreservlets;

public class StringBean {
  private String message = "No message specified";

  public String getMessage() {
    return(message);
  }

  public void setMessage(String message) {
    this.message = message;
  }
}
```

- **Installed in normal servlet directory**
  - *Tomcat_install_dir*\webapps\ROOT\WEB-INF\classes\coreservlets
  - *JRun_install_dir*\servers\default\default-app\WEB-INF\classes\coreservlets
- **Beans (and utility) classes must *always* be in packages!**

JSP Page That Uses StringBean

```
<jsp:useBean id="stringBean"
             class="coreservlets.StringBean" />
<OL>
<LI>Initial value (getProperty):
    <I><jsp:getProperty name="stringBean"
                        property="message" /></I>
<LI>Initial value (JSP expression):
    <I><%= stringBean.getMessage() %></I>
<LI><jsp:setProperty name="stringBean"
                     property="message"
                     value="Best string bean: Fortex" />
    Value after setting property with setProperty:
    <I><jsp:getProperty name="stringBean"
                        property="message" /></I>
<LI><%stringBean.setMessage("My favorite: Kentucky Wonder");%>
    Value after setting property with scriptlet:
    <I><%= stringBean.getMessage() %></I>
</OL>
```

JSP Page That Uses StringBean

Setting Bean Properties Case 1: Explicit Conversion & Assignment

```
<!DOCTYPE ...>
...
<jsp:useBean id="entry"
             class="coreservlets.SaleEntry" />

<%-- setItemID expects a String --%>
<jsp:setProperty
    name="entry"
    property="itemID"
    value='<%= request.getParameter("itemID") %>' />
```

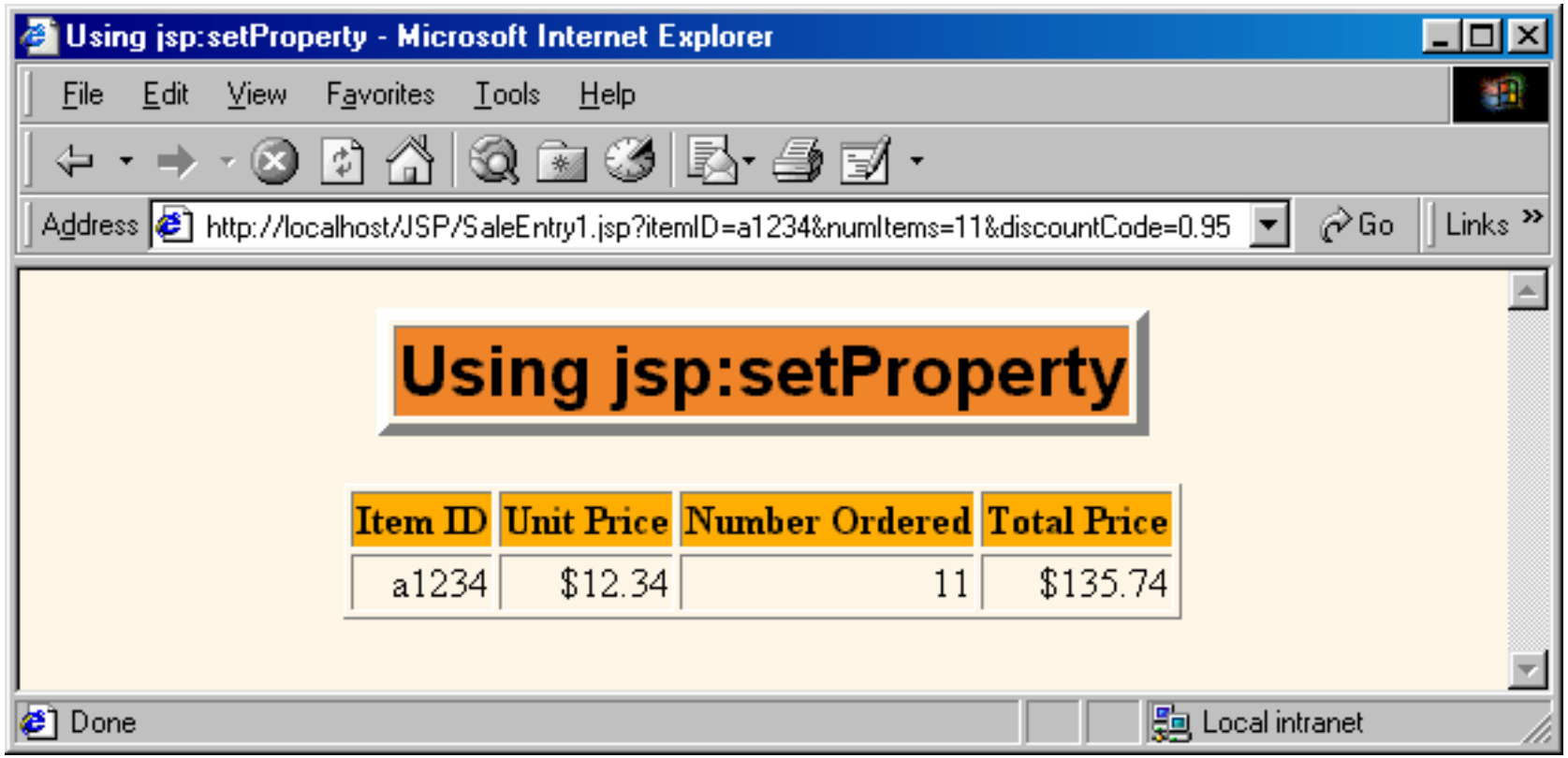
Setting Bean Properties Case 1: Explicit Conversion & Assignment

```
<%
int numItemsOrdered = 1;
try {
  numItemsOrdered =
    Integer.parseInt(request.getParameter("numItems"));
} catch(NumberFormatException nfe) {}
%>
<%-- setNumItems expects an int --%>
<jsp:setProperty
    name="entry"
    property="numItems"
    value="<%= numItemsOrdered %>" />
```

Setting Bean Properties Case 1: Explicit Conversion & Assignment

```
<%
double discountCode = 1.0;
try {
  String discountString =
    request.getParameter("discountCode");
  discountCode =
    Double.parseDouble(discountString);
} catch(NumberFormatException nfe) {}
%>
<%-- setDiscountCode expects a double --%>
<jsp:setProperty
    name="entry"
    property="discountCode"
    value="<%= discountCode %>" />
```

Setting Bean Properties Case 1: Explicit Conversion & Assignment



Case 2: Associating Individual Properties with Input Parameters

- **Use the param attribute of jsp:setProperty to indicate that**
  - Value should come from specified request parameter
  - Simple automatic type conversion should be performed for properties that expect values of standard types
    - boolean, Boolean, byte, Byte, char, Character, double, Double, int, Integer, float, Float, long, or Long.

Case 2: Associating Individual Properties with Input Parameters

```
<jsp:useBean id="entry"
             class="coreservlets.SaleEntry" />
<jsp:setProperty
    name="entry"
    property="itemID"
    param="itemID" />
<jsp:setProperty
    name="entry"
    property="numItems"
    param="numItems" />
<jsp:setProperty
    name="entry"
    property="discountCode"
    param="discountCode" />
```

Case 3: Associating All Properties with Input Parameters

- **Use "*" for the value of the property attribute of jsp:setProperty to indicate that**
  - Value should come from request parameter whose name matches property name
  - Simple automatic type conversion should be performed

Case 3: Associating All Properties with Input Parameters

