J2EE-Frameworks J2EE

ING2 EISTI

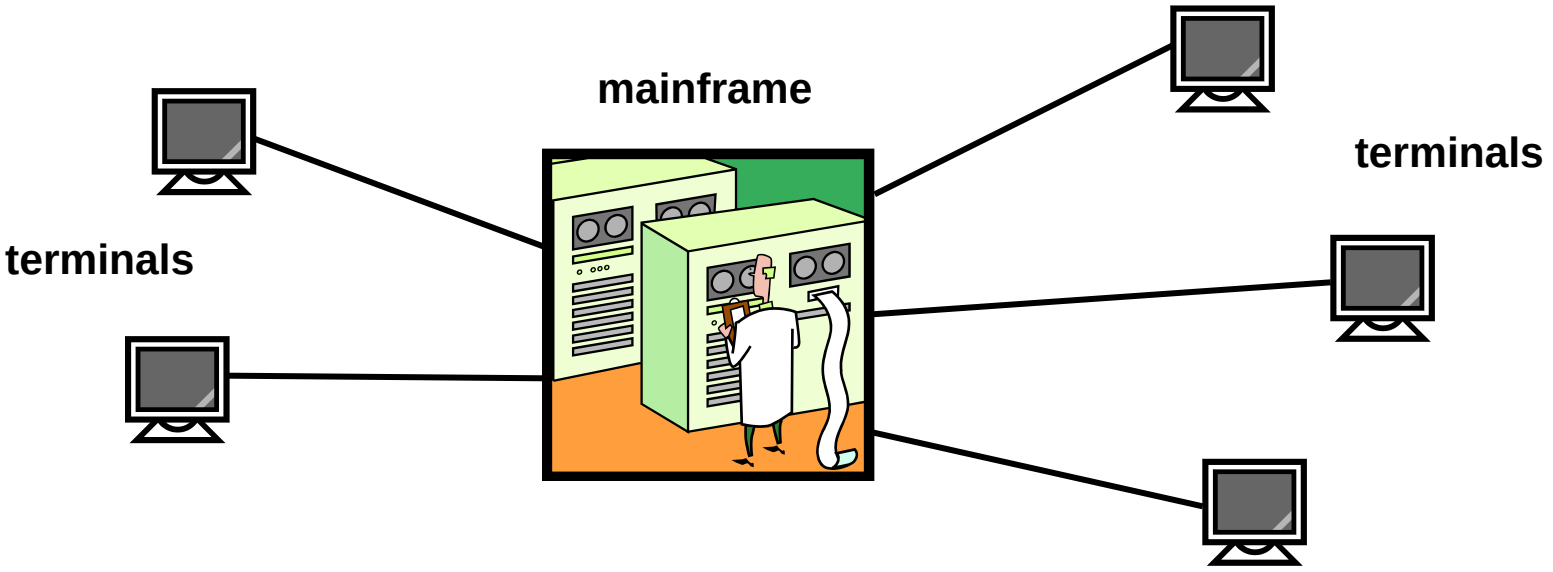
2015-2016

Course Content

1. Application servers
2. What is J2EE?
   - Main component types
   - J2EE APIs and Services
3. Install Tomcat server
4. Tomcat Hierarchy

1. Application Servers

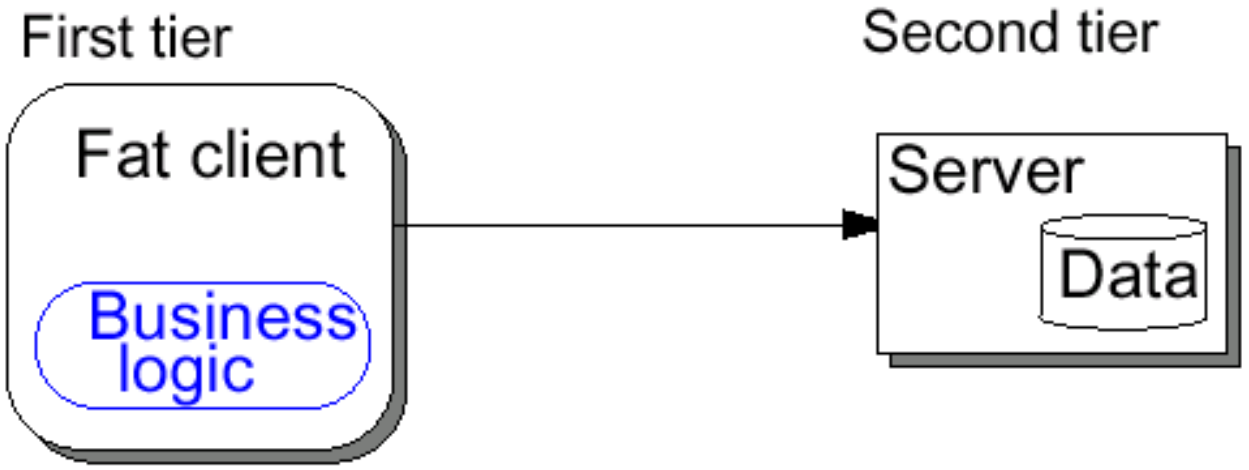
- *In the beginning, there was darkness and cold. Then, …*



Centralized, non-distributed

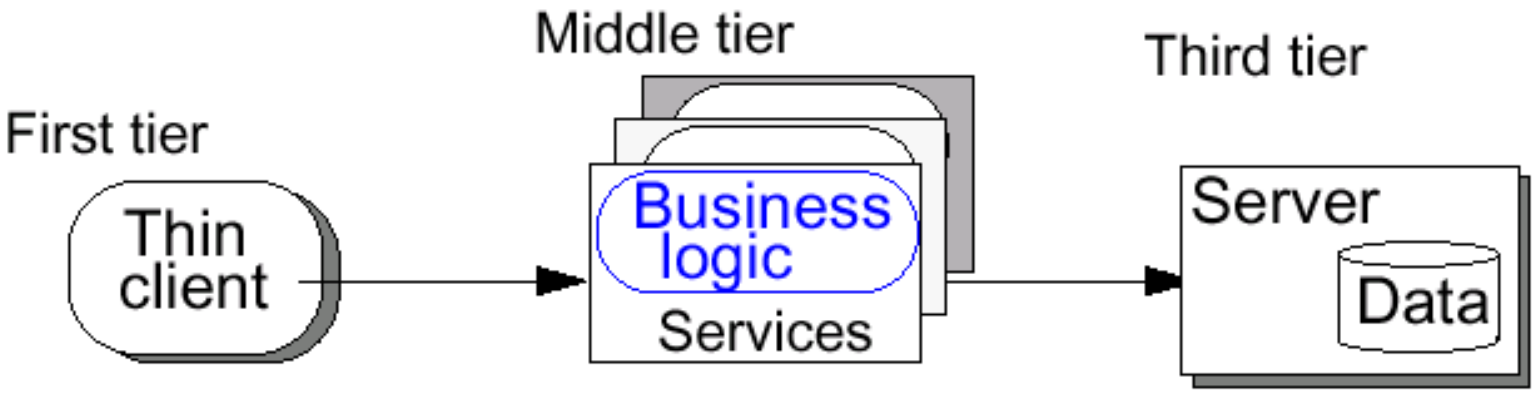
Application Servers

- In the 90's, systems should be *client-server*



Application Servers

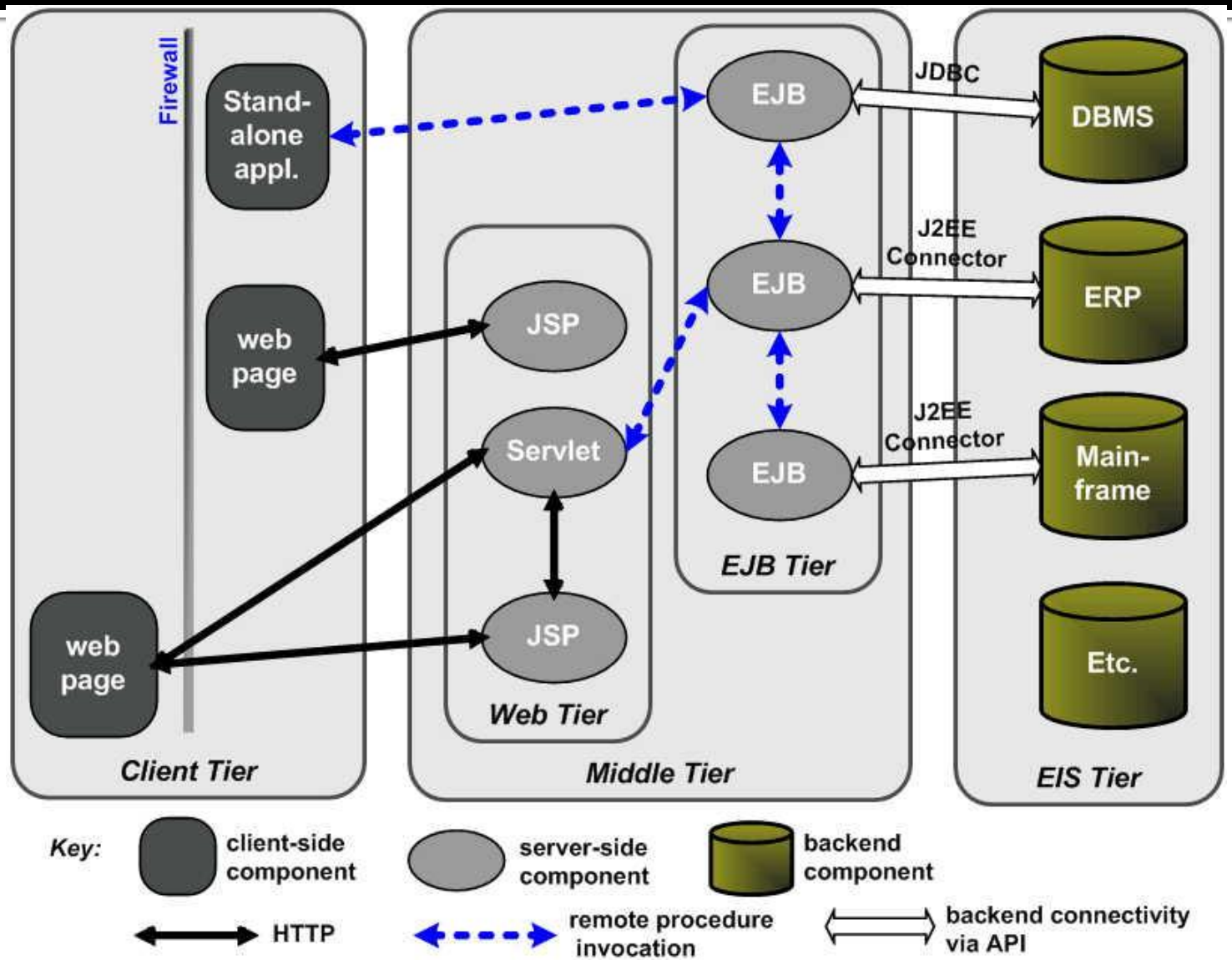
- Today, enterprise applications use the *multi-tier* model



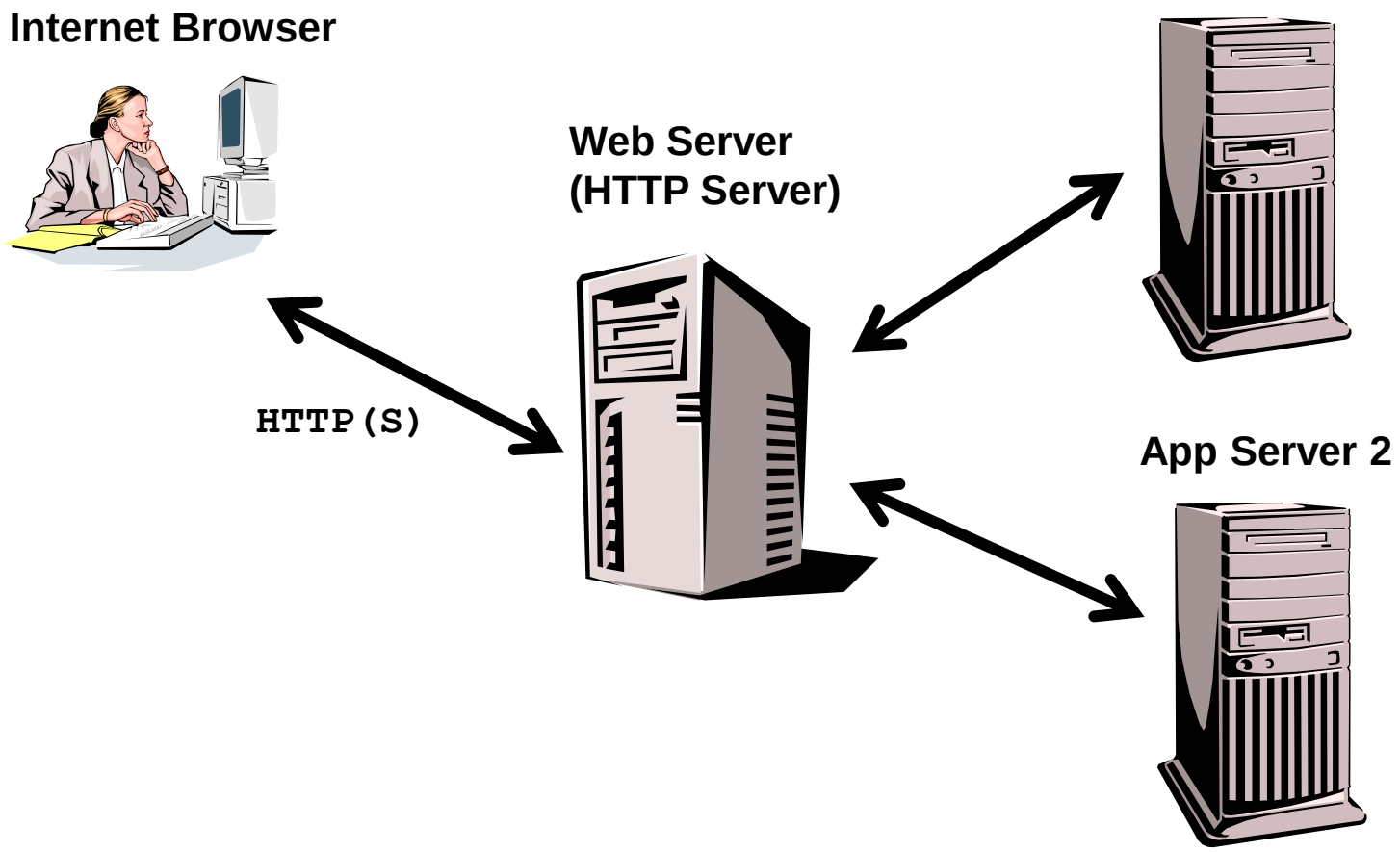
Application Servers

- "Multi-tier applications" have several independent components
- An *application server* provides the infrastructure and services to run such applications

J2EE Multi-tier Model



Web Server and Application Server



2. What is J2EE?

- **J2EE is an architecture for implementing enterprise class applications using Java and Internet Technology**
- **Solves problems of two tier architecture**
- It is a public specification that embodies several technologies
- J2EE defines a model for developing multi-tier, web based, enterprise applications with distributed components

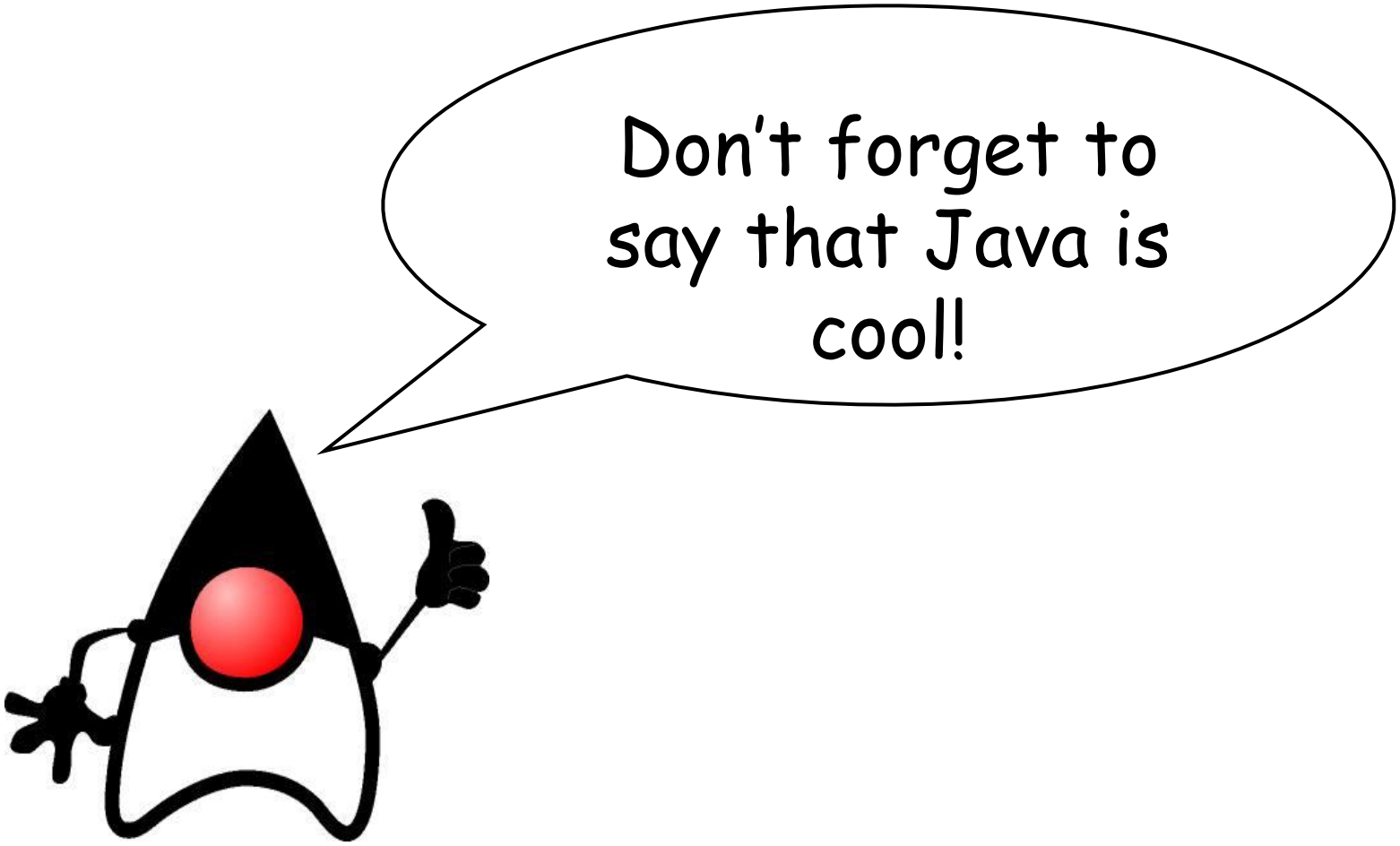
J2EE Benefits

- High availability
- Scalability
- Integration with existing systems
- Freedom to choose vendors of application servers, tools, components
- Multi-platform

J2EE Benefits

- Flexibility of scenarios and support to several types of clients
- Programming productivity:
  - Services allow developer to focus on business
  - Component development facilitates maintenance and reuse
  - Enables deploy-time behaviors
  - Supports division of labor

J2EE Benefits



Main technologies/ J2EE Component

- **Primary technologies**
  - JavaServer Pages (JSP)
  - Servlet
  - Enterprise JavaBeans (EJB)

☞ JSPs, servlets and EJBs are *application components*

JSP

- Used for web pages with dynamic content
- Processes HTTP requests (non-blocking call-and-return)
- Accepts HTML tags, special JSP tags, and scriptlets of Java code
- Separates static content from presentation logic
- Can be created by web designer using HTML tools

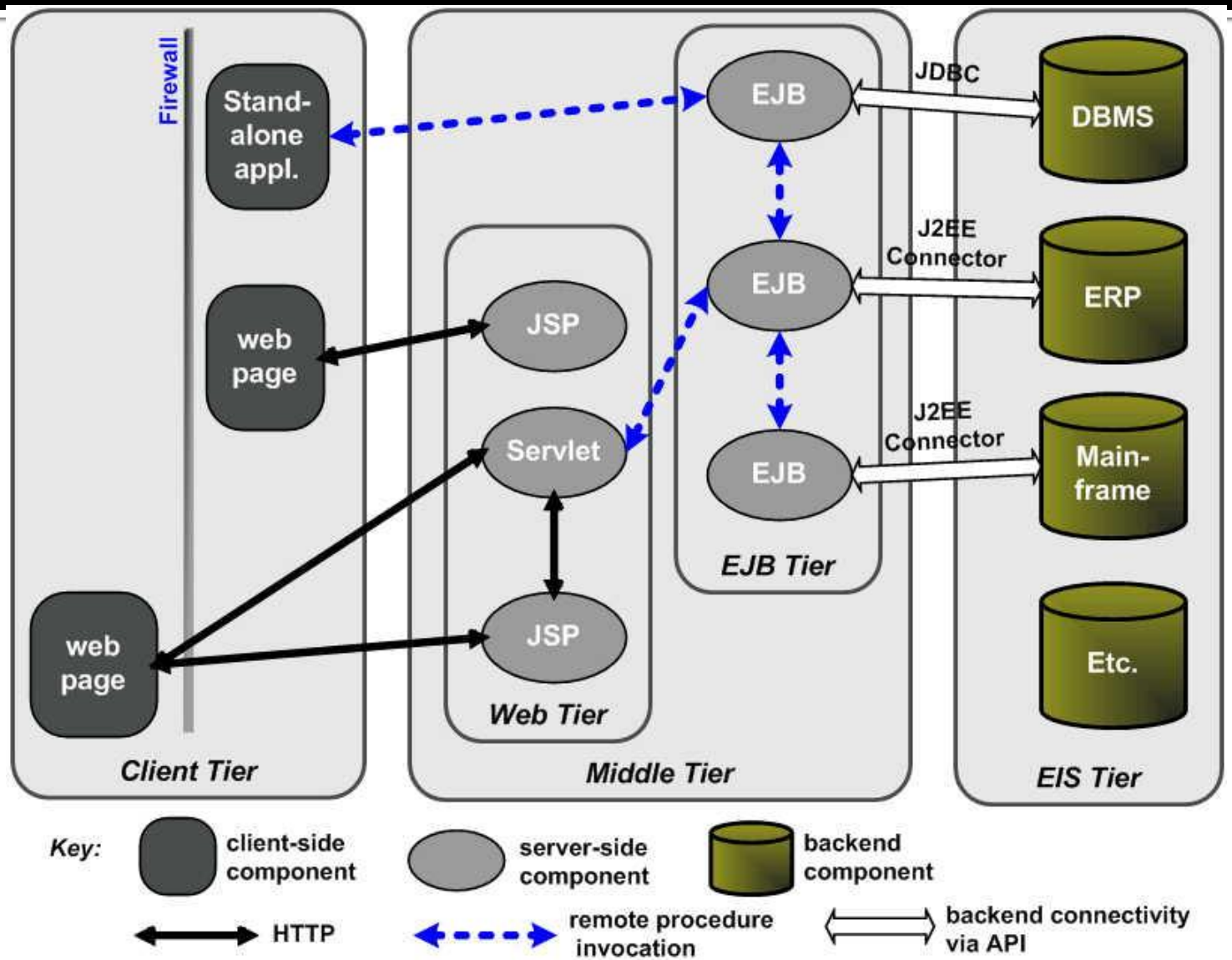
Servlet

- Used for web pages with dynamic content
- Processes HTTP requests (non-blocking call-and-return)
- Written in Java; uses print statements to render HTML
- Loaded into memory once and then called many times
- Provides APIs for session management

EJB

- EJBs are *distributed components* used to implement business logic (no UI)
- Developer concentrates on business logic
- Availability, scalability, security, interoperability and integrability handled by the J2EE server
- Client of EJBs can be JSPs, servlets, other EJBs and external aplications
- Clients see *interfaces*

J2EE Multi-tier Model



4. Examples

- JSP example
- Servlet example
- EJB example

JSP example


Hello, User - Netscape
File Edit View Go Communicator Help
My name is Duke. What's yours?
Jon
Submit
Reset
Document Done

JSP example

```
<%@ page import="hello.Greeting" %>
<jsp:useBean id="mybean" scope="page" class="hello.Greeting"/>
<jsp:setProperty name="mybean" property="*" />
<html>
<head><title>Hello, User</title></head>
<body bgcolor="#ffffff" background="background.gif">
<%@ include file="dukebanner.html" %>
<table border="0" width="700">
<tr>
<td width="150">   </td>
<td width="550">
<h1>My name is Duke. What's yours?</h1>
</td>
</tr>
```

JSP example

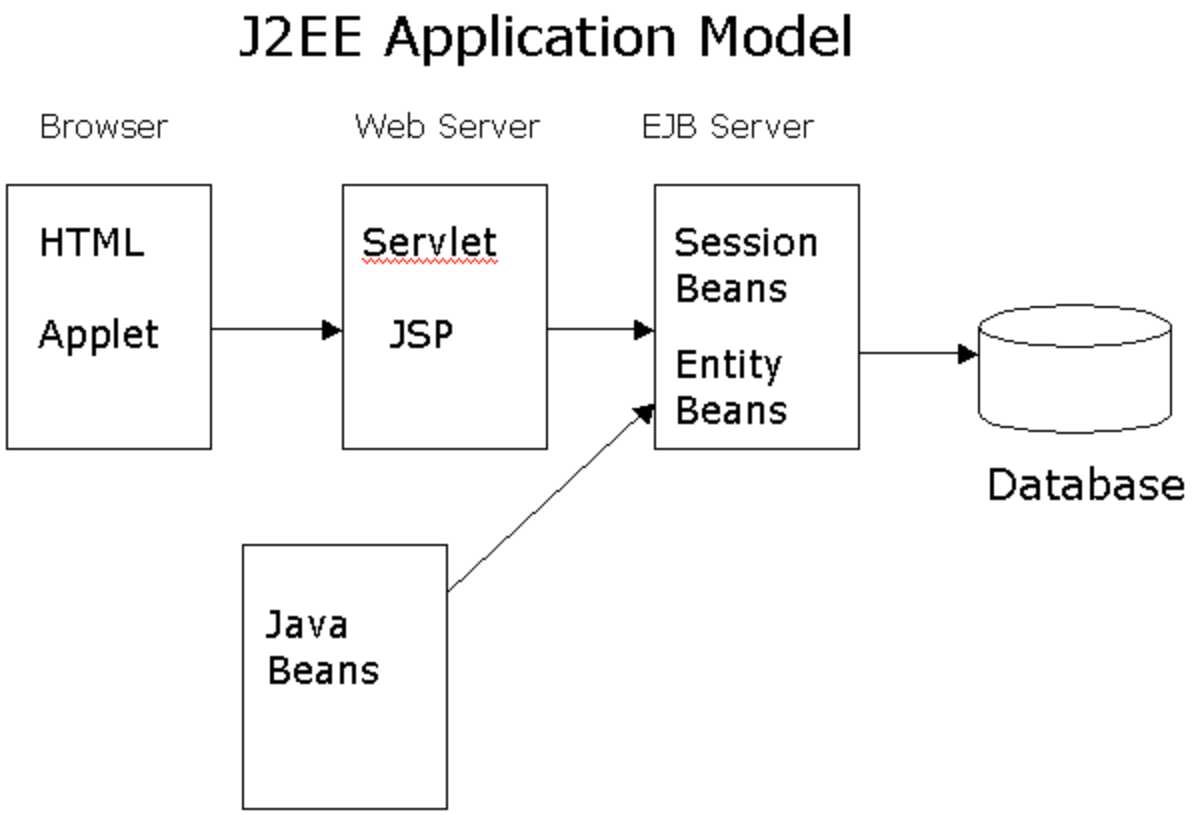
```
<tr> <td width="150"   </td> <td width="550">
<form method="get">
<input type="text" name="username" size="25"> <br>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</td> </tr>
</form> </table>
<%
    if (request.getParameter("username") != null) {
%>
<%@ include file="response.jsp" %>
<%
    }
%>
</body>
</html>
```

Servlet example

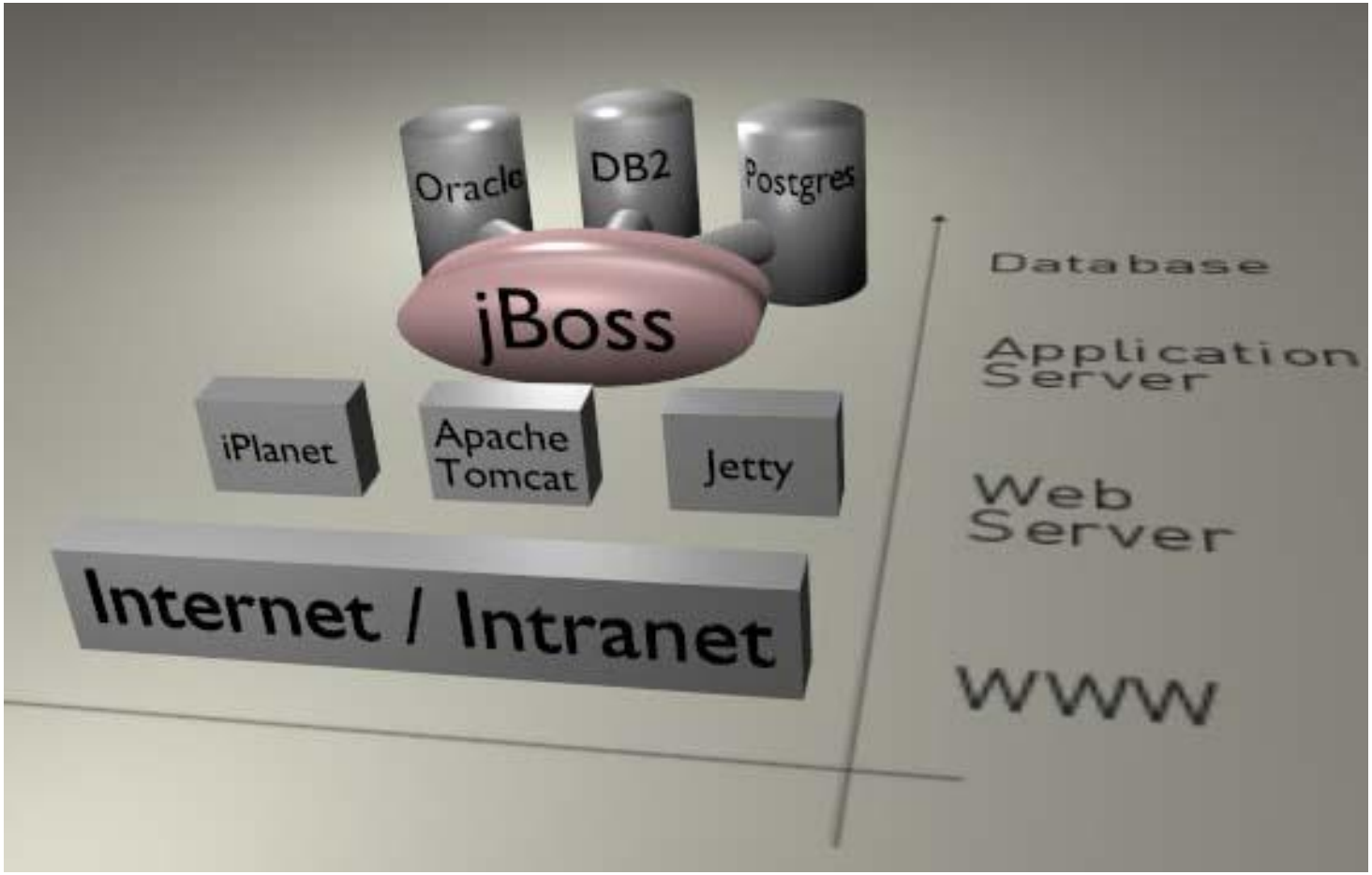
```
public class HelloWorldServlet extends HttpServlet {

  public void service(HttpServletRequest req,
       HttpServletResponse res) throws IOException {

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<html><head><title>Hello
                 World Servlet</title></head>");
    out.println("<body><h1>Hello
                 World!</h1></body></html>");
  }

}
```

Questions / Let's start with Tomcat

J2EE Application Model

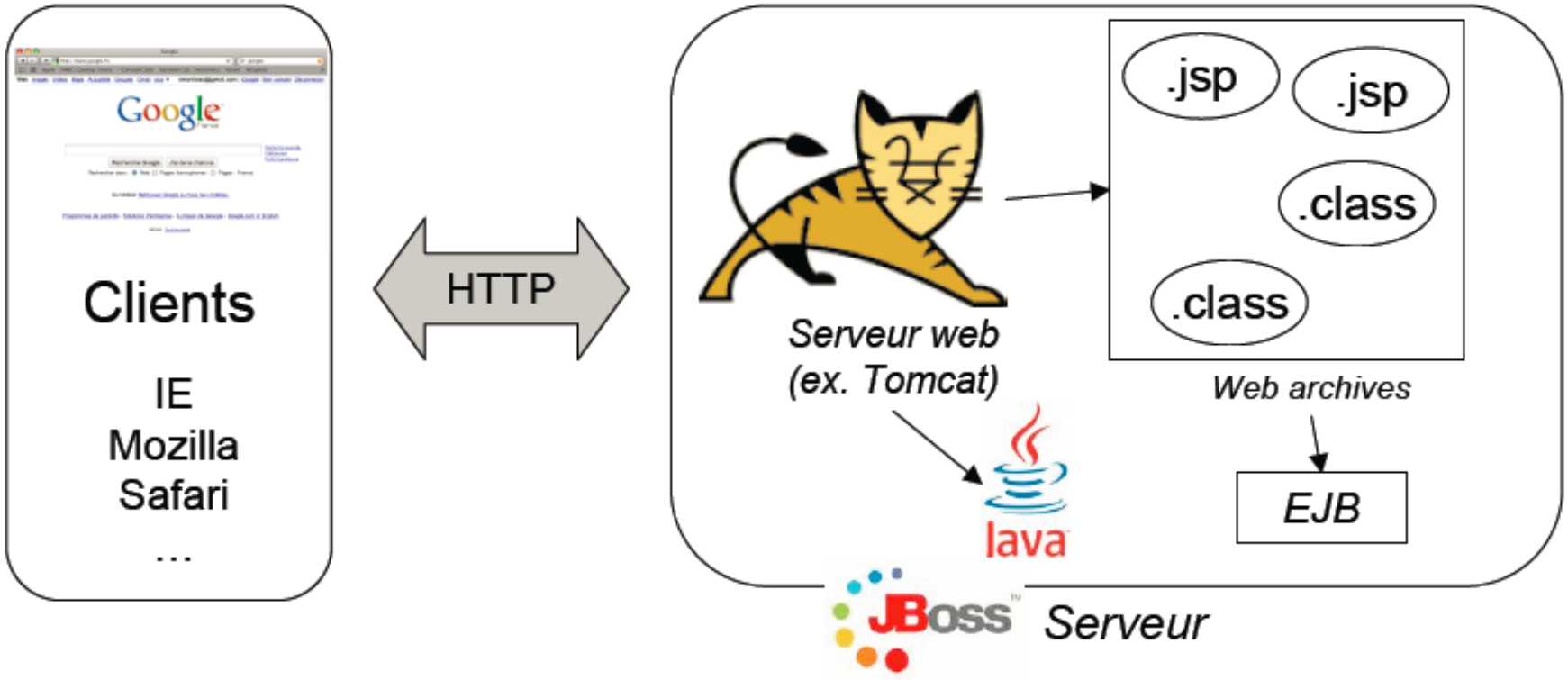


JBoss- Application Server



Part II

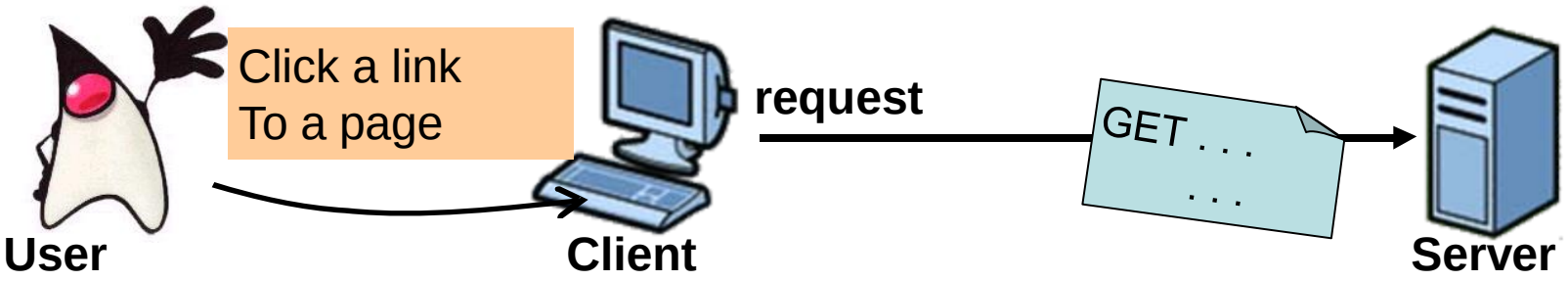
JSP/Servlet


Google
Clients
IE
Mozilla
Safari
...
HTTP
.jsp
.jsp
.class
.class
Serveur web
(ex. Tomcat)
Web archives
Java
EJB
JBoss
Serveur

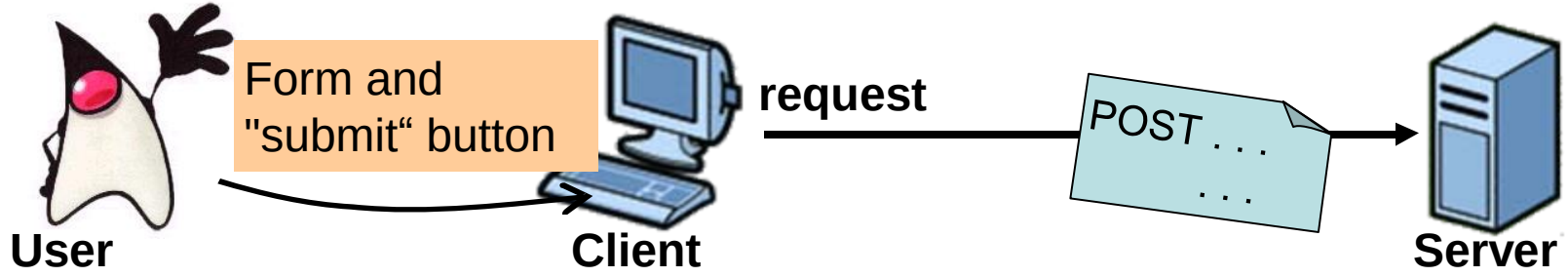
HTTP Request

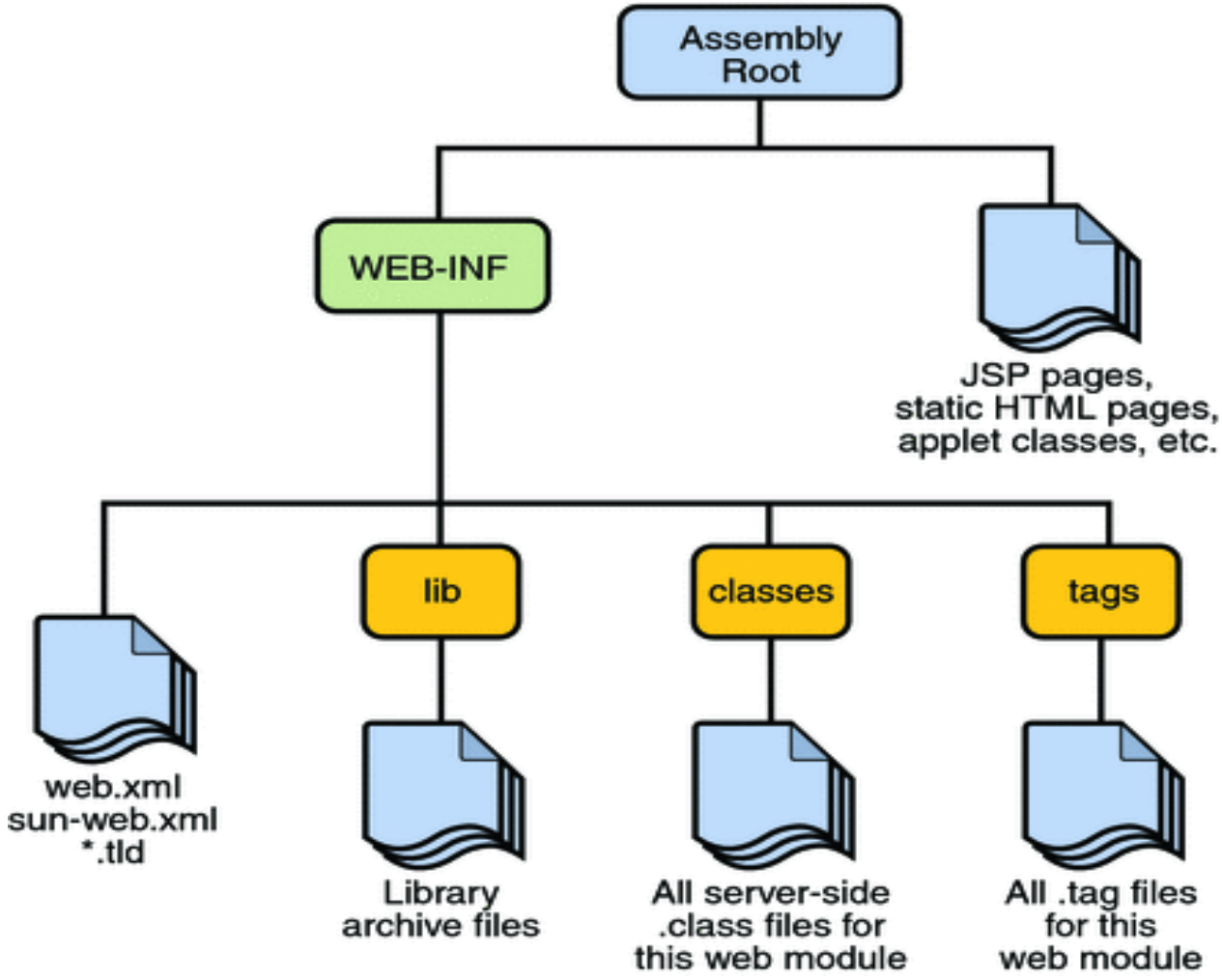
- Two main methods: GET and POST

GET



GET ou **POST**




Assembly Root
WEB-INF
JSP pages, static HTML pages, applet classes, etc.
lib
classes
tags
web.xml
sun-web.xml
*.tld
Library archive files
All server-side .class files for this web module
All .tag files for this web module

Servlet

Overview of Servlets

- Are container managed web components
- Replace Common Gateway Interface(CGI) or Active Server Pages (ASP)
- Generate dynamic response to requests from web based clients
- Synchronize multiple concurrent client request
- Serve as client proxies

Servlet Operation

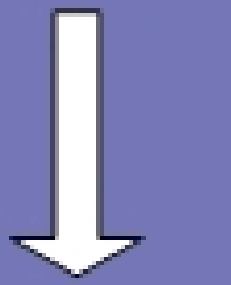
- Server is Java program that runs as separate thread inside servlet container.
- Servlet container is part of web server
- It interact with web client using response request paradigm

Interface Servlet and the Servlet Life Cycle

- Interface **`Servlet`**
  - All servlets must implement this interface
  - All methods of interface **`Servlet`** are invoked by servlet container
- Servlet life cycle
  - Servlet container invokes the servlet's **`init`** method
  - Servlet's **`service`** method handles requests
  - Servlet's **`destroy`** method releases servlet resources when the servlet container terminates the servlet
- Servlet implementation
  - **`GenericServlet`**
  - **`HttpServlet`**

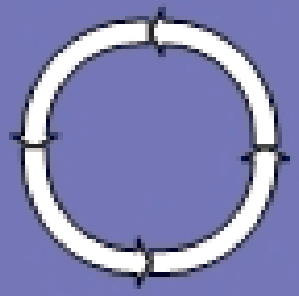
Servlet Life Cycle

Servlet LifeCycle



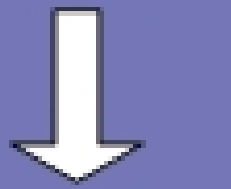
init(ServletConfig config)

Called on application startup or the first time the servlets is invoked. Useful for initialization of servlet resources like database connections.



service(javax.servlet.ServletRequest req,
javax.servlet.ServletResponse res)

Each time a message arrives the service method of the servlet is invoked in a separate thread to process it.



destroy()

Called on application disposal.

HttpServlet Class

- Overrides method **`service`**
- Two most common HTTP request types
  - **`get`** requests
  - **`post`** requests
- Method **`doGet`** responds to **`get`** requests
- Method **`doPost`** responds to **`post`** requests
- **`HttpServletRequest`** and **`HttpServletResponse`** objects

Servlet Code

```
package servlet;
   import java.io.IOException;
   import javax.servlet.ServletException;
   import javax.servlet.http.HttpServlet;
   import javax.servlet.http.HttpServletRequest;
   import javax.servlet.http.HttpServletResponse;

public class MyServlet extends HttpServlet {

   protected void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException

   {

   }

   protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException

   {

   }

   }
```

HttpServletRequest Interface

- Web server
  - creates an **`HttpServletRequest`** object
  - passes it to the servlet's **`service`** method
- **`HttpServletRequest`** object contains the request from the client

HttpServletRequest Interface

Method	Description
`String getParameter( String name )`	
	Obtains the value of a parameter sent to the servlet as part of a `get` or `post` request. The `name` argument represents the parameter name.
`Enumeration getParameterNames()`	
	Returns the names of all the parameters sent to the servlet as part of a `post` request.
`String[] getParameterValues( String name )`	
	For a parameter with multiple values, this method returns an array of strings containing the values for a specified servlet parameter.
`Cookie[] getCookies()`	
	Returns an array of `Cookie` objects stored on the client by the server. `Cookie` objects can be used to uniquely identify clients to the servlet.
`HttpSession getSession( boolean create )`	
	Returns an `HttpSession` object associated with the client's current browsing session. This method can create an `HttpSession` object (`true` argument) if one does not already exist for the client. `HttpSession` objects are used in similar ways to `Cookie`s for uniquely identifying clients.

Fig. Some methods of interface `HttpServletRequest`.

`HttpServletResponse` Interface

- Web server
  - creates an **`HttpServletResponse`** object
  - passes it to the servlet's **`service`** method

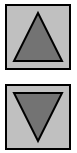
24.2.4 HttpServletResponse Interface (Cont.)

Method	Description
`void addCookie( Cookie cookie )`	
	Used to add a `Cookie` to the header of the response to the client. The `Cookie`'s maximum age and whether `Cookie`s are enabled on the client determine if `Cookie`s are stored on the client.
`ServletOutputStream getOutputStream()`	
	Obtains a byte-based output stream for sending binary data to the client.
`PrintWriter getWriter()`	
	Obtains a character-based output stream for sending text data to the client.
`void setContentType( String type )`	
	Specifies the MIME type of the response to the browser. The MIME type helps the browser determine how to display the data (or possibly what other application to execute to process the data). For example, MIME type `"text/html"` indicates that the response is an HTML document, so the browser displays the HTML page.