```
<jsp:useBean id="entry"
             class="coreservlets.SaleEntry" />
<jsp:setProperty name="entry" property="*" />
```

- **This is extremely convenient for making "form beans" -- objects whose properties are filled in from a form submission.**
  - You can even divide the process up across multiple forms, where each submission fills in part of the object.

Sharing Beans

- **You can use scope attribute to specify additional places where bean is stored**
  - Still also bound to local variable in _jspService
  - **<jsp:useBean id="…" class="…" scope="…" />**
- **Lets multiple servlets or JSP pages share data**
- **Also permits conditional bean creation**
  - Create new object only if you can't find existing one

Values of the scope Attribute

- **page (<jsp:useBean … scope="page"/> or <jsp:useBean…>)**
  - Default value. Bean object should be placed in the PageContext object for the duration of the current request. Lets methods in same servlet access bean
- **application (<jsp:useBean … scope="application"/>)**
  - Bean will be stored in ServletContext (available through the application variable or by call to getServletContext()). ServletContext is shared by all servlets in the same Web application (or all servlets on server if no explicit Web applications are defined).

Values of the scope Attribute

- **session (<jsp:useBean … scope="session"/>)**
  - Bean will be stored in the HttpSession object associated with the current request, where it can be accessed from regular servlet code with getAttribute and setAttribute, as with normal session objects.
- **request (<jsp:useBean … scope="request"/>)**
  - Bean object should be placed in the ServletRequest object for the duration of the current request, where it is available by means of getAttribute

Conditional Bean Operations

- **Bean conditionally created**
  - jsp:useBean results in new bean being instantiated only if no bean with same id and scope can be found.
  - If a bean with same id and scope is found, the preexisting bean is simply bound to variable referenced by id.
- **Bean properties conditionally set**
  - <jsp:useBean ... /> replaced by <jsp:useBean ...>statements</jsp:useBean>
  - The statements (jsp:setProperty elements) are executed *only* if a new bean is created, not if an existing bean is found.

Conditional Bean Creation: AccessCountBean

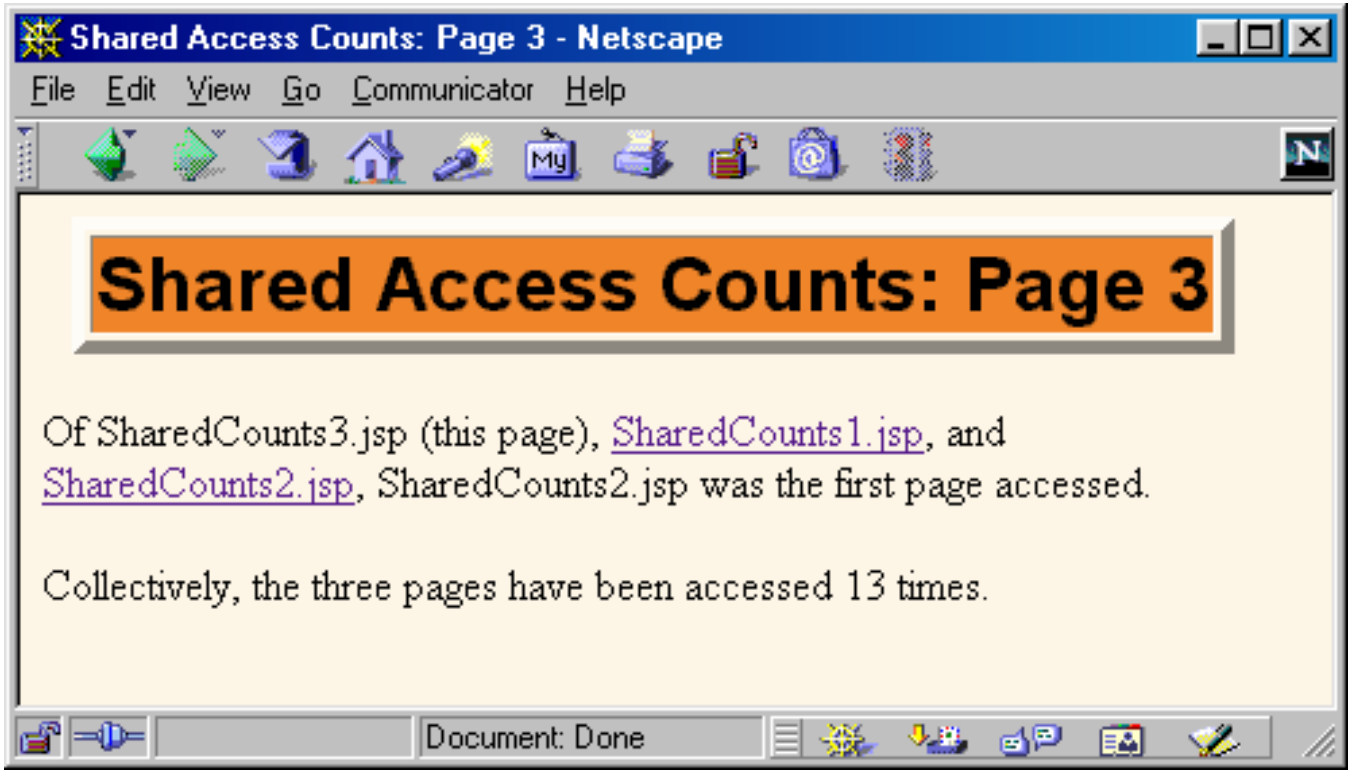
```
public class AccessCountBean {
  private String firstPage;
  private int accessCount = 1;

  public String getFirstPage() {
    return(firstPage);
  }

  public void setFirstPage(String firstPage) {
    this.firstPage = firstPage;
  }

  public int getAccessCount() {
    return(accessCount);
  }

  public void setAccessCountIncrement(int increment) {
    accessCount = accessCount + increment;
  }
}
```

Conditional Bean Creation: SharedCounts1.jsp