Fig. Some methods of interface `HttpServletResponse`.

Handling HTTP get Requests

- **`get`** request
  - Retrieve the content of a URL
- Example: **`WelcomeServlet`**
  - a servlet handles HTTP **`get`** requests



Outline

```
1  // Fig.: WelcomeServlet.java
2  // A simple servlet to process get requests.
3
4  import javax.servlet.*;
5  import javax.servlet.http.*;
6  import java.io.*;
7
8  public class WelcomeServlet extends HttpServlet {
9
10    // process "get" requests from clients
11    protected void doGet( HttpServletRequest request,
12       HttpServletResponse response )
13          throws ServletException, IOException
14    {
15       response.setContentType( "text/html" );
16       PrintWriter out = response.getWriter();
17
18       // send XHTML page to client
19
20       // start XHTML document
21       out.println( "<?xml version = \"1.0\"?>" );
22
23       out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
24          "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
25          "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
26
```

WelcomeServlet

Lines 4-5

Lines 11-42

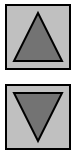
Import the `javax.servlet` and `javax.servlet.http` packages.

Extends `HttpServlet` to handle HTTP `get` requests

Override method `doGet` to provide custom get request processing.

Uses the `response` object's `getWriter` method to obtain a reference to the `PrintWriter` object that enables the servlet to send content to the client.

Create the XHTML document by writing strings with the `out` object's `println` method.



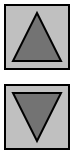
Outline

WelcomeServlet

Line 41

```
27          out.println( "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
28
29          // head section of document
30          out.println( "<head>" );
31          out.println( "<title>A Simple Servlet Example</title>" );
32          out.println( "</head>" );
33
34          // body section of document
35          out.println( "<body>" );
36          out.println( "<h1>Welcome to Servlets!</h1>" );
37          out.println( "</body>" );
38
39          // end XHTML document
40          out.println( "</html>" );
41          out.close();  // close stream to complete the page
42       }
43    }
```

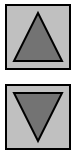
Closes the output stream, flushes the output buffer and sends the information to the client.



Outline

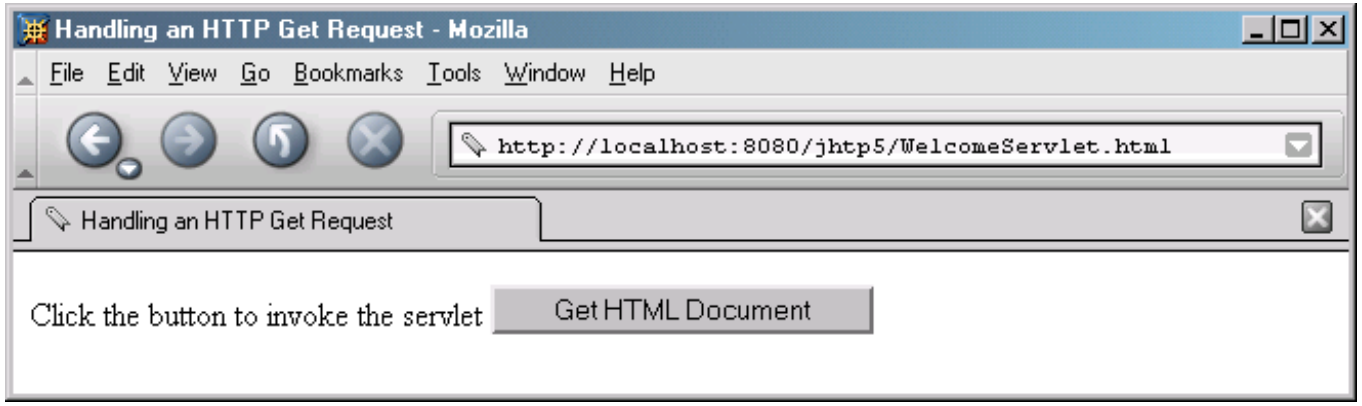
WelcomeServlet.html

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig.: WelcomeServlet.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9     <title>Handling an HTTP Get Request</title>
10 </head>
11
12 <body>
13    <form action = "/jhtp5/welcome1" method = "get">
14
15       <p><label>Click the button to invoke the servlet
16          <input type = "submit" value = "Get HTML Document" />
17       </label></p>
18
19    </form>
20 </body>
21 </html>
```



Outline

Program output



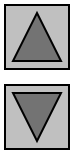
Deploying a Web Application

- Web applications
  - JSPs, servlets and their supporting files
- Deploying a Web application
  - Directory structure
    - Context root
  - Web application archive file (WAR file)
  - Deployment descriptor
    - **`web.xml`**

Deploying a Web Application (Cont.)

Directory	Description
context root	This is the root directory for the Web application. All the JSPs, HTML documents, servlets and supporting files such as images and class files reside in this directory or its subdirectories. The name of this directory is specified by the Web application creator. To provide structure in a Web application, subdirectories can be placed in the context root. For example, if your application uses many images, you might place an images subdirectory in this directory. The examples of this chapter use `jhtp5` as the context root.
`WEB-INF`	This directory contains the Web application *deployment descriptor* (*`web.xml`*).
`WEB-INF/classes`	This directory contains the servlet class files and other supporting class files used in a Web application. If the classes are part of a package, the complete package directory structure would begin here.
`WEB-INF/lib`	This directory contains Java archive (JAR) files. The JAR files can contain servlet class files and other supporting class files used in a Web application.

Fig. Web application standard directories.



Outline

```
1  <!DOCTYPE web-app PUBLIC \
2     "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
3     "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
4
5  <web-app>
6
7     <!-- General descri
8     <display-name>
9        Java How to Program JSP
10       and Servlet Chapter Examples
11    </display-name>
12
13    <description>
14       This is the Web appli
15       demonstrate our JSP a
16    </description>
17
18    <!-- Servlet definitions -->
19    <servlet>
20       <servlet-name>welcome1</servlet-name>
21
22       <description>
23          A simple servlet th                quest.
24       </description>
25
```

Element `web-app` defines the configuration of each servlet in the Web application and the servlet mapping for each servlet.

Element `display-name` specifies a name that can be displayed to the administrator of the server on which the Web application is installed.

Element `description` specifies a description of the Web application that might be displayed to the administrator of the server.

Eleme t.

Element `servlet-name` is the name for the servlet.

Element `description` specifies a description for this particular servlet.

web.xml

Lines 5-37

Lines 8-11

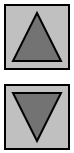
Lines 13-16

Lines 19-29

Line 20

Lines 22-24

26-28



Outline

```
26        <servlet-class>
27           WelcomeServlet
28        </servlet-class>
29     </servlet>
30
31     <!-- Servlet mappings -->
32     <servlet-mapping>
33        <servlet-name>welcome1</servlet-name>
34        <url-pattern>/welcome1</url-pattern>
35     </servlet-mapping>
36
37  </web-app>
```

Element `servlet-class` specifies compiled servlet's fully qualified class name.

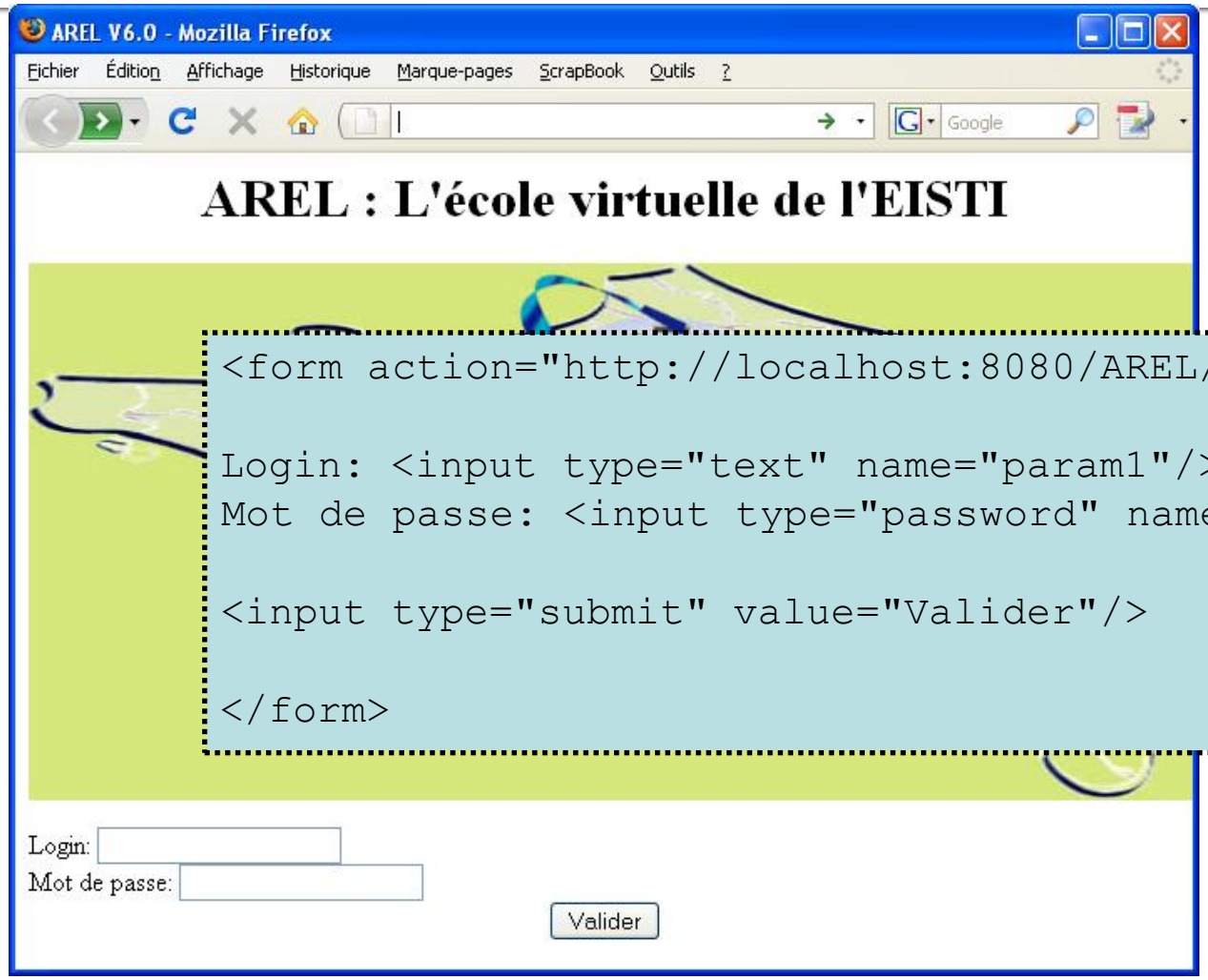
Element `servlet-mapping` specifies `servlet-name` and `url-pattern` elements.

web.xml

es 26-28

es 32-35

GET form




```
<form action="http://localhost:8080/AREL/LogServlet">

Login: <input type="text" name="param1"/><br/>
Mot de passe: <input type="password" name="param2"/><br/>

<input type="submit" value="Valider"/>

</form>
```

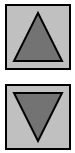
GET form

GET /AREL/LogServlet?param1=monlogin&param2=monpass HTTP/1.1
host: localhost:8080
user-agent: Mozilla/5.0 (…) Gecko/2008092417 Firefox/3.0.3
accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
accept-language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
…

```
public class LogServlet extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response) throws IOException{
    String login=request.getParameter("param1");
    String password=request.getParameter("param2");
    if (checkUserAndPassword(login, password)){
        grantAccessTo(login);
    } else{
        sendAuthentificationFailure(login);
    }
  }
}
```

Handling HTTP get Requests Containing Data

- Servlet **`WelcomeServlet2`**
  - Responds to a **`get`** request that contains data



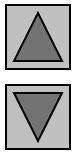
Outline

WelcomeServlet2 responds to a get request that contains data.

Line 15

```
1  // Fig.WelcomeServlet2.java
2  // Processing HTTP get requests containing data.
3
4  import javax.servlet.*;
5  import javax.servlet.http.*;
6  import java.io.*;
7
8  public class WelcomeServlet2 extends HttpServlet {
9
10    // process "get" request from client
11    protected void doGet( HttpServletRequest request,
12       HttpServletResponse response )
13          throws ServletException, IOException
14    {
15       String firstName = request.getParameter( "firstname" );
16
17       response.setContentType( "text/html" );
18       PrintWriter out = response.getWriter();
19
20       // send XHTML document to client
21
22       // start XHTML document
23       out.println( "<?xml version = \"1.0\"?>" );
24
```

The `request` object's `getParameter` method receives the parameter name and returns the corresponding `String` value.



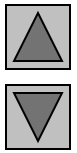
Outline

WelcomeServlet2 responds to a get request that contains data.

Line 39

```
25          out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
26             "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
27             "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
28
29          out.println( "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
30
31          // head section of document
32          out.println( "<head>" );
33          out.println(
34             "<title>Processing get requests with data</title>" );
35          out.println( "</head>" );
36
37          // body section of document
38          out.println( "<body>" );
39          out.println( "<h1>Hello " + firstName + ",<br />" );
40          out.println( "Welcome to Servlets!</h1>" );
41          out.println( "</body>" );
42
43          // end XHTML document
44          out.println( "</html>" );
45          out.close();  // close stream to complete the page
46       }
47    }
```

Uses the result of line 16 as part of the response to the client.



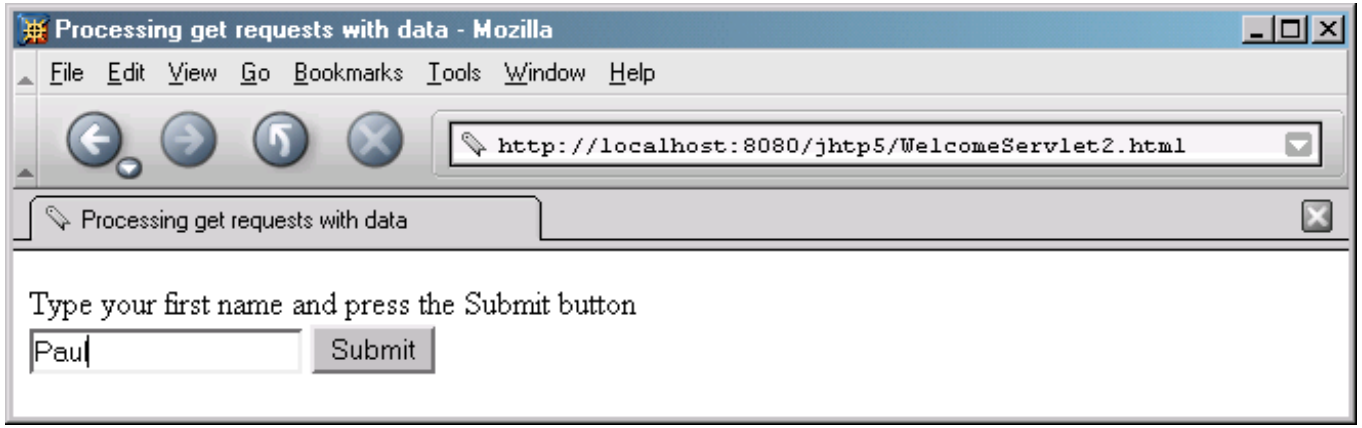
Outline

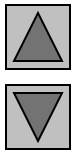
HTML document in which the form's action invokes WelcomeServlet2 through the alias welcome2 specified in web.xml.

Line 17

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig.  WelcomeServlet2.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9     <title>Processing get requests with data</title>
10 </head>
11
12 <body>
13    <form action = "/jhtp5/welcome2" method = "get">
14
15       <p><label>
16          Type your first name and press the Submit button
17          <br /><input type = "text" name = "firstname" />
18          <input type = "submit" value = "Submit" />
19       </p></label>
20
21    </form>
22 </body>
23 </html>
```

Get the first name from the user.





Outline

HTML document in which the form's action invokes WelcomeServlet2 through the alias welcome2 specified in web.xml.

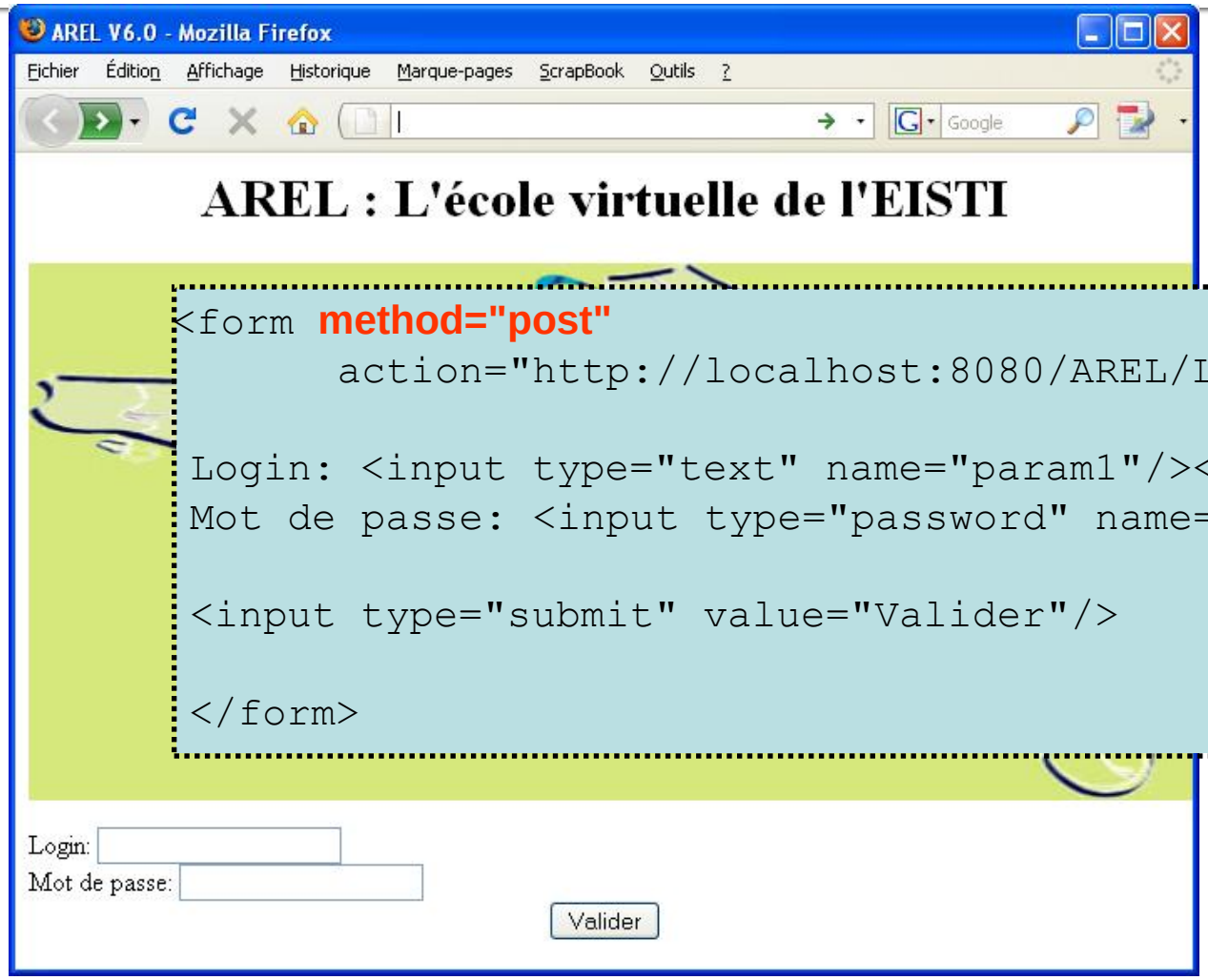
Program output

Handling HTTP get Requests Containing Data

Descriptor element	Value
servlet element	
servlet-name	welcome2
description	Handling HTTP get requests with data.
servlet-class	WelcomeServlet2
servlet-mapping element	
servlet-name	welcome2
url-pattern	/welcome2

Fig. Deployment descriptor information for servlet WelcomeServlet2.

Formulaire POST




```
<form method="post"
      action="http://localhost:8080/AREL/LogServlet">

Login: <input type="text" name="param1"/><br/>
Mot de passe: <input type="password" name="param2"/><br/>

<input type="submit" value="Valider"/>

</form>
```

Traitement formulaire POST

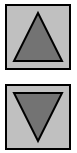
POST /AREL/LogServlet HTTP/1.1
host: localhost:8080
user-agent: Mozilla/5.0 (…) Gecko/2008092417 Firefox/3.0.3
accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
…
content-type: application/x-www-form-urlencoded
content-length: 30

param1=monlogin&param2=monpass

```
public class LogServlet extends HttpServlet{
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response) throws IOException{

                Idem que doGet
    }
}
```

Handling HTTP post Requests

- HTTP post request
  - Post data from an HTML form to a server-side form handler
  - Browsers cache Web pages
- Servlet WelcomeServlet3
  - Responds to a **`post`** request that contains data

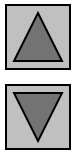


Outline

WelcomeServlet3 responds to a post request that contains data.

```
1  // Fig.: WelcomeServlet3.java
2  // Processing post requests containing data.
3
4  import javax.servlet.*;
5  import javax.servlet.http.*;
6  import java.io.*;
7
8  public class WelcomeServlet3 extends HttpServlet {
9
10    // process "post" request from client
11    protected void doPost( HttpServletRequest request,
12       HttpServletResponse response )
13          throws ServletException, IOException
14    {
15       String firstName = request.getParameter( "firstname" );
16
17       response.setContentType( "text/html" );
18       PrintWriter out = response.getWriter();
19
20       // send XHTML page to client
21
22       // start XHTML document
23       out.println( "<?xml version = \"1.0\"?>" );
24
```

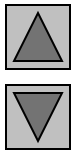
Declare a `doPost` method to responds to post requests.



Outline

WelcomeServlet3.java

```
25          out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
26             "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
27             "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
28
29          out.println( "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
30
31          // head section of document
32          out.println( "<head>" );
33          out.println(
34             "<title>Processing post requests with data</title>" );
35          out.println( "</head>" );
36
37          // body section of document
38          out.println( "<body>" );
39          out.println( "<h1>Hello " + firstName + ",<br />" );
40          out.println( "Welcome to Servlets!</h1>" );
41          out.println( "</body>" );
42
43          // end XHTML document
44          out.println( "</html>" );
45          out.close();  // close stream to complete the page
46       }
47    }
```

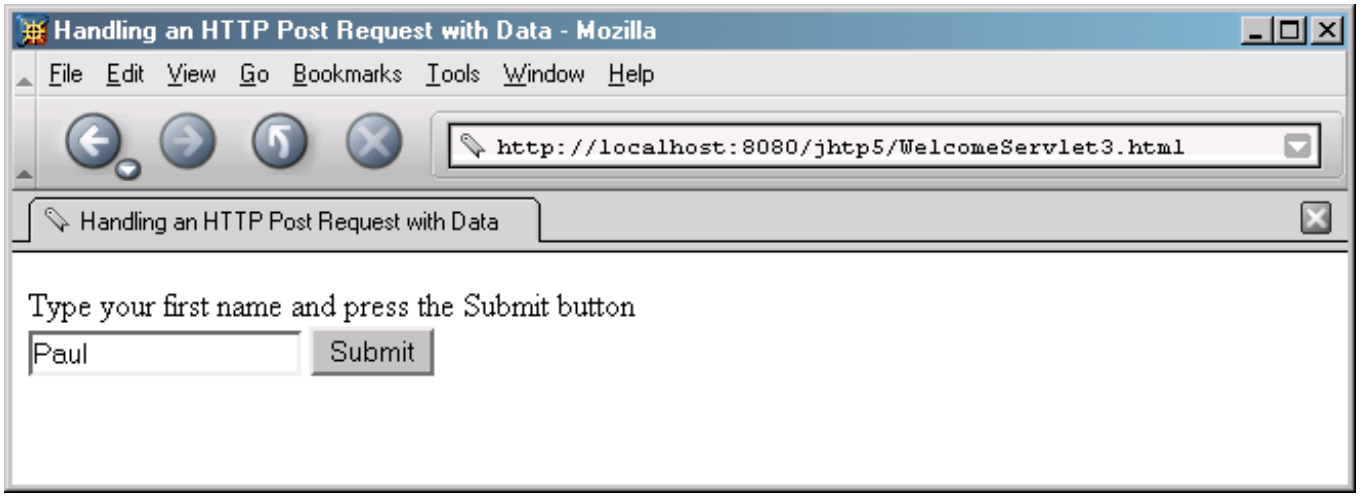


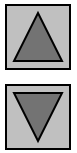
Outline

HTML document in which the form's action invokes WelcomeServlet3 through the alias welcome3 specified in

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. : WelcomeServlet3.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9     <title>Handling an HTTP Post Request with Data</title>
10 </head>
11
12 <body>
13    <form action = "/jhtp5/welcome3" method = "post">
14
15       <p><label>
16          Type your first name and press the Submit button
17          <br /><input type = "text" name = "firstname" />
18          <input type = "submit" value = "Submit" />
19       </label></p>
20
21    </form>
22 </body>
23 </html>
```

Provide a `form` in which the user can input a name in the `text` input element `firstname`, then click the `Submit` button to invoke `WelcomeServlet3`.





Outline

HTML document in which the form's action invokes WelcomeServlet3 through the alias welcome3 specified in web.xml.

Program output

24.5 Handling HTTP post Requests (Cont.)

Descriptor element	Value
servlet element	
servlet-name	welcome3
description	Handling HTTP post requests with data.
servlet-class	WelcomeServlet3
servlet-mapping element	
servlet-name	welcome3
url-pattern	/welcome3

Fig. Deployment descriptor information for servlet WelcomeServlet3.

Form verification

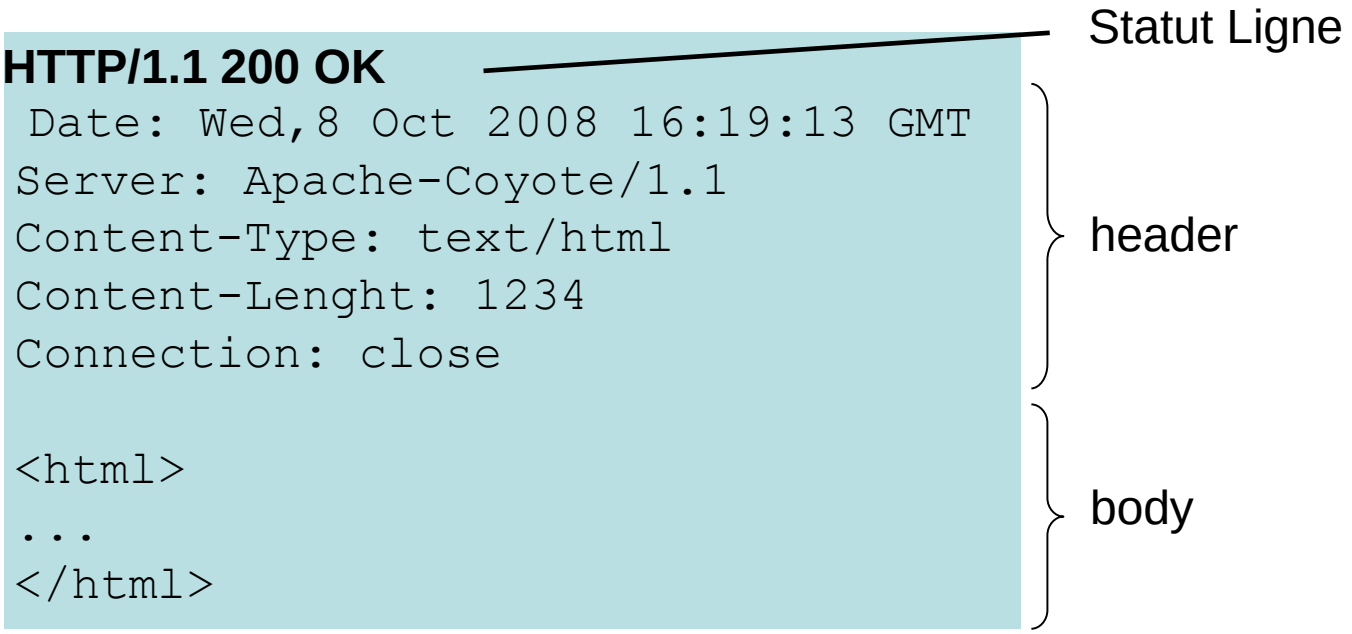
Missing data

- Form missing field
  - **getParameter** return *null*
- Returned empty field
  - **getParameter** returns an empty string (or a chain with spacings)

```
String param= request.getParameter("someName");
if((param == null) || (param.trim().equals(""))){
    doSomethingForMissingValues(...);
} else{
    doSomethingWithParameter(param);
}
```

HTTP response header

- Ex:



Some answers codes

- 200 OK
- 301 MOVED
- 403 FORBIDDEN
- 404 NOT FOUND
- 503 SERVICE UNAVAILABLE

Status Codes

- response.setStatus(int statusCode)
  - Ex: SC_OK, SC_NOT_FOUND, etc…
- response.sendError(int code, String msg)
- response.sendRedirect(String url)

sendError Example

```
public class LogServlet extends HttpServlet{
public void doGet (HttpServletRequest request,
                   HttpServletResponse response) throws IOException{
     String login= request.getParameter("param1");
     String password= request.getParameter("param2");

     if ((param1 == null) || (param1.trim().equals(""))){
       response.sendError(HttpServletResponse.SC_NOT_FOUND,
                          "Empty login");
         return;
    }

     if (checkUserAndPassword(login, password)){
         grantAccessTo(login);
     } else{
       response.sendError(HttpServletResponse.SC_UNAUTHORIZED,
                          "Access Denied to"+ login);
    }
  }
}
```

sendRedirect Example

```
public class WrongDestination extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException{
    String userAgent= request.getHeader("User-Agent");
    if ((userAgent != null) &&
        (userAgent.contains("MSIE")){
      response.sendRedirect("http://home.netscape.com");
     } else{
      response.sendRedirect("http://www.microsoft.com");
    }
  }
}
```

setContentType

Type	Meaning
application/msword	Microsoft Word document
application/octet-stream	Unrecognized or binary data
application/pdf	Acrobat (.pdf) file
application/postscript	PostScript file
application/vnd.ms-excel	Excel spreadsheet
application/vnd.ms-powerpoint	Powerpoint presentation
application/x-gzip	Gzip archive
application/x-java-archive	JAR file
application/x-java-vm	Java bytecode (.class) file
application/zip	Zip archive
audio/basic	Sound file in .au or .snd format
audio/x-aiff	AIFF sound file
audio/x-wav	Microsoft Windows sound file
audio/midi	MIDI sound file
text/css	HTML cascading style sheet
text/html	HTML document
text/plain	Plain text
text/xml	XML document
image/gif	GIF image
image/jpeg	JPEG image
image/png	PNG image
image/tiff	TIFF image
video/mpeg	MPEG video clip
video/quicktime	QuickTime video clip

Java Server Page (JSP)

Servlet Example: disadvantage

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

public class Clock extendsHttpServlet{
    public void doGet(HttpServletRequestrequest,
                      HttpServletResponse response)
        Throws IOException{
         PrintWriter out=  response.getWriter();
          java.util.Date today=newjava.util.Date();

       String docType="<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0    "+
                      "Transitional//EN\">\n";

        out.println(docType);
        out.println("<html>");
        out.println("<head>\n<title>Clock</title>\n</head>");
         out.println("<body>\n"+
                     "<h1>Time on server</h1>\n"+
                      "<p>"+  today+  "</p>\n"+
                    "</body>");
        out.println("</html>");
    }
```

classical servlet : redefine doGet() of HttpServlet

HTML Code Included in In Java

Servlet Problem
HTML in Java

1. it's ugly

```
out.println(docType);
out.println("<html>");
out.println("<head>\n<title>Clock</title>\n</head>");
 out.println("<body>\n"+
          "<h1>Time on server</h1>\n"+
              "<p>"+  today+"</p>\n"+
             "</body>");
out.println("</html>");
```

3. with Java the Servlets facilitate HTTP requests and responses processing, **but** they are not suitable to HTML writing

   - **out.println("<html><head><title>"+title+"</title>…");**

2. The EISTI engineers know Java, but web developers, NOT !!
3. Difficult to separate the different tasks of development (Web code vs. Business code)

Solution : Java in HTML

- A JSP is identical to a HTML page where you can add Java code (same principle as with PHP)

```
<!DOCTYPE HTML PUBLIC
 "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
    <title>Clock</title>
  </head>
  <body>
   <h1>Time on server</h1>
              <p> <%= new java.util.Date() %> </p>
  </body>
</html>
```

What is JSP technology?

JavaServer Pages (JSP) technology provides a simplified, fast way to create web pages that display dynamically-generated content.

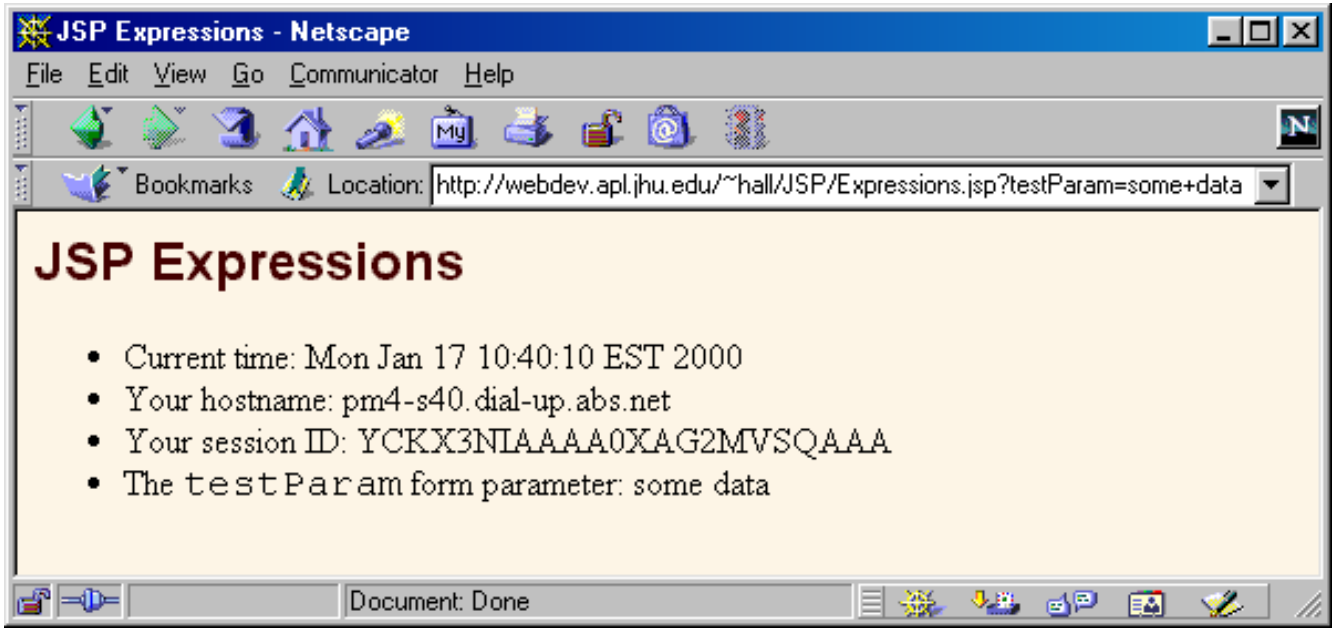
- Text based documents describe how to process a request and create a response
- Contains HTML or XML and other JSP elements defined by JSP specification.
- Are Installed on web server
- are web components that sits on top of java servlet mode.

Example

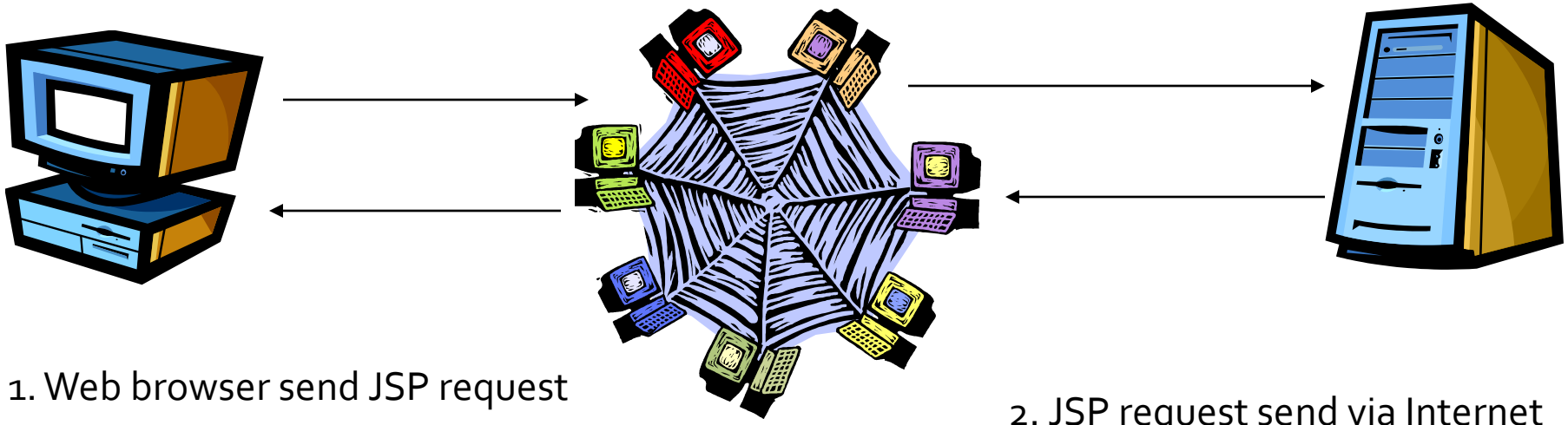
```
<HTML><HEAD>
<TITLE>JSP Expressions</TITLE>
<META NAME="author" CONTENT="Marty Hall">
<META NAME="keywords"
  CONTENT="JSP,expressions,JavaServer,Pages,servlets">
<META NAME="description"
      CONTENT="A quick example of JSP expressions.">
<LINK REL=STYLESHEET HREF="JSP-Styles.css"
  TYPE="text/css">
</HEAD>
<BODY>
<H2>JSP Expressions</H2>
<UL>
  <LI>Current time: <%= new java.util.Date() %>
  <LI>Your hostname: <%= request.getRemoteHost() %>
  <LI>Your session ID: <%= session.getId() %>
  <LI>The <CODE>testParam</CODE> form parameter:
      <%= request.getParameter("testParam") %>
</UL></BODY></HTML>
```

Example Result

- With default setup, if location was
  - C:\<tomcatHome>\webapps\ROOT\Expressions.jsp
- URL would be
  - http://localhost/Expressions.jsp



How JSP works?

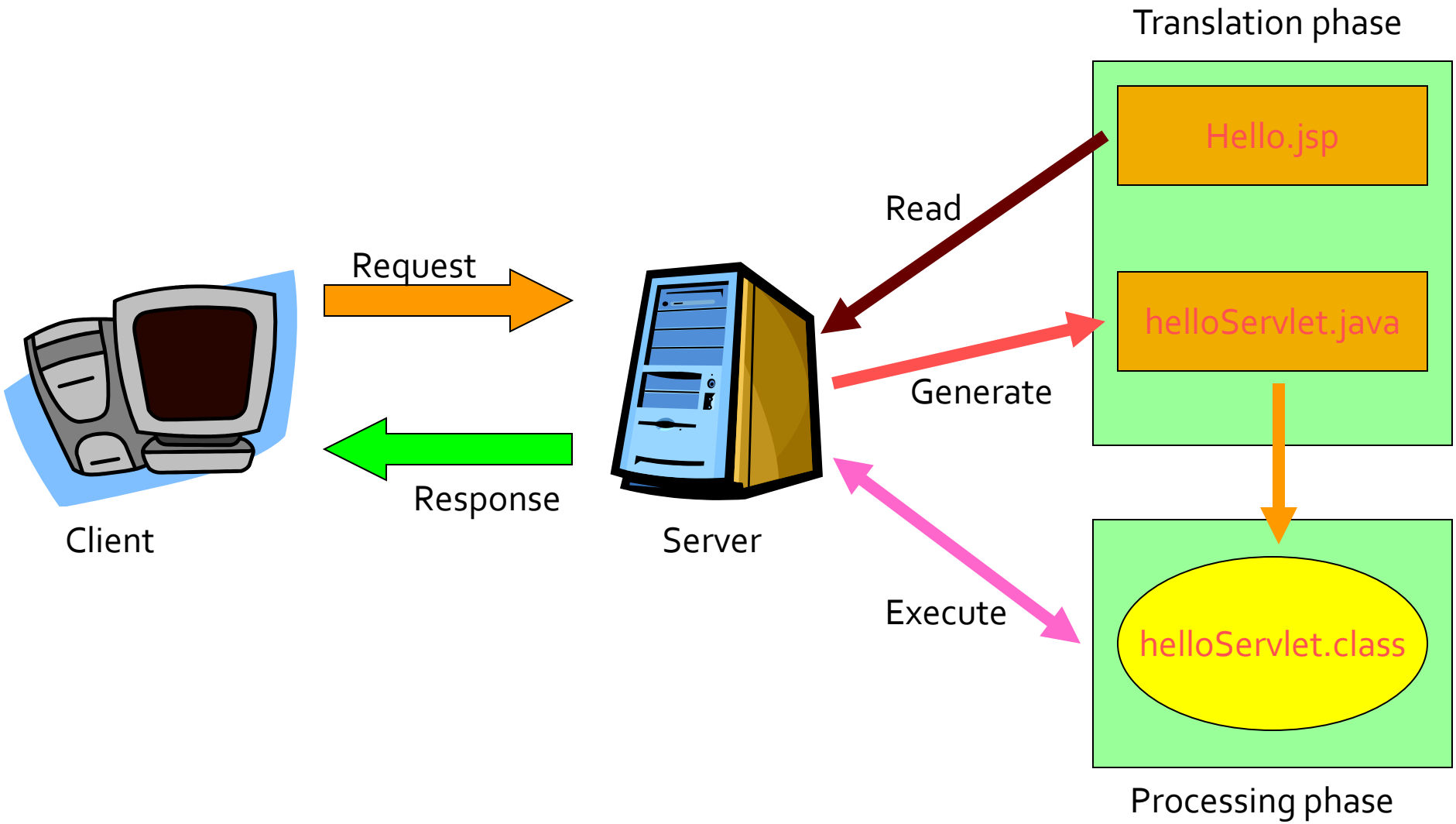


1. Web browser send JSP request

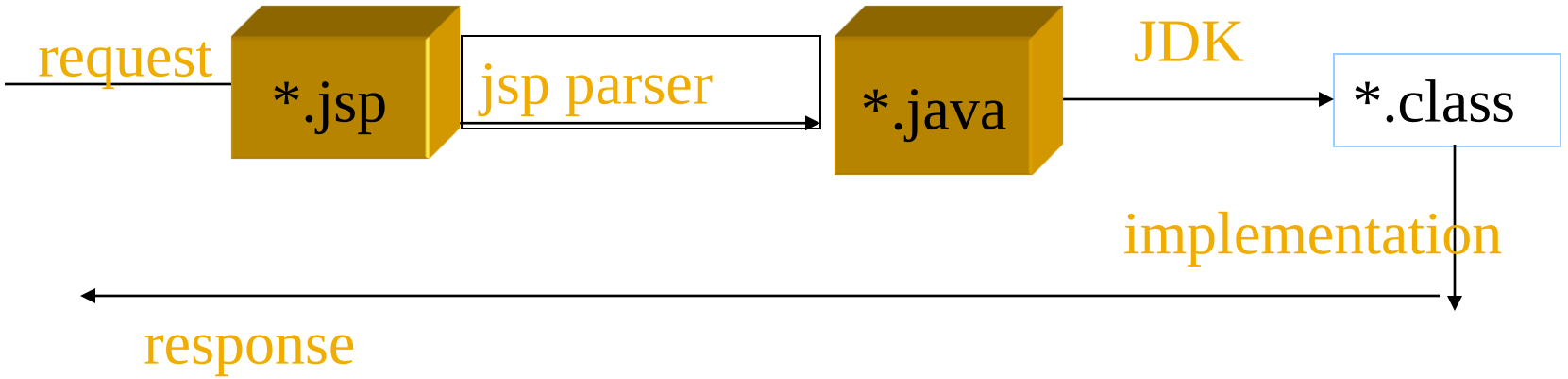
2. JSP request send via Internet to the web server
3. The web server send the JSP file (template pages) to JSP servlet engine
4. Parse JSP file
5. Generate servlet source code
6. Compile servlet to class
7. Instantiate servlet