```
<jsp:useBean id="counter"
             class="coreservlets.AccessCountBean"
             scope="application">
  <jsp:setProperty name="counter"
                   property="firstPage"
                   value="SharedCounts1.jsp" />
</jsp:useBean>
Of SharedCounts1.jsp (this page),
<A HREF="SharedCounts2.jsp">SharedCounts2.jsp</A>, and
<A HREF="SharedCounts3.jsp">SharedCounts3.jsp</A>,
<jsp:getProperty name="counter" property="firstPage" />
was the first page accessed.
<P>
Collectively, the three pages have been accessed
<jsp:getProperty name="counter" property="accessCount" />
times.
<jsp:setProperty name="counter"
                 property="accessCountIncrement"
                 value="1" />
```

Accessing SharedCounts1, SharedCounts2, SharedCounts3

- SharedCounts2.jsp was accessed first.
- Pages have been accessed twelve previous times by an arbitrary number of clients



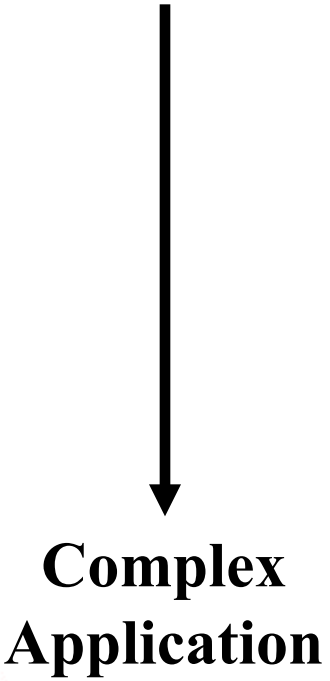
Summary

- **Benefits of jsp:useBean**
  - Hides the Java syntax
  - Makes it easier to associate request parameters with Java objects (bean properties)
  - Simplifies sharing objects among multiple requests or servlets/JSPs
- **jsp:useBean**
  - Creates or accesses a bean
- **jsp:getProperty**
  - Puts bean property (i.e. get*Xxx* call) into servlet output
- **jsp:setProperty**
  - Sets bean property (i.e. passes value to set*Xxx*)

Integrating Servlets and JSP: The MVC Architecture

- Reasons to combine servlets and JSP
- Approach to integration
- Dispatching requests
- Storing data for later retrieval
- Example 1:
  an on-line travel agent
- Example 2:
  an on-line boat store
- Including requests:
  showing raw servlet and JSP output

Uses of JSP Constructs



- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Custom tags
- Servlet/JSP combo (MVC), with beans and possibly custom tags

Why Combine Servlets & JSP?

- **Typical picture: use JSP to make it easier to develop and maintain the HTML content**
  - For simple dynamic code, call servlet code from scripting elements
  - For slightly more complex applications, use custom classes called from scripting elements
  - For moderately complex applications, use beans and custom tags
- **But, that's not enough**
  - For complex processing, starting with JSP is awkward
  - Despite the ease of separating the real code into separate classes, beans, and custom tags, the assumption behind JSP is that a *single* page gives a *single* basic look

Possibilities for Handling a Single Request

- **Servlet only**
  - Output is a binary type. E.g.: an image
  - No output. E.g.: you are doing forwarding or redirection as in Search Engine example.
  - Format/layout of page is highly variable. E.g.: portal.
- **JSP only**
  - Output is mostly character data. E.g.: HTML
  - Format/layout mostly fixed.
- **Combination**
  - A single request will result in multiple substantially different-looking results.
  - Complicated data processing, but relatively fixed layout.
- **These apply to a *single* request**
  - You still use both servlets and JSP within your *overall* application.

Approach

- **Joint servlet/JSP process:**
  - Original request is answered by a servlet
  - Servlet processes request data, does database lookup, business logic, etc.
  - Results are placed in beans
  - Request is forwarded to a JSP page to format result
  - Different JSP pages can be used to handle different types of presentation
- **Often called the "MVC" (Model View Controller) or "Model 2" approach to JSP**
- **Formalized in Apache Struts Framework**
  - http://jakarta.apache.org/struts/

Implementing MVC

- **The important thing is the idea**
  - Syntax not complicated
- **We already know**
  - How to extract previously-stored data in a JSP page
    - Use jsp:useBean with the scope attribute
- **Two pieces of syntax we don't yet know**
  - How does a servlet invoke a JSP page?
  - How does a servlet store data where it can be retrieved by
    - jsp:useBean with scope="request"
    - jsp:useBean with scope="session"
    - jsp:useBean with scope="application"

Dispatching Requests from Servlets to JSP Pages

- **First, call the getRequestDispatcher method of ServletContext**
  - Supply URL relative to server or Web application root
  - Example
    - String url = "/presentations/presentation1.jsp";
      RequestDispatcher dispatcher =
        getServletContext().getRequestDispatcher(url);
- **Second**
  - Call forward to completely transfer control to destination page (no communication with client in between, as there is with response.sendRedirect)
    - This is the normal approach with MVC
  - Call include to insert output of destination page and then continue on

Forwarding Requests: Example Code

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
  throws ServletException, IOException {
String operation = request.getParameter("submit");
if (operation == null) {
  operation = "unknown";
}
if (operation.equals("Add")) {
  gotoPage("/operations/Add.jsp",
           request, response);
} else if (operation.equals("Remove")) {
  gotoPage("/operations/Remove.jsp",
           request, response);
} else {
  gotoPage("/operations/unknownRequestHandler.jsp",
           request, response);
}
}
```

Forwarding Requests: Example Code (Continued)

```
private void gotoPage(String address,
                      HttpServletRequest request,
                      HttpServletResponse response)
    throws ServletException, IOException {
  RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher(address);
  dispatcher.forward(request, response);
}
```

Reminder: JSP useBean Scope Alternatives

- **request**
  - <jsp:useBean id="..." class="..." scope="request" />
- **session**
  - <jsp:useBean id="..." class="..." scope="session" />
- **application**
  - <jsp:useBean id="..." class="..." scope="application" />
- **page**
  - <jsp:useBean id="..." class="..." scope="page" />
    or just
    <jsp:useBean id="..." class="..." />
  - This scope is not used in MVC (Model 2) architecture

Storing Data for Later Use: The Servlet Request

- **Purpose**
  - Storing data that servlet looked up and that JSP page will use only in this request.
- **Servlet syntax to store data**

```
SomeClass value = new SomeClass(...);
request.setAttribute("key", value);
// Use RequestDispatcher to forward to JSP
```

- **JSP syntax to retrieve data**

```
<jsp:useBean
   id="key"
   class="somepackage.SomeClass"
   scope="request" />
```

Storing Data for Later Use: The Servlet Request (Variation)

- **Purpose**
  - Storing data that servlet looked up and that JSP page will use only in this request.
- **Servlet syntax to store data**
  - Add new request parameters to servlet request

```
String address ="/path/resource.jsp?newParam=value";
RequestDispatcher dispatcher =
  getServletContext().getRequestDispatcher(address);
dispatcher.forward(request, response);
```

- **JSP syntax to retrieve data**
  - No useBean syntax. However, recall that request parameters can be accessed without explicit Java code by means of jsp:setProperty.

Storing Data for Later Use: The Session Object

- **Purpose**
  - Storing data that servlet looked up and that JSP page will use in this request and in later requests from same client.
- **Servlet syntax to store data**

```
SomeClass value = new SomeClass(...);
HttpSession session =
  request.getSession(true);
session.setAttribute("key", value);
// Use RequestDispatcher to forward to JSP
```

- **JSP syntax to retrieve data**

```
<jsp:useBean
   id="key"
   class="somepackage.SomeClass"
   scope="session" />
```

Variation for Session Tracking

- **Use response.sendRedirect instead of RequestDispatcher.forward**
- **Distinctions: with sendRedirect:**
  - User sees JSP URL (user sees only servlet URL with RequestDispatcher.forward)
  - Two round trips to client (only one with forward)
- **Advantage of sendRedirect**
  - User can visit JSP page separately
    - User can bookmark JSP page
- **Disadvantage of sendRedirect**
  - Since user can visit JSP page without going through servlet first, JSP data might not be available
    - So, JSP page needs code to detect this situation

Storing Data for Later Use: The Servlet Context

- **Purpose**
  - Storing data that servlet looked up and that JSP page will use in this request and in later requests from *any* client.
- **Servlet syntax to store data**

```
synchronized(this) {
  SomeClass value = new SomeClass(…);
  getServletContext().setAttribute("key",
                                   value);
  // RequestDispatcher forwards to JSP
}
```

- **JSP syntax to retrieve data**

```
<jsp:useBean
   id="key"
   class="somepackage.SomeClass"
   scope="application" />
```

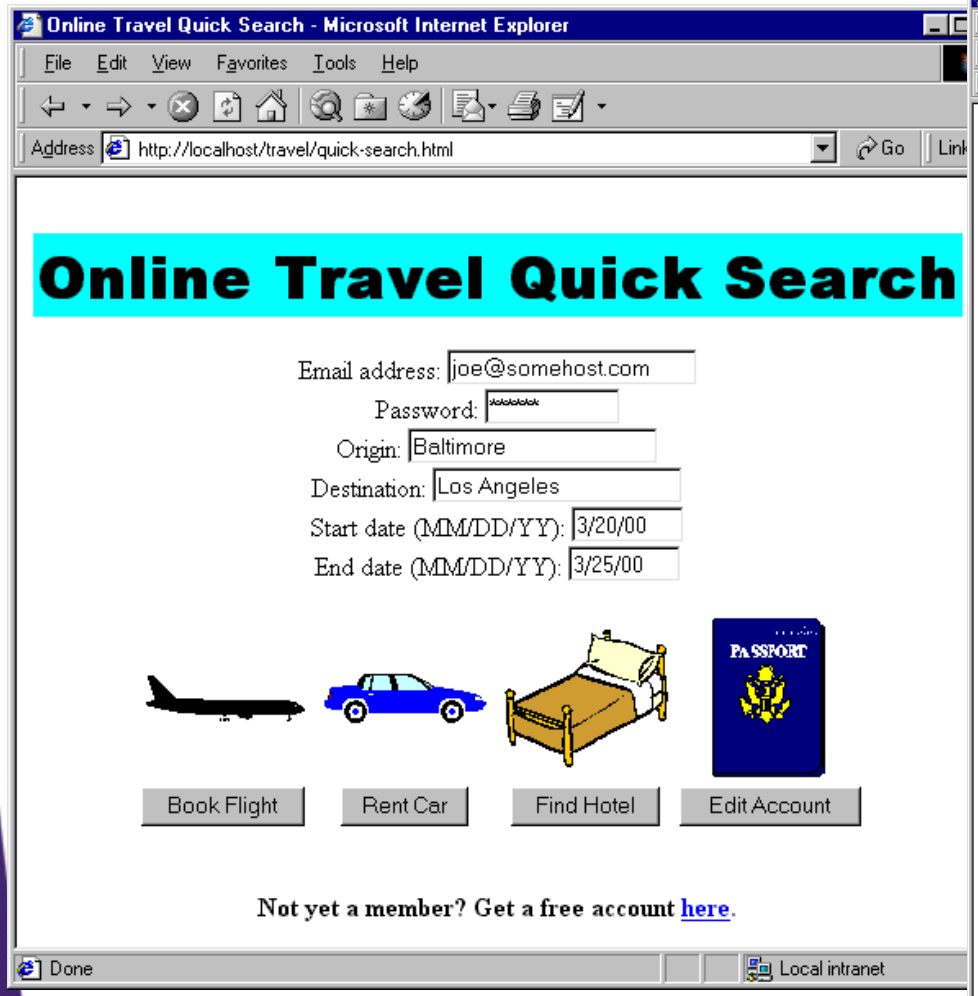
Relative URLs in JSP Pages

- **Issue:**
  - Forwarding with a request dispatcher is transparent to the client. *Original* URL is only URL browser knows about.
- **Why does this matter?**
  - What will browser do with tags like the following:

```
<IMG SRC="foo.gif" …>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
<A HREF="bar.jsp">…</A>
```

  - Answer: browser treats them as relative to *servlet URL*
- **Simplest solution:**
  - Use URLs that begin with a slash

MVC Example 1: An On-Line Travel Agent



MVC Example 1: An On-Line Travel Agent

- **All requests include**
  - Email address, password, trip origin, trip destination, start date, and end date
- **Original request answered by servlet**
  - Looks up real name, address, credit card information, frequent flyer data, etc., using email address and password as key. *Data stored in session object.*
- **Depending on what button user pressed, request forwarded to:**
  - Page showing available flights, times, and costs
  - Page showing available hotels, features, and costs
  - Rental car info, edit customer data, error handler