8. HTML send back to the browser

JSP page translation and processing phases

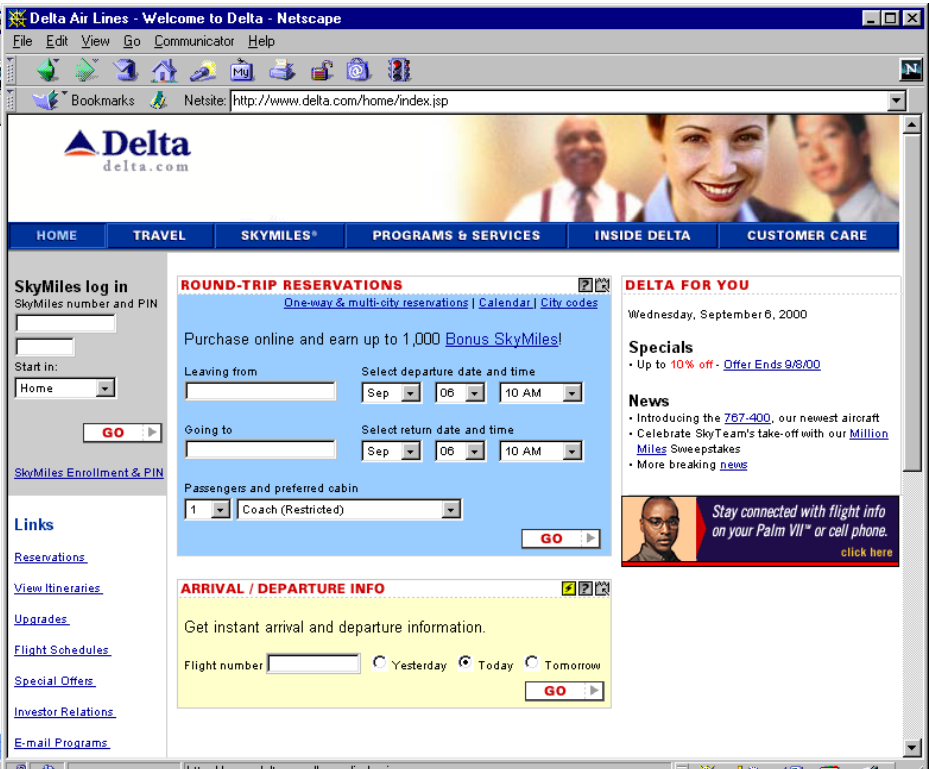


JSP's implementation



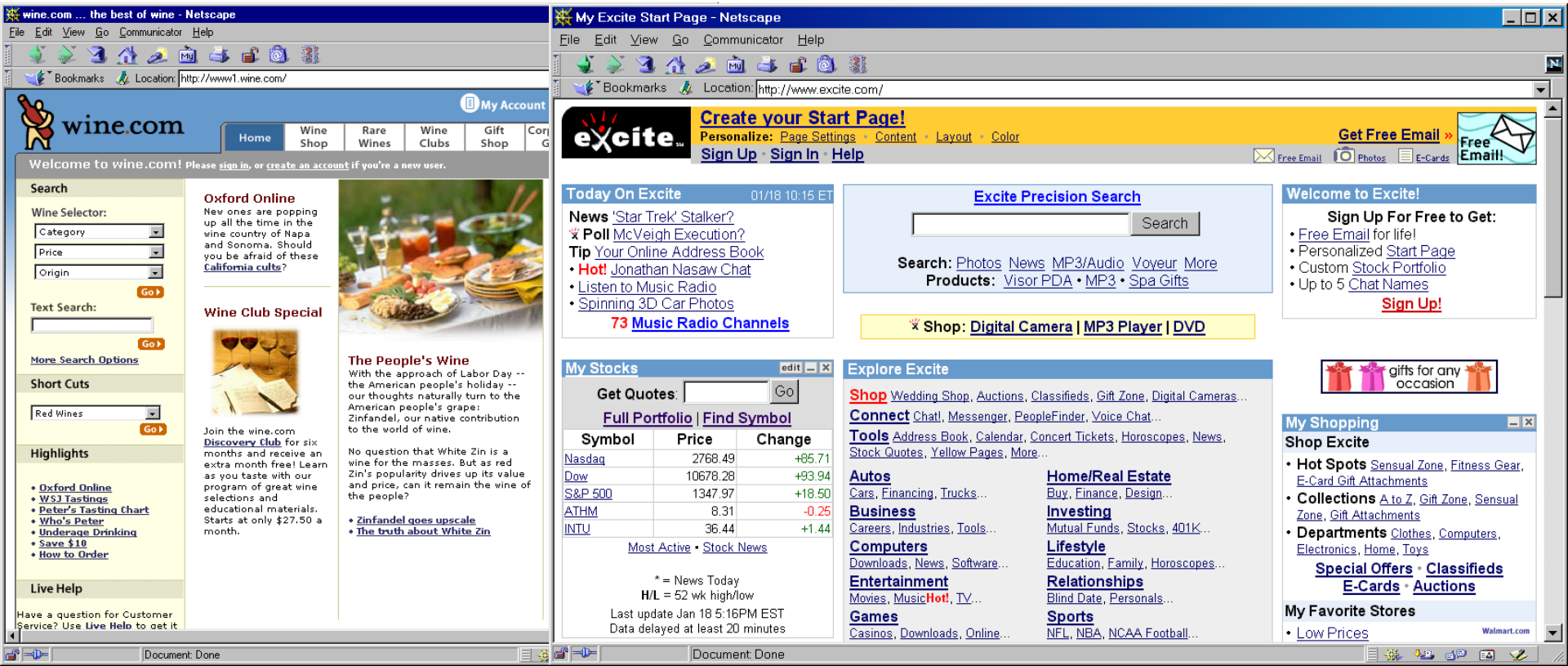
JSP/Servlets in the Real World

- Delta Airlines: entire Web site, including real-time schedule info
- First USA Bank: largest credit card issuer in the world; most on-line banking customers



JSP/Servlets in the Real World

- Excite: one of the top five Internet portals; one of the ten busiest sites on the Web



Template Pages

Server Page Template

```
<html>
<title>
A simple example
</title>

<body color="#FFFFFF">
The time now is
<%= new java.util.Date()
%>
</body>
</html>
```

Resulting HTML

```
<html>
<title>
A simple example
</title>

<body color="#FFFFFF">
The time now is
Mon Oct 28 09:50:11 PST2015
</body>
</html>
```

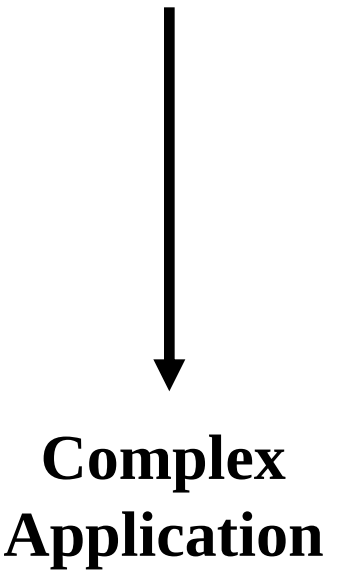
Hidden / HTML Comment

- An HTML comment is sent to the client's browser, but is not displayed. The information can be reviewed from the source code.
  - <!-- comment [<%= expression%>] -->
- A hidden comment is discarded before any processing of the JSP page and is not sent to the web browser.
  - <%-- comment -->

JSP Components

- There are three main types of JSP constructs that you embed in a page.
  - Scripting elements
    - You can specify Java code
    - Expressions, Scriptlets, Declarations
  - Directives
    - Let you control the overall structure of the servlet
    - Page, include, Tag library
  - Actions (Next Course)
    - Enable the use of server side Javabeans
    - Transfer control between pages

Uses of JSP Constructs: Use of Scripting elements



- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Custom tags
- Servlet/JSP combo (MVC architecture)

Types of Scripting Elements

- You can insert code into the servlet that will be generated from the JSP page.
- Expressions: <%= expression %>
  - Evaluated and inserted into the servlet's output. i.e., results in something like out.println(expression)
- Scriptlets: <% code %>
  - Inserted verbatim into the servlet's _jspService method (called by service)
- Declarations: <%! code %>
  - Inserted verbatim into the body of the servlet class, outside of any existing methods

Scriptlets JSP:
<% = expression %>

JSP Expressions

- Format
  - <%= Java Expression %>
- Result
  - Expression evaluated, converted to String, and placed into HTML page at the place it occurred in JSP page
  - That is, expression placed in _jspService inside out.print
- Examples
  - Current time: <%= new java.util.Date() %>
  - Your hostname: <%= request.getRemoteHost() %>
- XML-compatible syntax
  - <jsp:expression>Java Expression</jsp:expression>
  - XML version not supported by Tomcat 3. Until JSP 1.2, servers are not required to support it.

JSP/Servlet Correspondence

- Original JSP

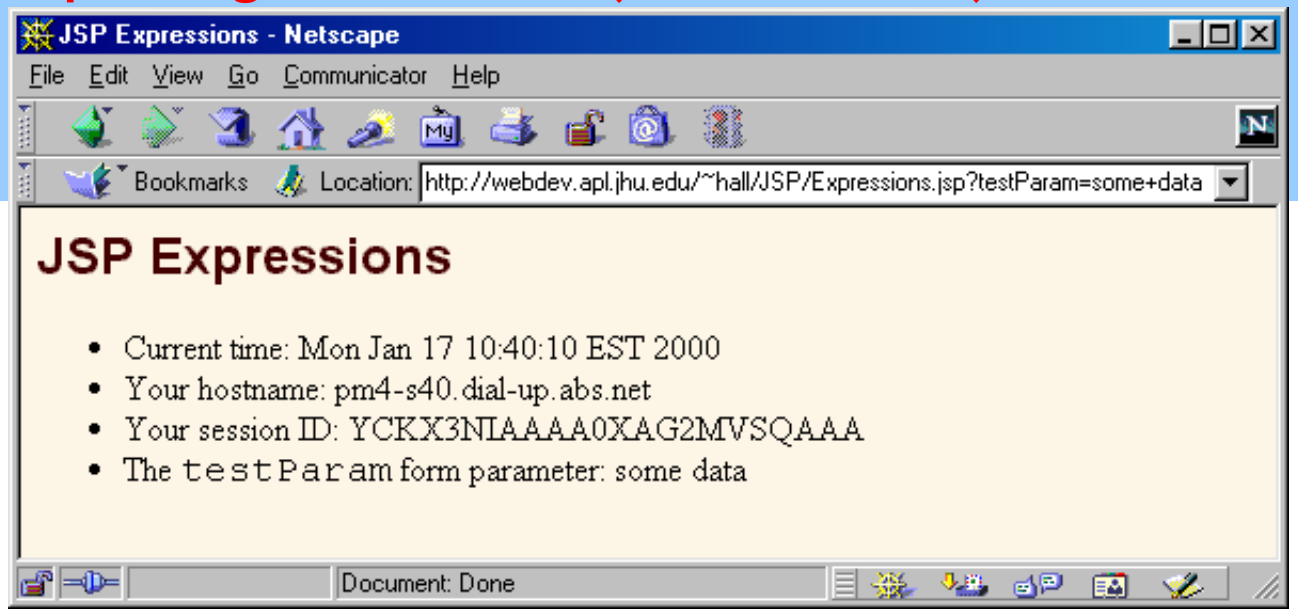
```
<H1>A Random Number</H1>
<%= Math.random() %>
```

- Possible resulting servlet code

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
    throws ServletException, IOException {
  request.setContentType("text/html");
  HttpSession session = request.getSession(true);
  JspWriter out = response.getWriter();
  out.println("<H1>A Random Number</H1>");
  out.println(Math.random());
  ...
}
```

Example Using JSP Expressions

```
<BODY>
<H2>JSP Expressions</H2>
<UL>
  <LI>Current time: <%= new java.util.Date() %>
  <LI>Your hostname: <%= request.getRemoteHost() %>
  <LI>Your session ID: <%= session.getId() %>
  <LI>The <CODE>testParam</CODE> form parameter:
      <%= request.getParameter("testParam") %>
</UL>
</BODY>
```



Predefined Variables (Implicit Objects)

- They are created automatically when a web server processes a JSP page.
  - **request**: The HttpServletRequest (1st arg to doGet)
  - **response**: The HttpServletResponse (2nd arg to doGet)
  - **session**
    - The HttpSession associated with the request (unless disabled with the session attribute of the page directive)
  - **out**
    - The stream (of type JspWriter) used to send output to the client
  - **application**
    - The ServletContext (for sharing data) as obtained via getServletConfig().getContext().
  - **page, pageContext, config, exception**

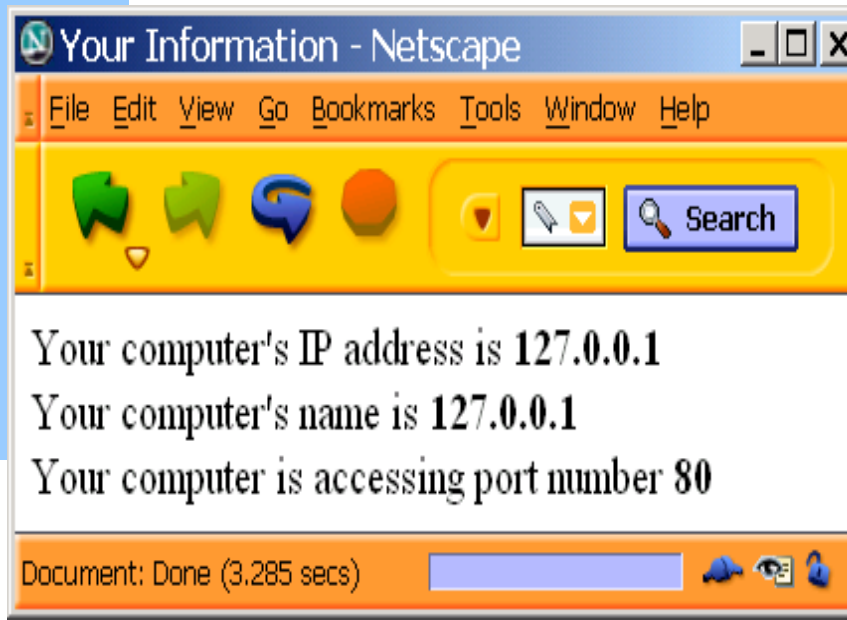
Implicit objects – Class files

- application: javax.servlet.ServletContext
- config: javax.servlet.ServletConfig
- exception: java.lang.Throwable
- out: javax.servlet.jsp.JspWriter
- page: java.lang.Object
- pageContext: javax.servlet.jsp.PageContext
- request: javax.servlet.ServletRequest
- response: javax.servlet.ServletResponse
- session: javax.servlet.http.HttpSession

Access Client Information

- The getRemoteHost method of the request object allows a JSP to retrieve the name of a client computer.

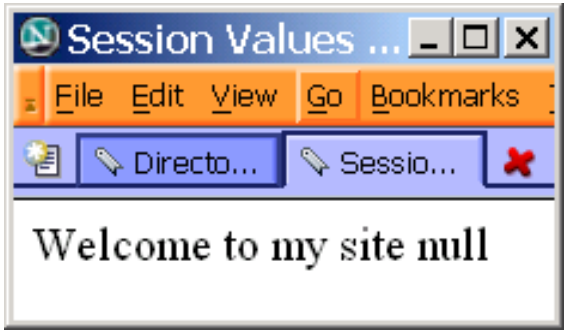
```
<html><head>
<title>Your Information</title>
</head><body>
Your computer's IP address is
<b><%= request.getRemoteAddr() %></b>
<br>Your computer's name is
<b><%= request.getRemoteHost() %></b>
<br>Your computer is accessing port
   number
<b><%= request.getServerPort() %></b>
</body></html>
```

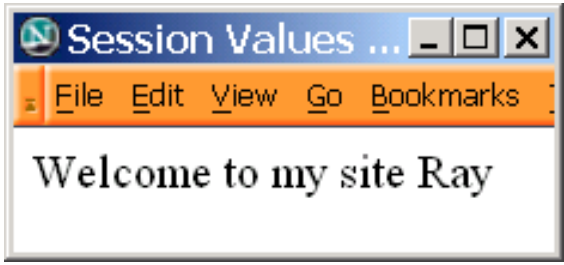


Working with Session object

- The session object has many useful methods that can alter or obtain information about the current session.
  - **setMaxInactiveInterval(second)**

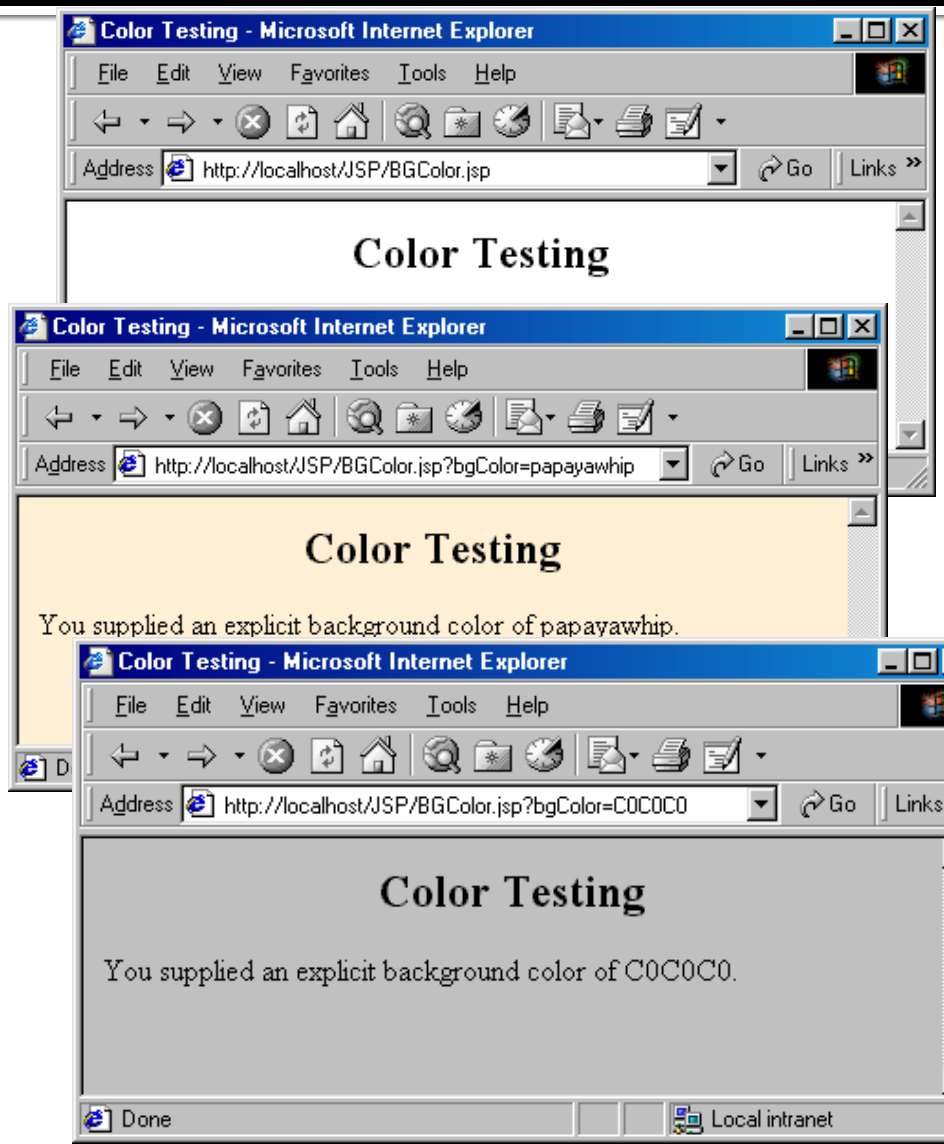
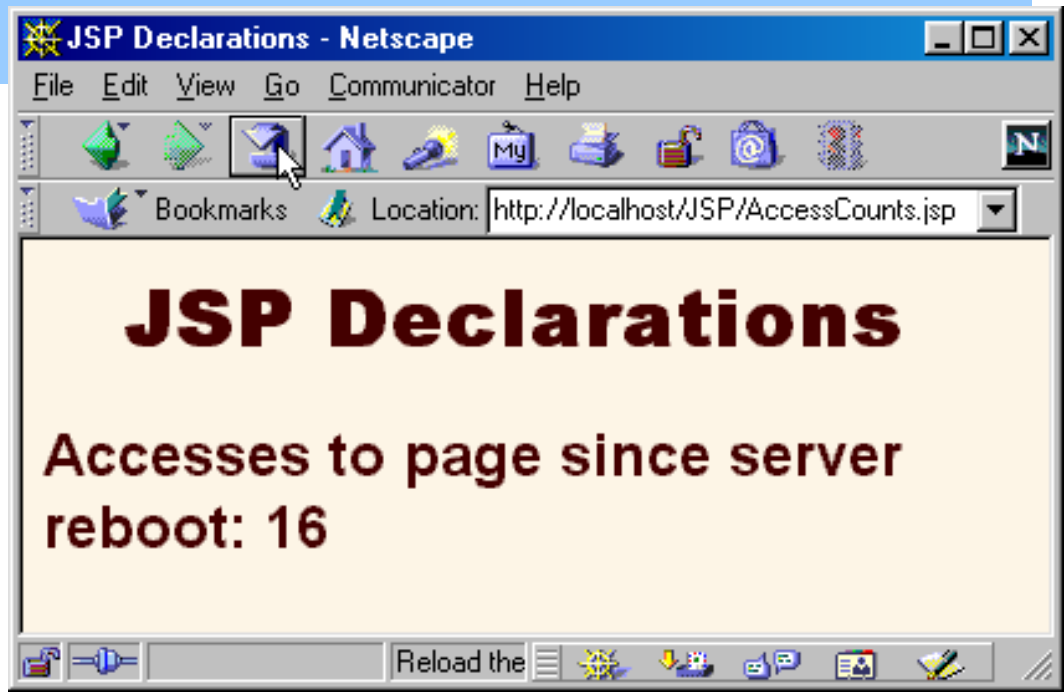
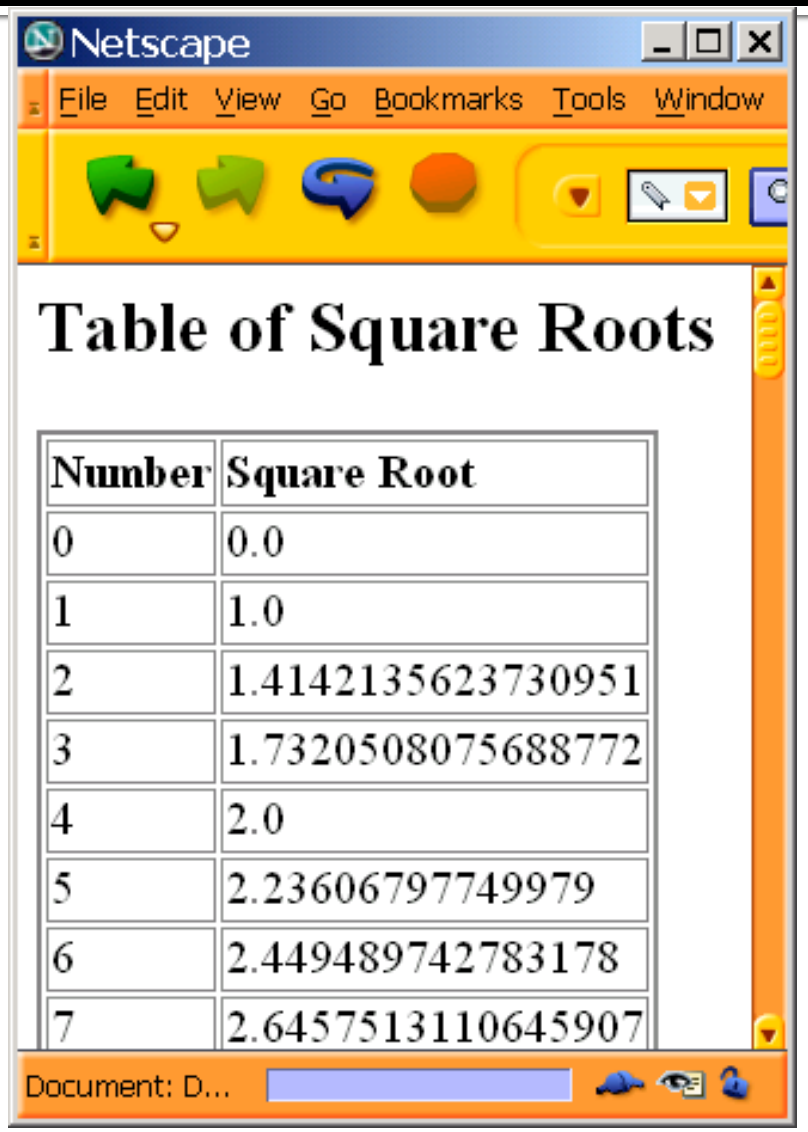
```
<html><head>
<title>Session Values</title>
</head><body>
<%
session.setMaxInactiveInterval(10);
String name = (String)
   session.getAttribute("username");
out.print("Welcome to my site " + name +
   "<br>");
%>
</body></html>
```

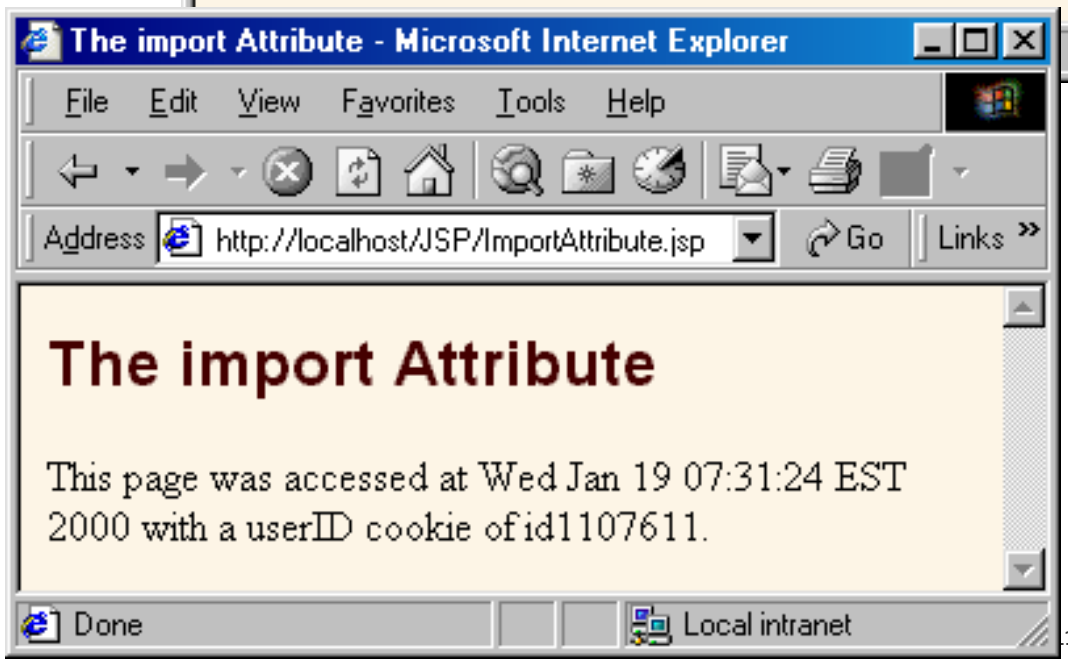
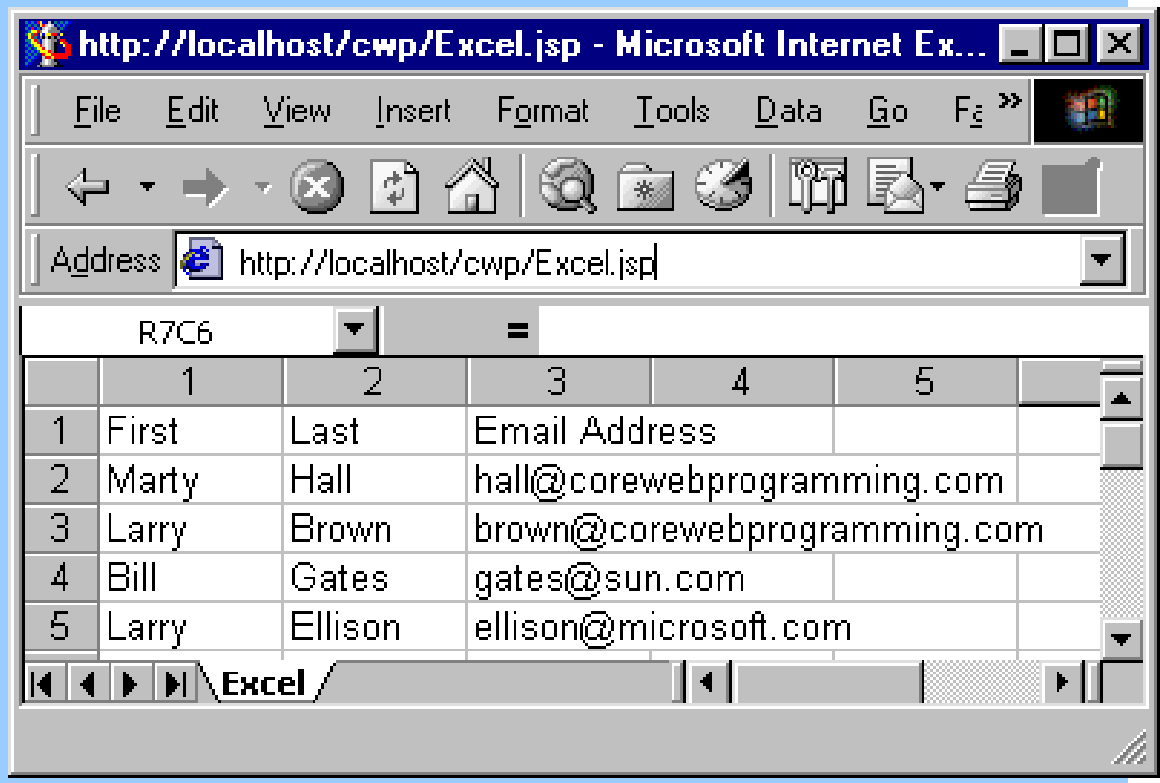
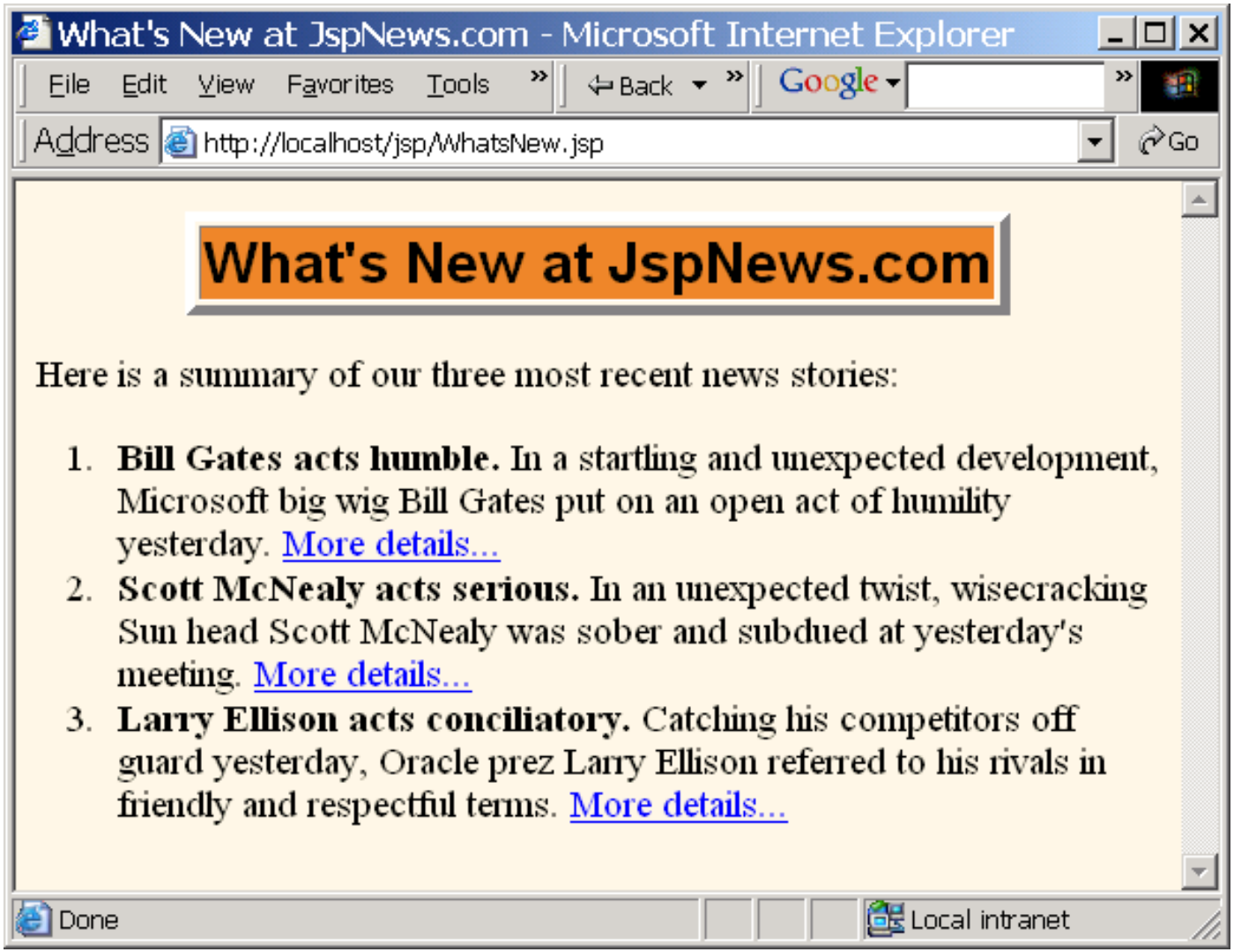
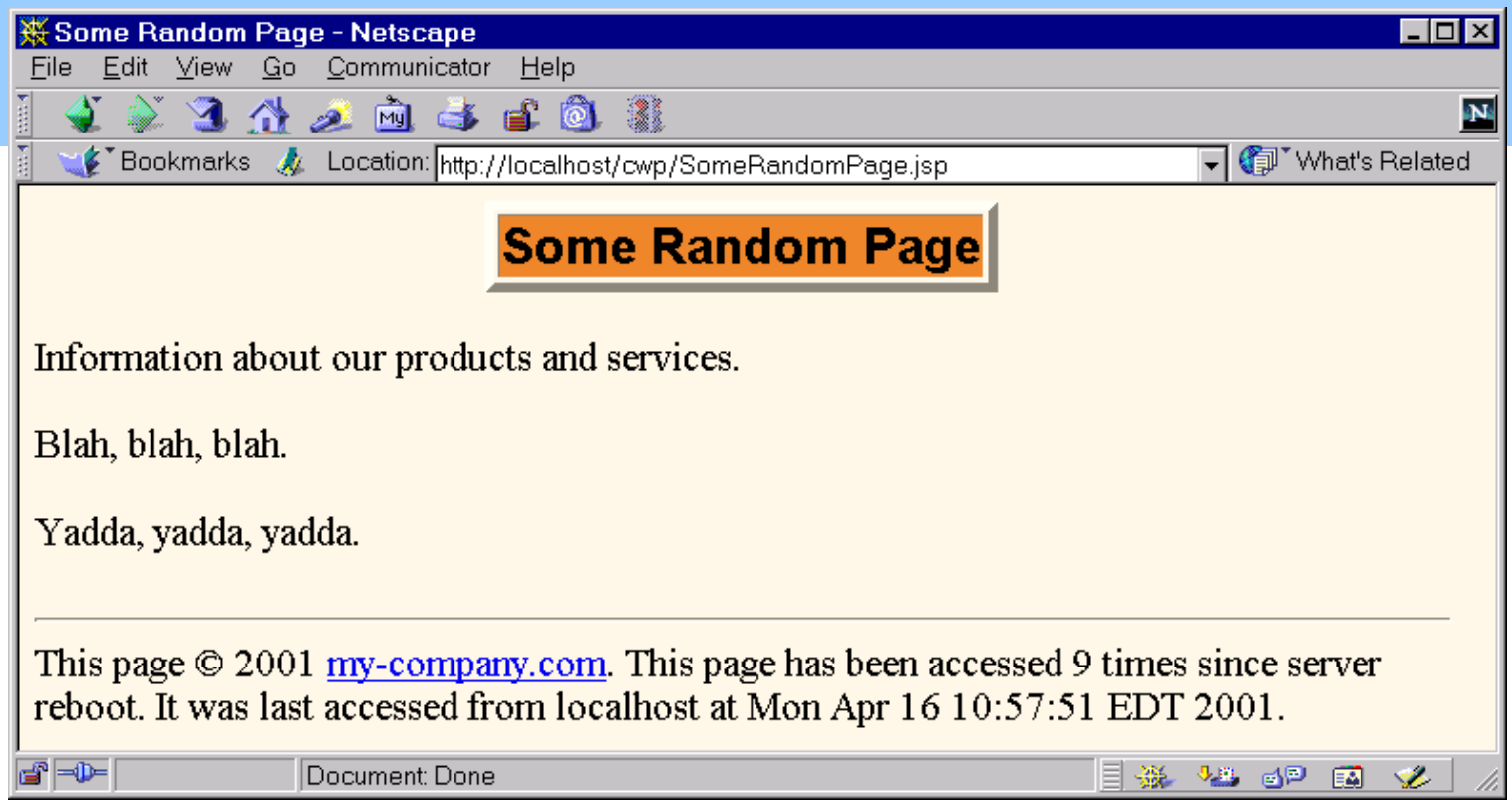




Scriptlets JSP: <% Code %>

JSP Scriptlets

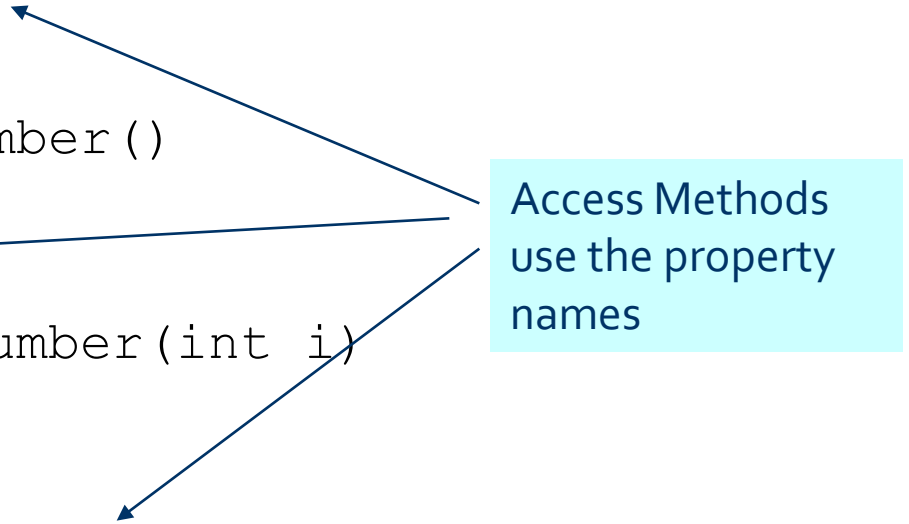
- Format: <% Java Code %>
- Result
  - Code is inserted verbatim into servlet's _jspService
- Example
  - ```
    <%
    String queryData = request.getQueryString();
    out.println("Attached GET data: " + queryData);
    %>
    ```
  - ```
    <% response.setContentType("text/plain"); %>
    ```
- XML-compatible syntax
  - ```
    <jsp:scriptlet>Java Code</jsp:scriptlet>
    ```
    ```