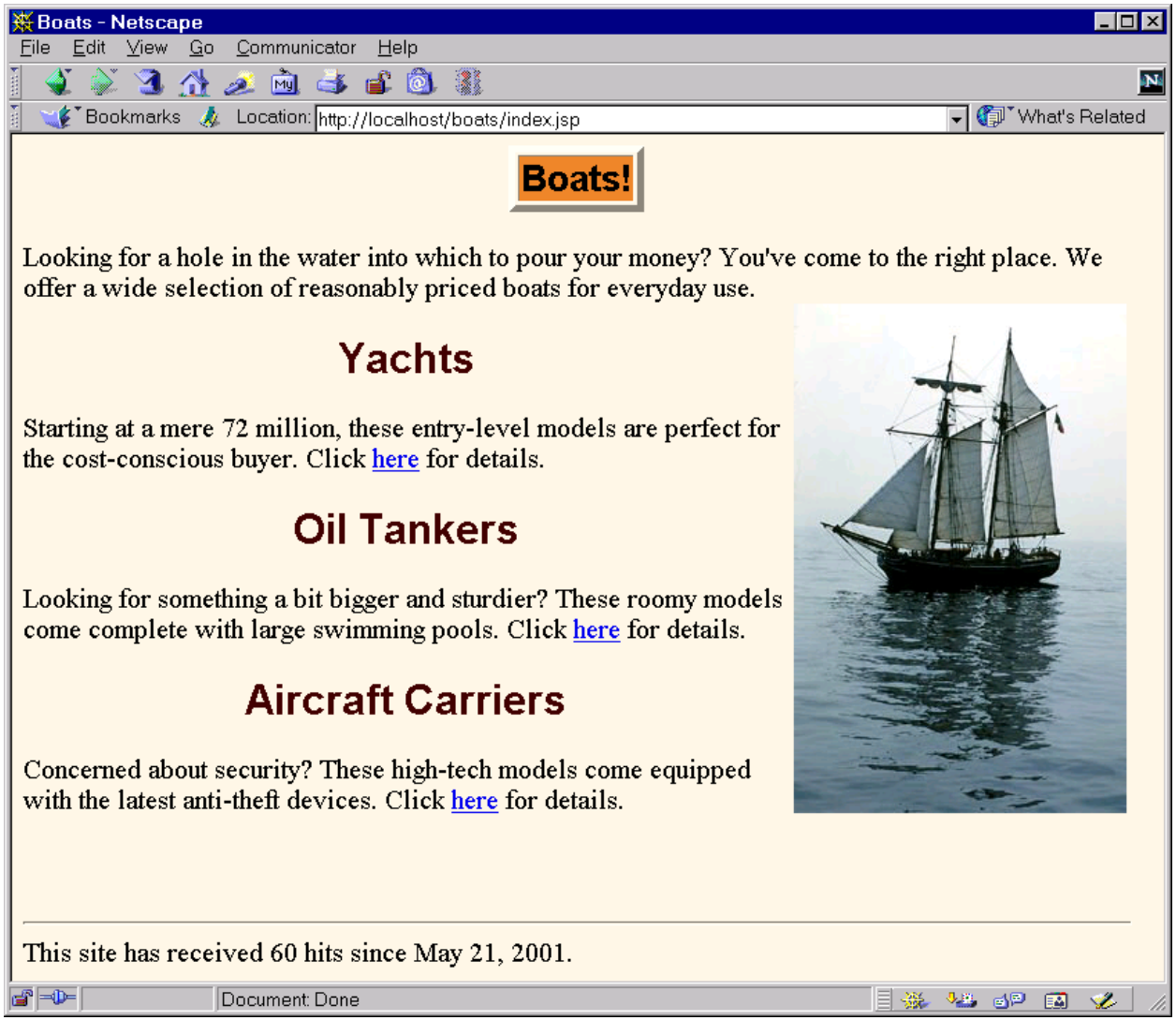
An On-Line Travel Agent: Servlet Code

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
  ...// Store data in TravelCustomer bean called "customer"
  HttpSession session = request.getSession(true);
  session.setAttribute("customer", customer);
  if (request.getParameter("flights") != null) {
    gotoPage("/travel/BookFlights.jsp",
             request, response);
  } else if ...
}
private void gotoPage(String address,
                      HttpServletRequest request,
                      HttpServletResponse response)
    throws ServletException, IOException {
  RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher(address);
  dispatcher.forward(request, response);
}
```

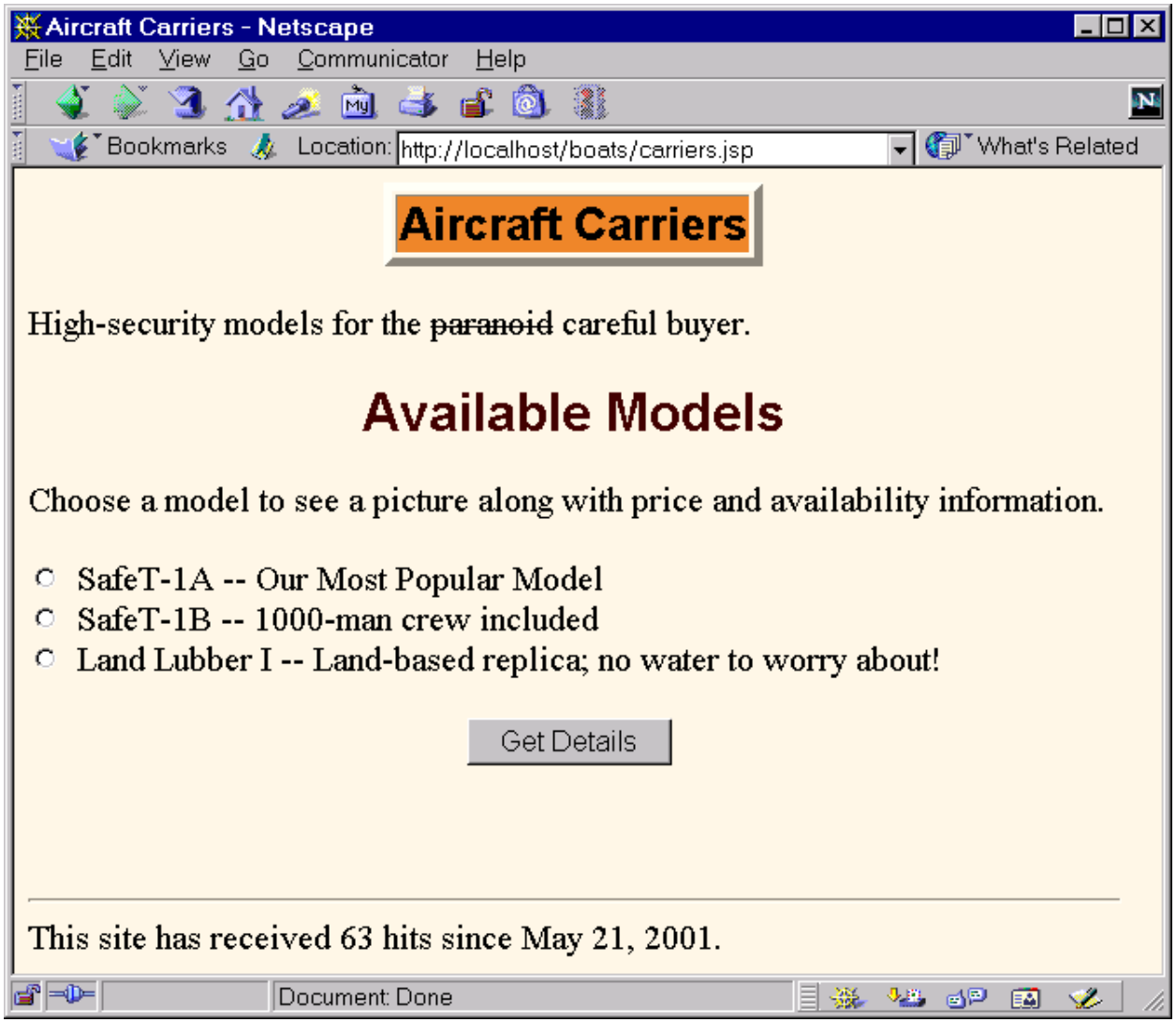
An On-Line Travel Agent: JSP Code (Flight Page)