JSP/Servlet Correspondence

- Original JSP

```
<%= foo() %>
<% bar(); %>
```

- Possible resulting servlet code

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
    throws ServletException, IOException {
  request.setContentType("text/html");
  HttpSession session = request.getSession(true);
  JspWriter out = response.getWriter();
  out.println(foo());
  bar();
  ...
}
```

JSP Scriptlets Example

```
<%
for (int i=100; i>=0; i--)
{
%>
   <%= i %> bottles of beer on
   the wall.<br>
<%
}
%>
```

Example Using JSP Scriptlets

```
<HTML>
<HEAD>
  <TITLE>Color Testing</TITLE>
</HEAD>
<%
String bgColor =
 request.getParameter("bgColor");
boolean hasExplicitColor;
if (bgColor != null) {
  hasExplicitColor = true;
} else {
  hasExplicitColor = false;
  bgColor = "WHITE";
}
%>
<BODY BGCOLOR="<%= bgColor %>">
```



JSP declaration
: <%! Code %>

JSP Declarations

- Format
  - <%! Java Code %>
- Result
  - Code is inserted verbatim into servlet's class definition, outside of any existing methods
- Examples
  - <%! private int someField = 5; %>
  - <%! private void someMethod(…) {…} %>
- XML-compatible syntax
  - <jsp:declaration>Java Code</jsp:declaration>

Example Using JSP Declarations

```
…
<body>
<h1>JSP Declarations</h1>
<%! private int accessCount = 0; %>
<h2>Accesses to page since server reboot:
<%= ++accessCount %></h2>
</body></html>
```

- After 15 total visits by an arbitrary number of different clients



JSP Tags + HTML Tags

```
<h2>Table of Square Roots</h2>
<table border=2>
  <tr>
    <td><b>Number</b></td>
    <td><b>Square root</b></td>
  </tr>
  <%
  for (int n=0; n<=100; n++)
  {
  %>
    <tr>
      <td><%=n%></td>
      <td><%=Math.sqrt(n)%></td>
    </tr>
  <%
  }
  %>
</table>
```



JSP Directives

JSP Directives

- Affect the overall structure of the servlet
- Two possible forms for directives
  - <%@ directive attribute="value" %>
  - <%@ directive attribute1="value1"
    attribute2="value2"
    ....
    attributeN="valueN" %>
- There are three types of directives
  - Page, include, and taglib

Purpose of the page Directive

- Give high-level information about the servlet that will result from the JSP page
- Can control
  - Which classes are imported
  - What class the servlet extends
  - What MIME type is generated
  - How multithreading is handled
  - If the servlet participates in sessions
  - The size and behavior of the output buffer
  - What page handles unexpected errors

The import Attribute

- Format
  - <%@ page import="package.class" %>
  - <%@ page import="package.class1,...,package.classN" %>
- Purpose
  - Generate import statements at top of servlet
- Notes
  - Although JSP pages can be almost anywhere on server, classes used by JSP pages must be in normal servlet dirs
  - For Tomcat, this is install_dir/webapps/ROOT/WEB-INF/classes or …/ROOT/WEB-INF/classes/directoryMatchingPackage

Example of import Attribute

```
...
<BODY><H2>The import Attribute</H2>
<%-- JSP page directive --%>
<%@ page import="java.util.*,cwp.*" %>

<%-- JSP Declaration --%>
<%!
private String randomID() {
  int num = (int)(Math.random()*10000000.0);
  return("id" + num);
}
private final String NO_VALUE = "<I>No Value</I>";
%>
<%
Cookie[] cookies = request.getCookies();
String oldID = ServletUtilities.getCookieValue(cookies, "userID",
   NO_VALUE);
String newID;
if (oldID.equals(NO_VALUE)) { newID = randomID();
} else { newID = oldID; }
LongLivedCookie cookie = new LongLivedCookie("userID", newID);
response.addCookie(cookie);
%>
<%-- JSP Expressions --%>
This page was accessed at <%= new Date() %> with a userID
cookie of <%= oldID %>.
 </BODY></HTML>
```

Example of import Attribute

- First access

- Subsequent accesses



The contentType Attribute

- Format
  - <%@ page contentType="MIME-Type" %>
  - <%@ page contentType="MIME-Type;
    charset=Character-Set"%>
- Purpose
  - Specify the MIME type of the page generated by the servlet that results from the JSP page

```
First    Last     Email Address
Marty    Hall     hall@corewebprogramming.com
Larry    Brown    brown@corewebprogramming.com
Bill     Gates    gates@sun.com
Larry    Ellison  ellison@microsoft.com
```

```
<%@ page contentType="application/vnd.ms-excel" %>
<%-- There are tabs, not spaces, between columns. --%>
```

Generating Excel Spreadsheets



JSP Actions

- There are *seven* standard JSP actions.
  - Include, param, forward, plugin, …
  - Include action is similar to include directive.
  - You can add additional parameters to the existing request by using the param action.
  - The plugin action inserts object and embed tags (such as an applet) into the response to the client.
  - In the coming slides, we will talk about “include” and “plugin” actions.

Including Pages 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)

Including Pages: Example Code

```
...
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      What's New at JspNews.com</TABLE>
<P>
Here is a summary of our three 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" />
</OL>
</BODY></HTML>
```

Including Pages: 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 many 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
    - <%-- Navbar.jsp modified 3/1/02 --%>
      <%@ include file="Navbar.jsp" %>

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>
```



Packaging

- Web archive creation (war)
- Archive Deployement

Break Time – 15 minutes

.

JSP & Java beans

Intro

- A major problem with JSP is tendency to mix java code with HTML →

  -→ web designer write the HTML and programmer writes the java –maintenance issue if mixed

  -→JSP pages not easy to re-use if both presentation code <html> and business processing code (java) is together –cannot "re-use" business logic

Intro

- Java beans provide another mechanism to hide java code from the JSP page
- Java beans can be easily reused – can create libraries of reusable components for plugging into JSP
- Java beans follow certain rules and guidelines in order to enable their use as components - **design patterns**

Background: What Are Beans?

- Java classes that follow certain conventions
  - Must have a zero-argument (empty) constructor
    - Either explicitly include the constructor
    - OR don't have any constructor
  - Should have no public instance variables (fields)
  - Persistent values should be accessed through methods called **`getXxx`** and **`setXxx`**
    - If an object has a method named **`getTitle`** that returns a value of type **`String`** , the object is said to have a **`String`** property named **`title`**
    - boolean properties use **`isXxx`** instead of **`getXxx`**
- For more info on beans, see http://java.sun.com/beans/docs/

Java bean properties

- E.g. a java bean called Userbean.java might have properties such as name, address, age..