```
<BODY>
<H1>Best Available Flights</H1>
<CENTER>
<jsp:useBean id="customer"
             class="coreservlets.TravelCustomer"
             scope="session" />
Finding flights for
<jsp:getProperty name="customer"
                 property="fullName" />
<P>
<jsp:getProperty name="customer" property="flights" />
...
```

MVC Example 2: An Online Boat Store



MVC Example 2: An Online Boat Store



MVC Example 2: An Online Boat Store

SafeT-1A - Netscape

File Edit View Go Communicator Help

Bookmarks Location: http://localhost/boats/DisplayItem?itemNum=SafeT-1A What's Related

SafeT-1A

Item Number

SafeT-1A

Description

A safe and secure boat, perfect for family vacations. **Note:** no crew provided. If crew is desired, please see model SafeT-1B.

Cost

$3,167,492,481.99. A real bargain!

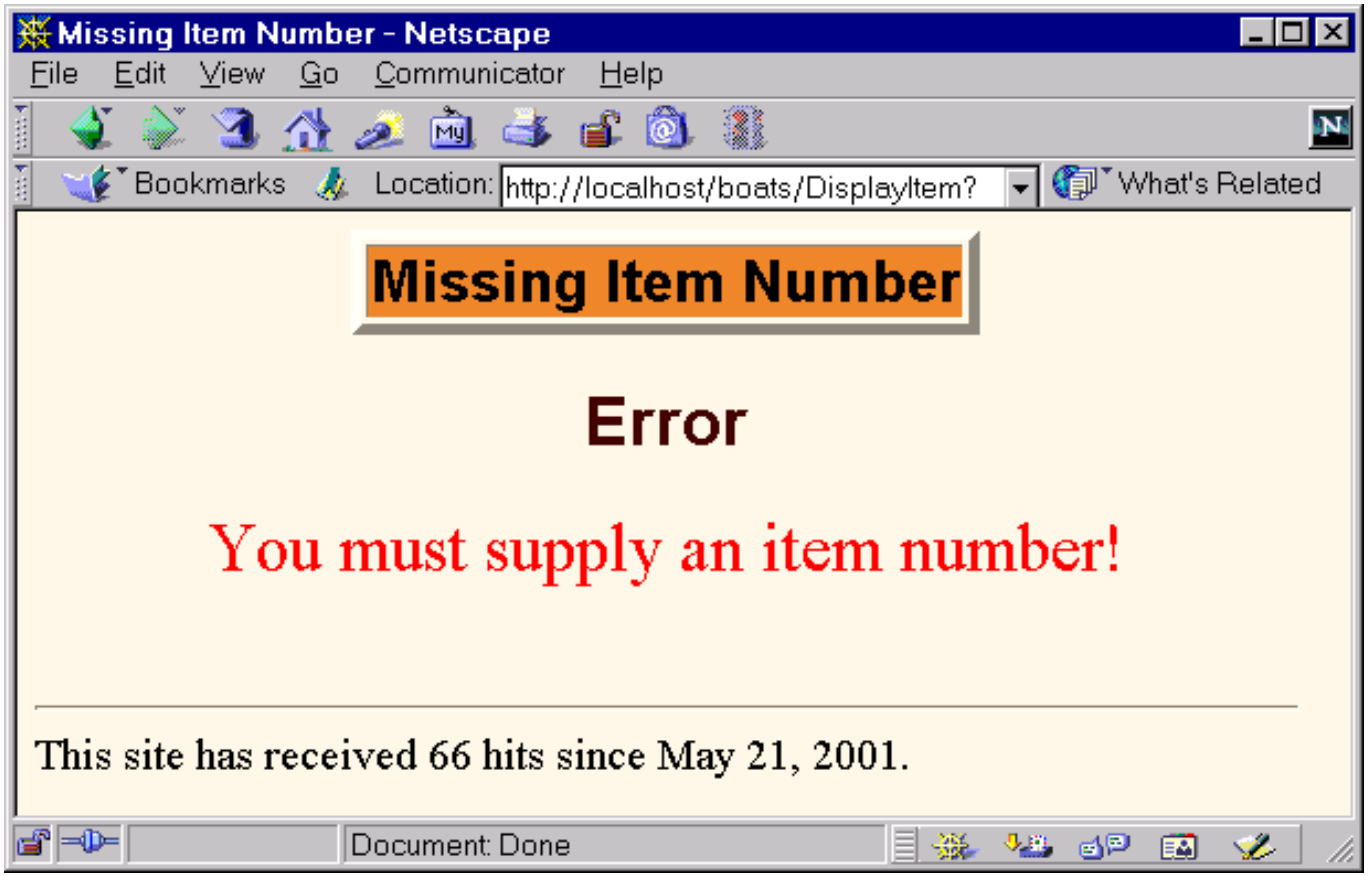
Ordering

Submit Order

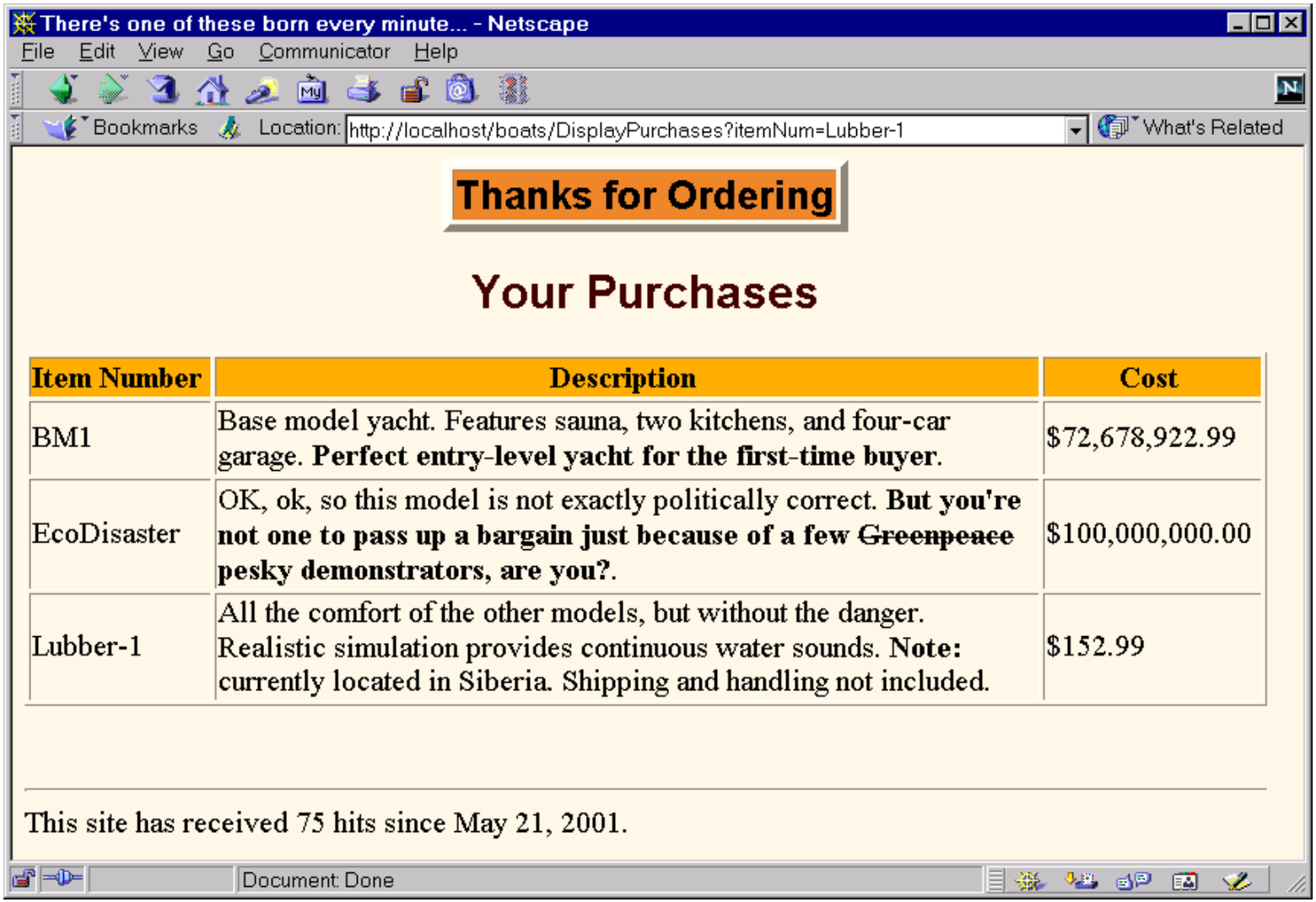
This site has received 64 hits since May 21, 2001.

Document: Done

MVC Example 2: An Online Boat Store



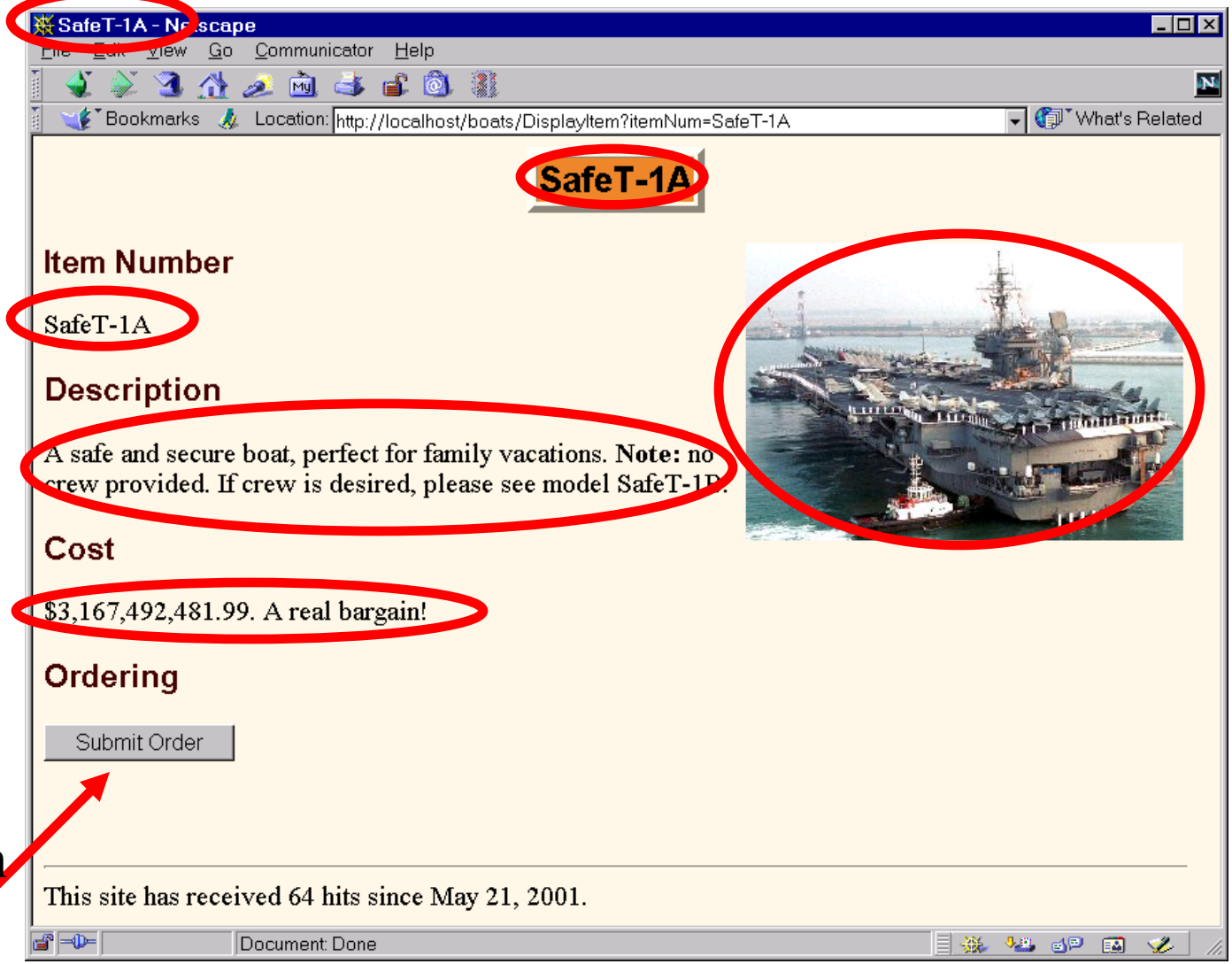
MVC Example 2: An Online Boat Store



MVC Example 2: Servlet Code

```
public class ShowItem extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    String itemNum = request.getParameter("itemNum");
    String destination;
    if (itemNum == null) {
      destination = "/MissingItem.jsp";
    } else {
      destination = "/ShowItem.jsp";
      ItemTable shipTable = ShipTable.getShipTable();
      SimpleItem item = shipTable.getItem(itemNum);
      request.setAttribute("item", item);
    }
    RequestDispatcher dispatcher =
      getServletContext().getRequestDispatcher(destination);
    dispatcher.forward(request, response);
  }
}
```

MVC Example 2: An Online Boat Store



Hidden Field

MVC Example 2: JSP Code (ShowItem.jsp)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
…
<jsp:useBean id="item"
             class="moreservlets.SimpleItem"
             scope="request" />
<TITLE><jsp:getProperty name="item" property="itemNum" />
</TITLE>
…
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
  <jsp:getProperty name="item" property="itemNum" /></TABLE>
<P>
<IMG SRC="<jsp:getProperty name='item' property='imageURL' />"
     ALIGN="RIGHT">

<H3>Item Number</H2>
<jsp:getProperty name="item" property="itemNum" />
<H3>Description</H2>
<jsp:getProperty name="item" property="description" />
```

MVC Example 2: JSP Code (ShowItem.jsp Cont.)

```
<H3>Cost</H2>
<jsp:getProperty name="item" property="costString" />.
A real bargain!

<H3>Ordering</H2>
<FORM ACTION="DisplayPurchases">
  <INPUT TYPE="HIDDEN" NAME="itemNum"
         VALUE="<jsp:getProperty name='item'
                                 property='itemNum' />">
  <INPUT TYPE="SUBMIT" VALUE="Submit Order">
</FORM>

<%@ taglib uri="/WEB-INF/tlds/count-taglib.tld"
           prefix="boats" %>
<boats:count />
</BODY>
</HTML>
```

MVC Example 2: Bean Code (SimpleItem.java)

```
public class SimpleItem {
  private String itemNum = "Missing item number";
  private String description = "Missing description";
  private String imageURL = "Missing image URL";
  private double cost;
  private NumberFormat formatter =
    NumberFormat.getCurrencyInstance();

  public SimpleItem(String itemNum,
                    String description,
                    String imageURL,
                    double cost) {
    setItemNum(itemNum);
    setDescription(description);
    setImageURL(imageURL);
    setCost(cost);
  }

  public SimpleItem() {}

...
}
```

Forwarding Requests from JSP Pages -- jsp:forward

- You usually forward from a servlet to a JSP page, but you can also forward from JSP

```
<% String destination;
   if (Math.random() > 0.5) {
     destination = "/examples/page1.jsp";
   } else {
     destination = "/examples/page2.jsp";
   }
%>
<jsp:forward page="<%= destination %>" />
```

- Question: can you forward from a servlet to another servlet? How do you know?