- Java bean properties can be set
and retrieved with **setter** and **getter** methods (called accessor methods)

- E.g. a java bean with a property `colour`

- To read or get the value property `colour,` use a get method called `getColour()`

- To update or set the value property colour, use a get method called `setColour()`

Design patterns for java beans

- Zero argument constructor
- `setPropertyName()` methods – consisting of "set" plus property name with each word capitalised
- `getPropertyName()` methods – consisting of "get" plus property name with each word capitalised
- `is`*`PropertyName`*`()` method – returns a boolean value
- Property (e.g. firstName) always start with a lower case letter

Simple bean example

```
package com;

public class SimpleBean implements java.io.Serializable
   {    /* Properties .. Always private*/
  private String username = null;

  private int luckynumber = 0;

   /* Empty Constructor */
   public SimpleBean() {}

   /* Getter and Setter Methods */
    public String getUsername()
    {
       return username;
    }
```

Allows an object to be saved to disk for re-use later

Properties are private

Access Methods use the property names

Simple bean (cont.)

```
public void setUsername(String s)
 {
    username = s;
 }
 public int getLuckynumber()
 {
    return luckynumber;
 }
 public void setLuckynumber(int i)
{
    luckynumber = i;
 }
}
```



JSP Actions for a bean

- JSP container support several JSP actions to use beans within a JSP page
- <jsp:useBean> identifies the bean to use
- <jsp:setProperty> sets bean properties
- <jsp:getProperty> retrieves bean properties

Using Bean in JSP

- Syntax:

```
<jsp:useBean
  id="objectName" class="package.ClassName" />
```

- Purpose: Allow instantiation of classes without explicit Java syntax

- `<jsp:useBean id="book1" class="cwp.Book" />`
  is equivalent to
  `<% cwp.Book book1 = new cwp.Book(); %>`

- But **`useBean`** has two additional features
  - Simplifies setting fields based on incoming request parameters
  - Easier to share beans within a web application

Accessing Bean Properties

- Syntax:

```
<jsp:getProperty
    name="objectName" property="propertyName" />
```

- Purpose: Allow access to bean properties (i.e., calling **`getXxx`** methods) without explicit Java code

- `<jsp:getProperty name="book1" property="title" />`
  is equivalent to

```
<%= book1.getTitle() %>
```

Modifying Bean Properties

- Syntax:

```
<jsp:setProperty name="objectName"
   property="propertyName" value="value" />
```

- Purpose: Allow setting of bean properties (i.e., calling **`setXxx`** methods) without explicit Java code

- ```
  <jsp:setProperty name="book1"
     property="title" value="Servlets and JSP" />
  ```
  is equivalent to the following scriptlet
  ```
  <% book1.setTitle("Servlets and JSP"); %>
  ```
  ```

Example: StringBean Class

```
package cwp;
public class StringBean {
  private String message;

  public StringBean() { message = "No message specified"; }

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

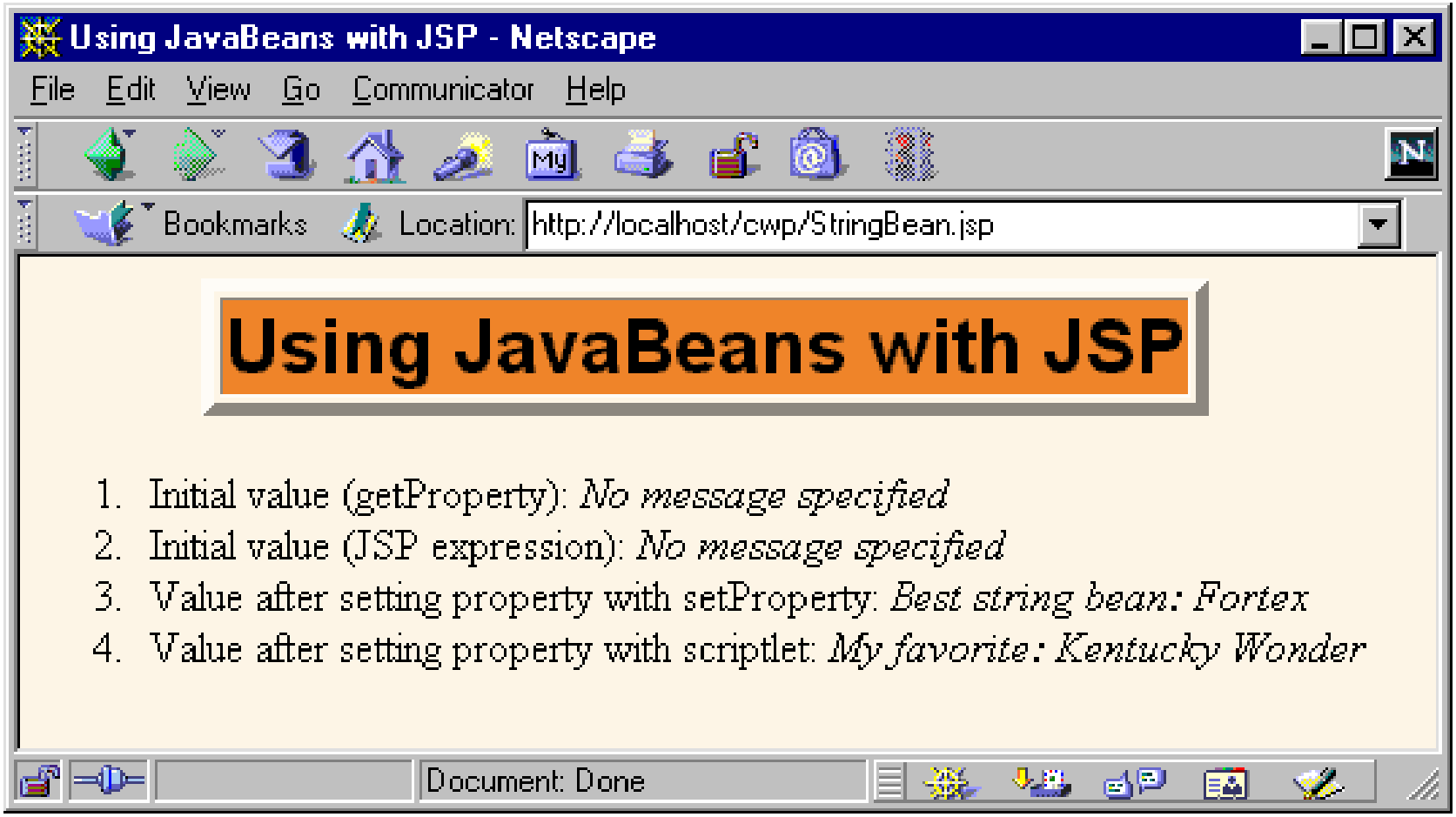
- Java Bean classes used in JSP pages must be kept in a package.

JSP Page that Uses StringBean

```
<jsp:useBean id="stringBean" class="cwp.StringBean" />
<ol>
<li>Initial value (getProperty):
    <i><jsp:getProperty name="stringBean"
                        property="message" /></i></li>
<li>Initial value (JSP expression):
    <i><%= stringBean.getMessage() %></i></li>
<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>
<li>
<% stringBean.setMessage("My favorite: Kentucky Wonder"); %>
    Value after setting property with scriptlet:
    <i><%= stringBean.getMessage() %></i></li>
</ol>
```

It is possible but not a good practice to mix Java Scriptlets and beans tags.

JSP Page that Uses StringBean (Output)



Associating Bean Properties with Request Parameters (Method 1)

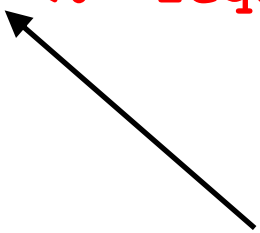
- Setting Bean Properties directly using scriptlets
- Syntax:

```
<jsp:setProperty … value='<%= expression %>' />
```

- Type of expression must match the type of the property
- If a property is a String, you can write
  -
```
<jsp:setProperty …
        value='<%= request.getParameter("…") %>' />
```
- If a property is not a String, you have to convert the string value to the appropriate type first before assigning the value using this method.

Setting Bean Properties Case 1: Explicit Conversion & Assignment

```
...
<jsp:useBean id="entry"
             class="cwp.SaleEntry" />

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



Can use either single or double quotes.
If using single quote, value can contains double quotes, and vice versa. Otherwise need to use \' and \"

Setting Bean Properties Case 1: Explicit Conversion & Assignment

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

Case 2: Associating Individual Properties with Input Parameters

```
<jsp:useBean id="entry"
             class="cwp.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" />
```

- The `param` attribute indicates that:
  - Value should come from specified request parameters
  - Automatic type conversion is performed
  - If parameter value is null, the corresponding property is not set.

Case 3: Associating *All* Properties with Input Parameters

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

- "*" in `property` attribute indicates that:
  - Value should come from request parameter whose name matches property name
  - Automatic type conversion is performed
  - If parameter value is null, the corresponding property is not set.

Sharing Beans

- You can use **scope** attribute to specify where bean is stored/accessible
  - **<jsp:useBean id="…" class="…" scope="…" />**
  - Bean still also bound to local variable in **_jspService**
- Bean is conditionally created
  - jsp:useBean results in new bean object only if no bean with same ID and scope can be found
  - If a bean with same the ID and scope is found, the bean is bound to the variable referenced by ID

Scope of a java bean

A Java bean is only created when it is needed. Like all objects, has a concept of scope:

- **Page scope** (default) – new bean for every page view (bean placed in the PageContext object)
- **Request scope** – new bean created for every new client request to the page

  (bean is placed in the ServletRequest object)
- **Session scope** – new bean on first request for new user session (bean is placed in the HttpSession object)
- **Application scope** – new bean first request to the application (bean is placed in the webapplication's ServletContext)

Conditional Bean Operations

- Bean properties conditionally set
  - `<jsp:useBean …>`

    `<jsp:setProperty … >`

    …

    `</jsp:useBean>`
  - The `jsp:setProperty` statements are executed only if a new bean is created, not if an existing bean is found

Review: Using JavaBeans Components with JSP

- Benefits of jsp:useBean
  - Hides the Java programming language syntax
  - Makes it easier to associate request parameters with 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. getXxx call) into output
- jsp:setProperty
  - Sets bean property (i.e. passes value to setXxx)

JSP that uses a java bean

```
<%@ page language="java" contentType="text/html" %>
<%@page import="com.SimpleBean"%>

<html>
  <body bgcolor="white">
    <jsp:useBean  id="simpleBean" class="com.SimpleBean"
scope="session"/>
      <jsp:setProperty name="simpleBean" property="*" />

    The following information was saved:
    <ul>
      <li>User Name: <jsp:getProperty
                       name="simpleBean" property="username" />

      <li>Lucky number: <jsp:getProperty
                       name="simpleBean" property="luckynumber" />
    </ul>
       </body>
</html>
```

Package name

Example: Capturing user info using a bean

- A form that takes name, birth date, email address, gender, lucky number, favourite foods (radio box..) – and displays what the user has entered when the "submit" button is pressed…

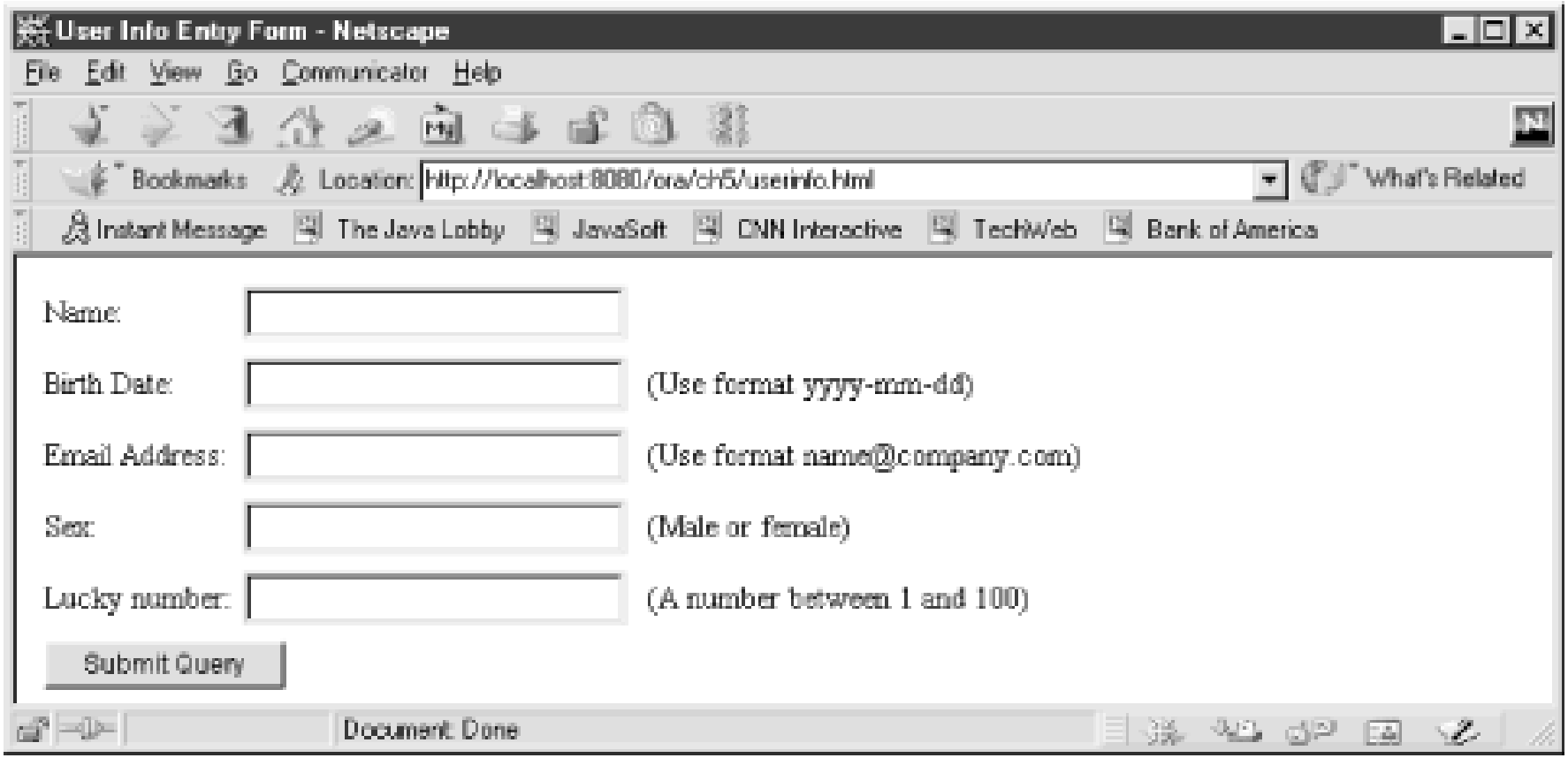
  Java bean (e.g. `Userinfo.java`) is used to 1) capture and hold user info to pass around the application. and… 2) validate user info….

Example: User info Bean properties

Table 5-2: Properties for UserInfoBean			
Property Name	**Java Type**	**Access**	**Description**
userName	`String`	read/write	The user's full name
birthDate	`String`	read/write	The user's birth date in the format *yyyy-mm-dd* (e.g., `2000-07-07`)
emailAddr	`String`	read/write	The user's email address in the format *name@company.com*
gender	`String`	read/write	The user's sex (`male` or `female`)
luckyNumber	`String`	read/write	The user's lucky number (between 1 and 100)
valid	`boolean`	read	`true` if the current values of all properties are valid, `false` otherwise

http://www.oreilly.com/catalog/jserverpages/chapter/ch05.html#39961

Example: Capturing User Info



Example: Capturing user info with a Bean - Sample form

```
<%@ page contentType="text/html" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
  <head>
    <title>User Info Entry Form</title>
  </head>
  <body bgcolor="white">
    <form action="input_bean.jsp" method="post">
      <table>
        <tr>
          <td>Name:</td>
          <td>
            <input type="text" name="userName">
          </td>
        </tr>
        <tr>
          <td>Birth Date:</td>
          <td>
            <input type="text" name="birthDate">
          </td>
          <td>(Use format yyyy-mm-dd)</td>
        </tr>
```

Various Input fields on form..

Example: Capturing user info with a Bean: sample form

```
<td>Email Address:</td>
        <td>
          <input type="text" name="emailAddr">
        </td>
        <td>(Use format name@company.com)</td>
      </tr>
      <tr>
        <td>Sex:</td>
        <td>
          <input type="radio" name="gender" value="m" checked>Male<br>
          <input type="radio" name="gender" value="f">Female
        </td>
      </tr>
      <tr>
        <td>Lucky number:</td>
        <td>
          <input type="text" name="luckyNumber">
        </td>
        <td>(A number between 1 and 100)</td>
      </tr>
      etc..
```

Various Input fields on form..

Example: Capturing user info with a Bean

input_bean.jsp: stores the data into a bean and displays the bean properties

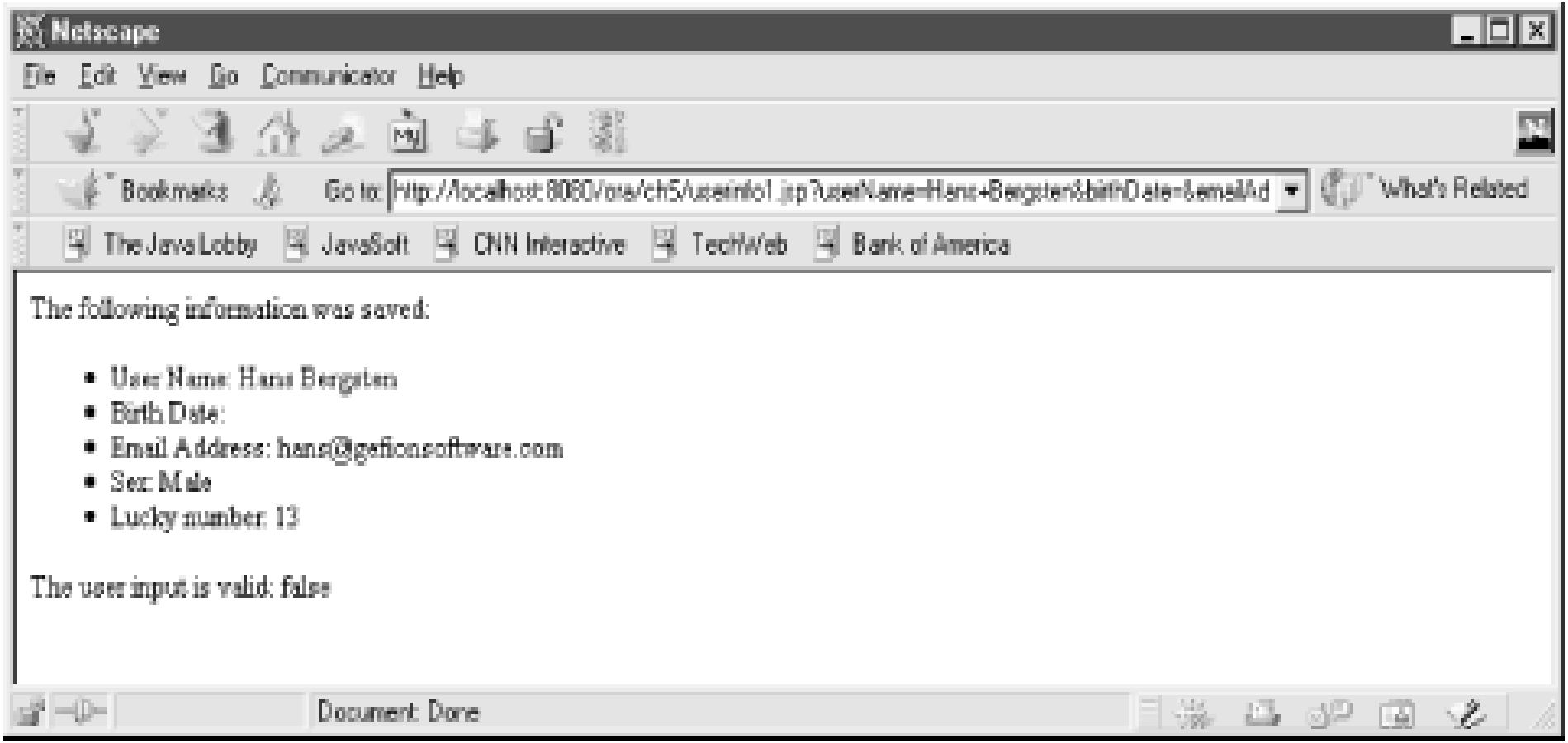
```
<%@ page language="java" contentType="text/html" %>
<html>
  <body bgcolor="white">
    <jsp:useBean
      id="userInfo"
      class="com.ora.jsp.beans.userinfo.UserInfoBean">
      <jsp:setProperty name="userInfo" property="*" />
    </jsp:useBean>

    The following information was saved:
    <ul>
      <li>User Name: <jsp:getProperty
                       name="userInfo" property="userName" />
      <li>Birth Date: <jsp:getProperty
                        name="userInfo" property="birthDate" />
      <li>Email Address: <jsp:getProperty
                           name="userInfo" property="emailAddr" />
      <li>Sex: <jsp:getProperty
                 name="userInfo" property="sex" />
      <li>Lucky number: <jsp:getProperty
                          name="userInfo" property="luckyNumber" />
    </ul>
    The user input is valid: <jsp:getProperty
                               name="userInfo" property="valid" />
```

Create a bean

Assign the properties in the bean to the values in the form.. Quick way using *

Output the values of the bean properties – using bean ID and property name

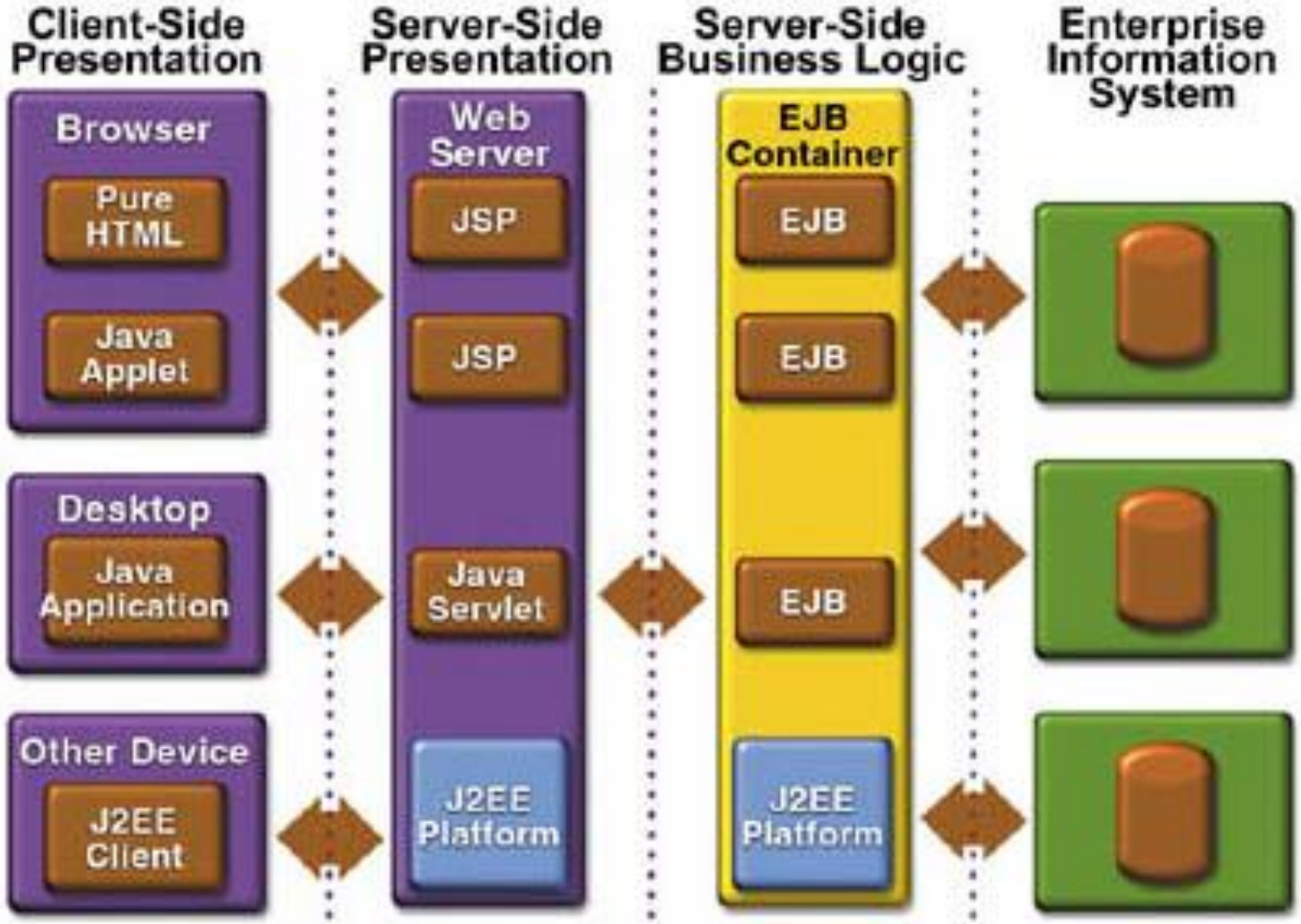
Example: Output of JSP is:



Example: Bean breakdown

- Userinfo bean will contain getter and setter methods for each of the five attribute (name, date of birth, etc.
- To include validation – can simple add methods into the bean to do validation of each of the attributes
  - This removes complexity from the JSP page
  - Gives re-usable functionality via the java bean (e.g. email address difficult to validate in a JSP page without using scriptlets)
  - Maintainance easy - If Validation rule changes – only change once in the java bean –

MVC / J2EE


Client-Side Presentation
Server-Side Presentation
Server-Side Business Logic
Enterprise Information System
Browser
Pure HTML
Java Applet
Desktop
Java Application
Other Device
J2EE Client
Web Server
JSP
JSP
Java Servlet
J2EE Platform
EJB Container
EJB
EJB
EJB
J2EE Platform

J2EE and Design Patterns

- The J2EE architecture is built to enable component developers to use a **Model View Controller (MVC)** Design Pattern.

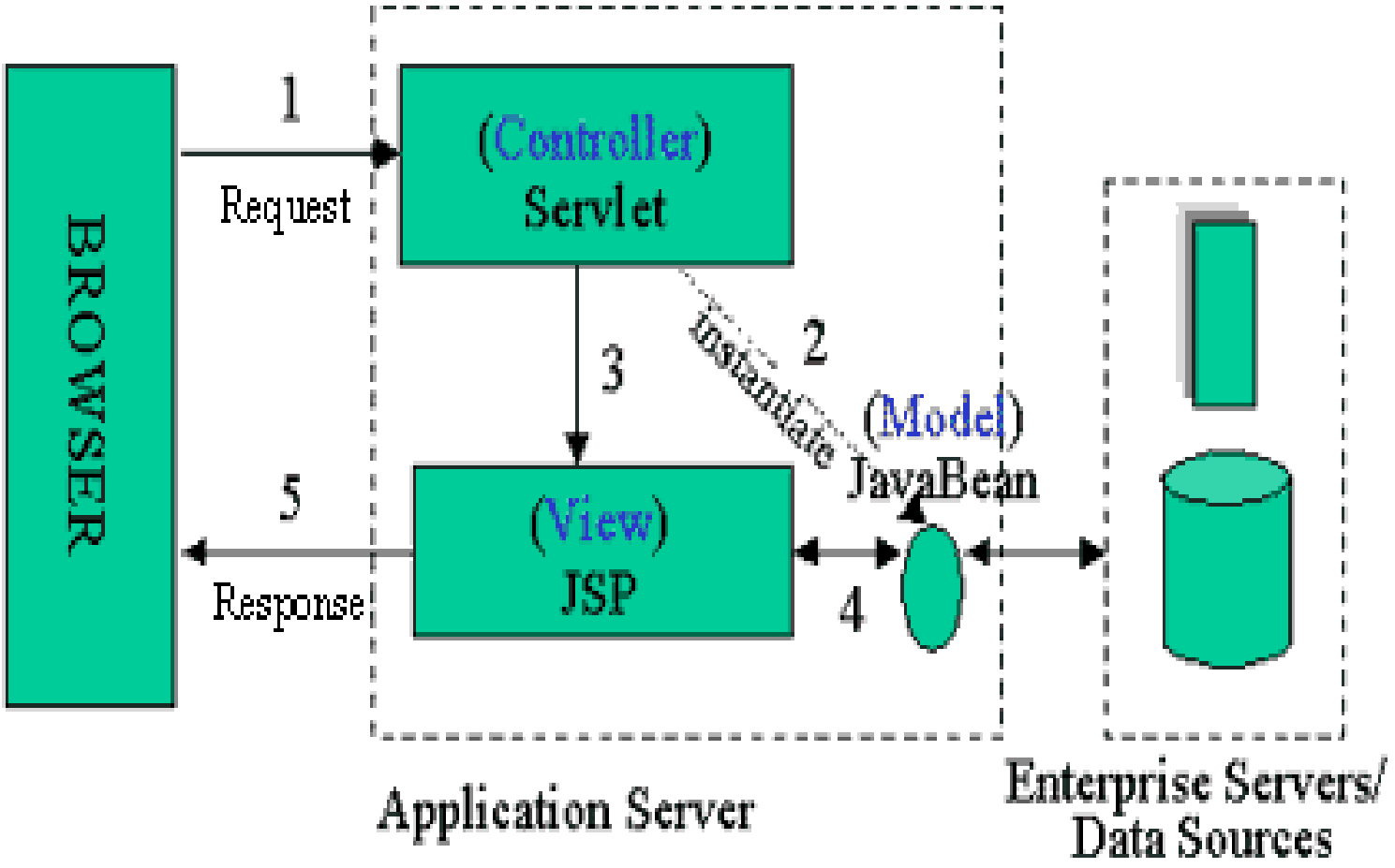
Details of MVC Design Pattern

- **Name** (essence of the pattern)
  - Model View Controller MVC
- **Context** (where does this problem occur)
  - MVC is an architectural pattern that is used when developing interactive application such as a shopping cart on the Internet.
- **Problem** (definition of the reoccurring difficulty)
  - User interfaces change often, especially on the internet where look-and-feel is a competitive issue. Also, the same information is presented in different ways. The core business logic and data is stable.

MVC continued

- **Solution** (how do you solve the problem)
  - Use the software engineering principle of "separation of concerns" to divide the application into three areas:
    - **Model** encapsulates the core data and functionality
    - **View** encapsulates the presentation of the data there can be many views of the common data
    - **Controller** accepts input from the user and makes request from the model for the data to produce a new view.

MVC Structure for J2EE



MVC -Servlet

- Syntaxe in a servlet to run a JSP :