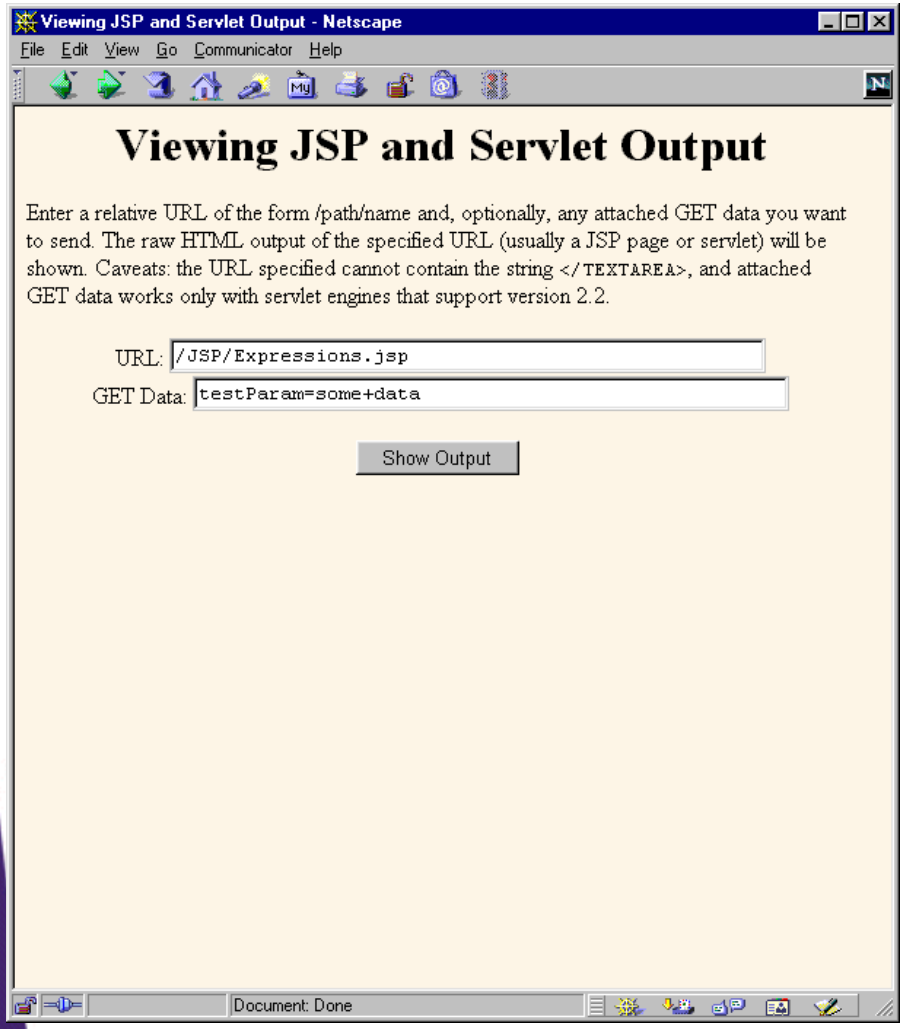
Including Pages Instead of Forwarding to Them

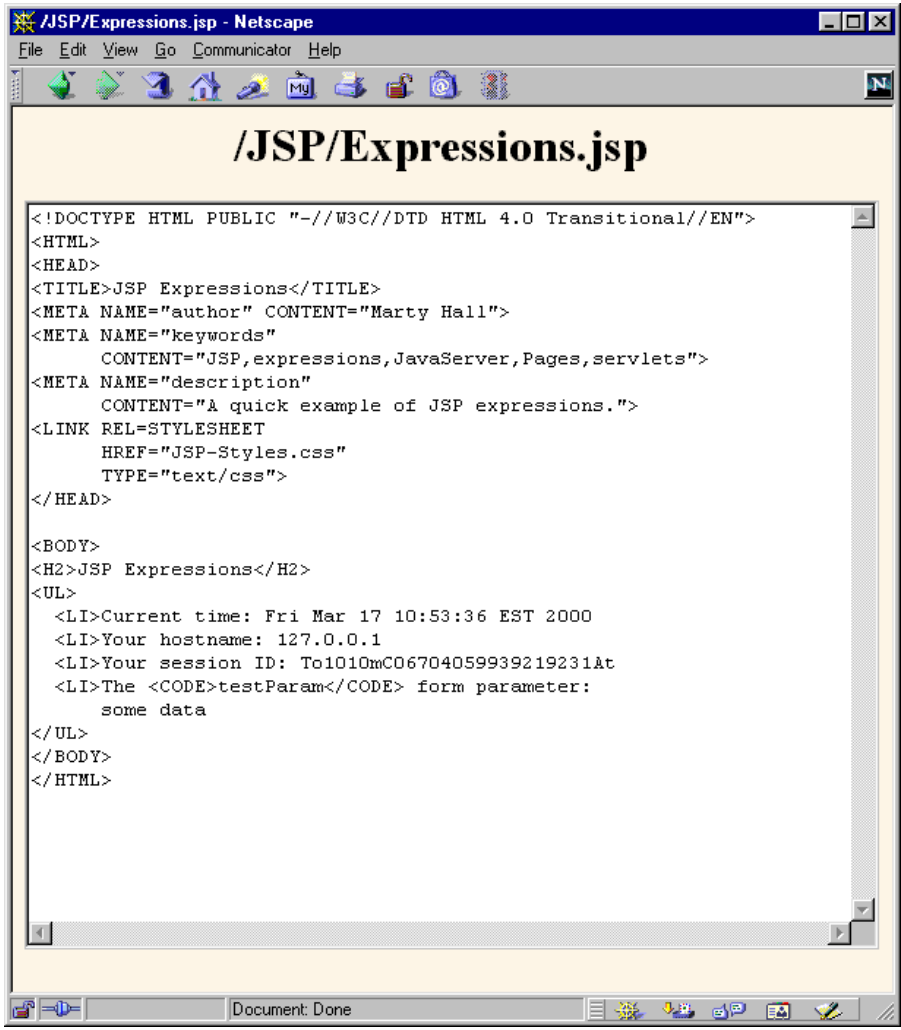
- **With the `forward` method of RequestDispatcher:**
  - Control is *permanently* transferred to new page
  - Original page *cannot* generate any output
- **With the `include` method of RequestDispatcher:**
  - Control is *temporarily* transferred to new page
  - Original page *can* generate output before and after the included page
  - Original servlet does not see the output of the included page (for this, see later topic on servlet/JSP filters)
  - Useful for portals: JSP presents pieces, but pieces arranged in different orders for different users

A Servlet that Shows Raw Servlet and JSP Output

```
out.println(...
            "<TEXTAREA ROWS=30 COLS=70>");
if ((url == null) || (url.length() == 0)) {
  out.println("No URL specified.");
} else {
  // Attaching data works only in version 2.2.
  String data = request.getParameter("data");
  if ((data != null) && (data.length() > 0)) {
    url = url + "?" + data;
  }
  RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher(url);
  dispatcher.include(request, response);
}
out.println("</TEXTAREA>\n" +
            ...);
```

A Servlet that Shows Raw Servlet and JSP Output





Summary

- **Use MVC (Model 2) approach when:**
  - One submission will result in more than one basic look
  - Several pages have substantial common processing
- **Architecture**
  - A servlet answers the original request
  - Servlet does the real processing & stores results in beans
    - Beans stored in HttpServletRequest, HttpSession, or ServletContext
  - Servlet forwards to JSP page via forward method of RequestDispatcher
  - JSP page reads data from beans by means of jsp:useBean with appropriate scope (request, session, or application)