```
public void doPost (HttpServletRequest request,
                                        HttpServletResponse
   response)
{
   ServletContext context = getServletContext();
   //inherited from GenericServletRequest

   Dispatcher dispatcher =
       context.getRequestDispatcher("/myPage.jsp");

   dispatcher.forward(request, response);
}
```

MVC – Servlet-JSP

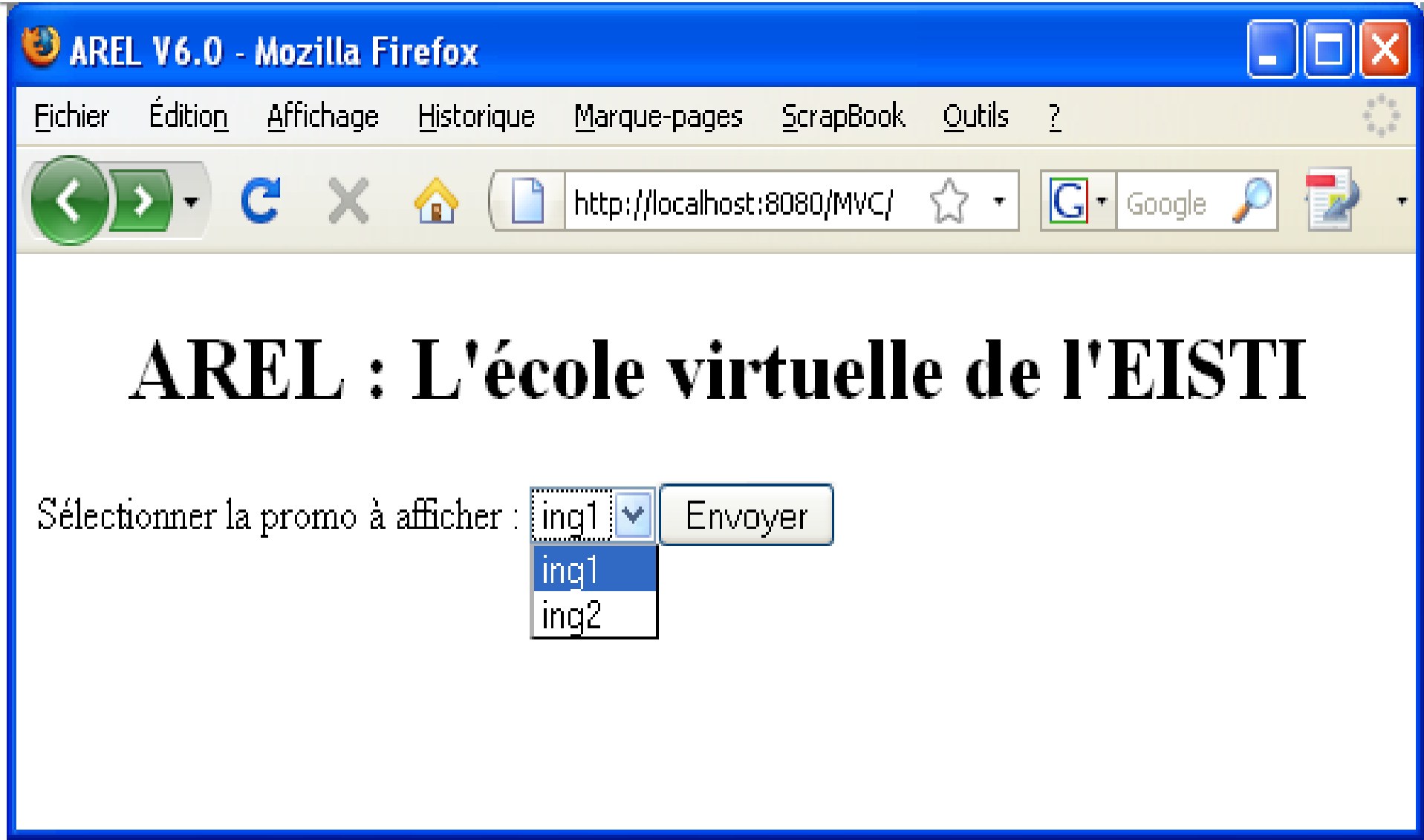
- The servlet can pass values to the JSP called by setAttribute ()

```
public void doPost(HttpServletRequest request, HttpServletResponse response) {
   // appelle les méthodes sur les objets métiers
   ArrayList theList= // un objet à passer
   // ajoute à la requête
   Request.setAttribute("ObjectName",  theList);
   ServletContext context = getServletContext();
   RequestDispatcher dispatcher =
                        context.getRequestDispatcher(« /jspToCall.jsp");
   dispatcher.forward(request, response);}
```

- JSP extracts the request objects through getAttribute ()

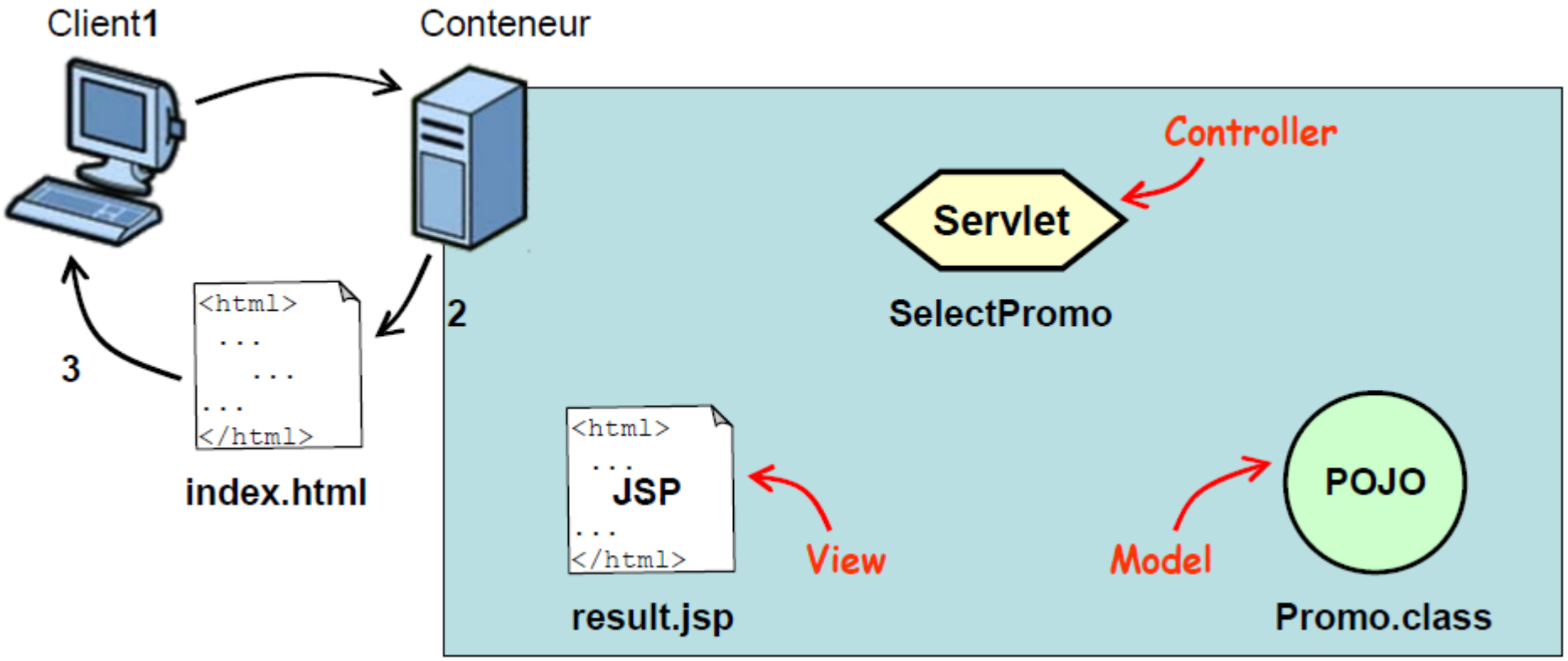
```
<% ArrayList theList = (ArrayList)request.getAttribute("nomDelObjet");
// now, use the ArrayList
%>
```

Example : Ex :ARELV6 –Promo List



MVC : Step1

- The client retrieves a form (index.html) to pass a query with parameters (1.2, puis3)

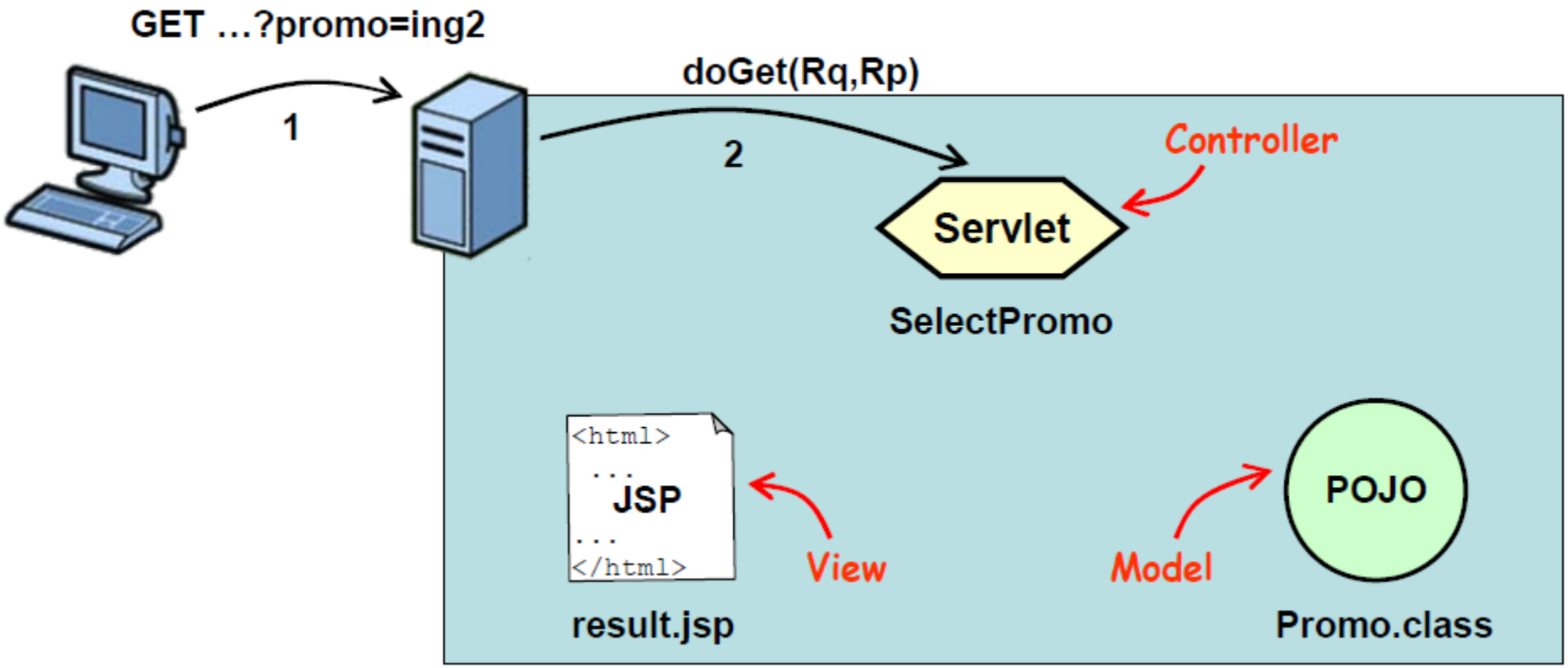


Form : index.html

```
<!DOCTYPEhtmlPUBLIC"-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type"
        content="text/html; charset=ISO-8859-1">
        <title>AREL V6.0</title>
    </head>
    <body>
        <h1 align="center"> AREL: L'école virtuelle de l'EISTI </h1>
        <form method="GET"  action="http://localhost:8080/MVC/SelectPromo">
            Sélectionner la promo à afficher:
            <select name="promo"  size="1">
                <option>ing1</option>
                <option>ing2</option>
            </select>
            <input type="SUBMIT"/>
        </form>
    </body>
</html>
```

MVC :Step2

1. The client sends its form (GET / POST parameters with)
2. The container transmits to the corresponding servlet (the controller)



Controller : SelectPromo.java

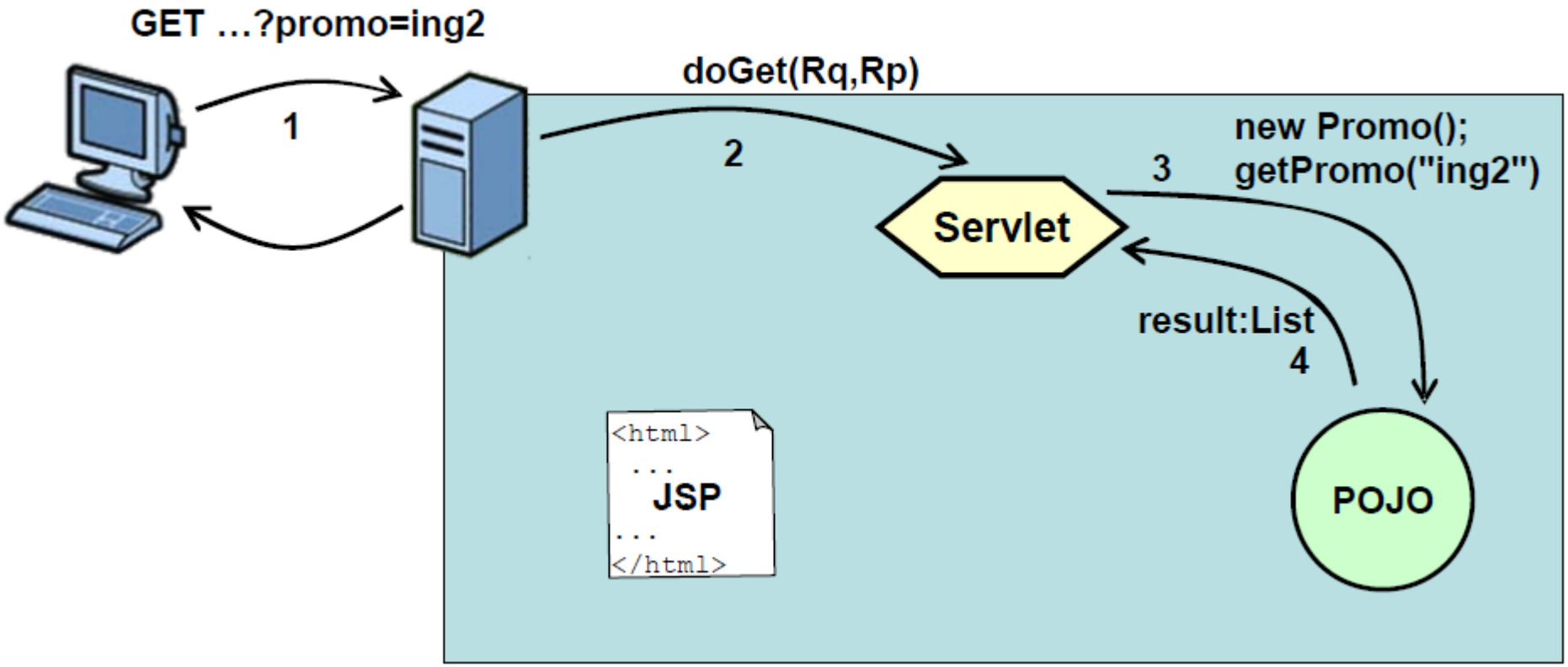
```
Package arel;
import...;
public class SelectPromo extends javax.servlet.http.HttpServlet
                                        Implements javax.servlet.Servlet{
   //...
   protected void doGet(HttpServletRequest request,
                                        HttpServletResponse response)
   Throws ServletException, IOException{
   String promoName = request.getParameter("promo");
   //...
}
}
```

Configuration : web.xml

```
<?xmlversion="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">
    <display-name>MVC</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
    <servlet>
        <description></description>
        <display-name>SelectPromo</display-name>
        <servlet-name>SelectPromo</servlet-name>
        <servlet-class>arel.SelectPromo</servlet-class>
    </servlet>
        <servlet-mapping>
        <servlet-name>SelectPromo</servlet-name>
        <url-pattern>/SelectPromo</url-pattern>
    </servlet-mapping>
</web-app>
```

MVC : Step3

- 3.The controller servlet queries the model on "ING2"
  4.The model returns to the controller the corresponding result

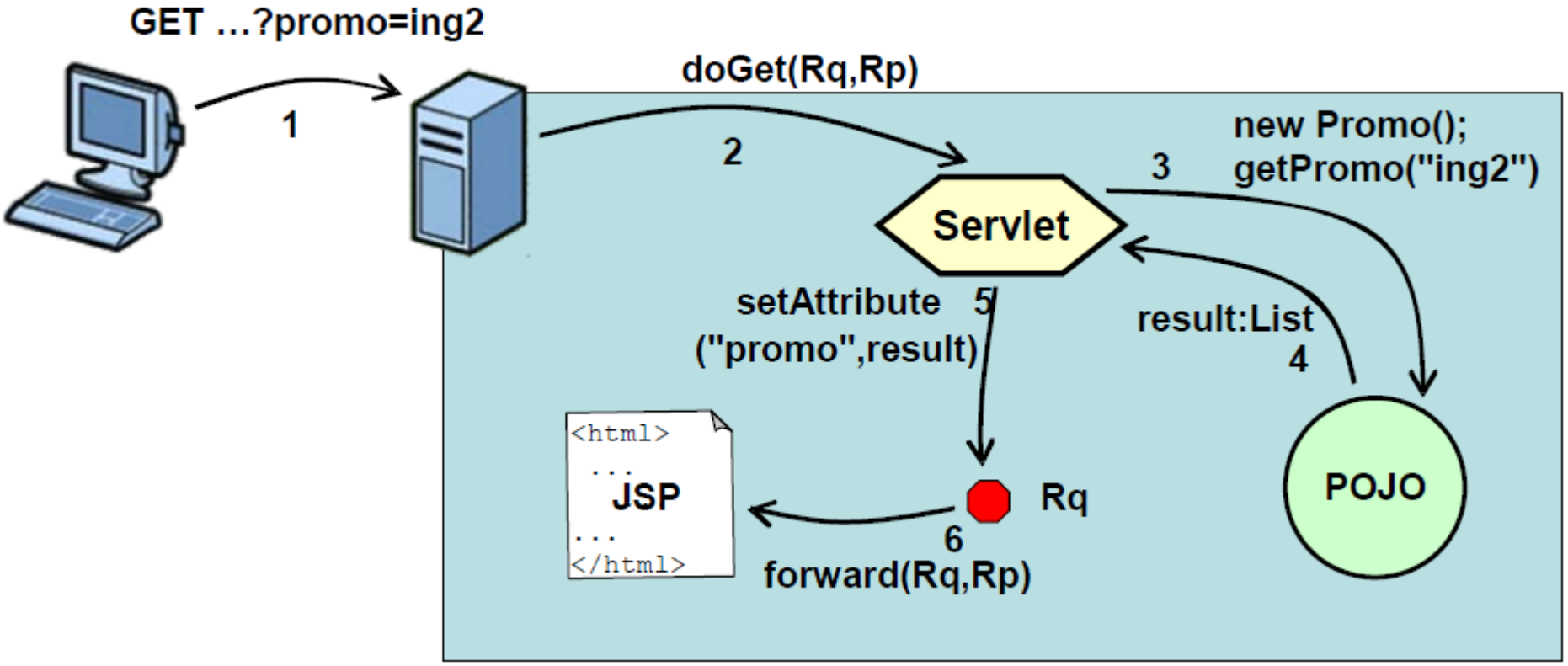


Model :Promo.java

```
Package arel;
import...;

public class Promo {

    Public  List<String> getPromo(String promo){

        List<String> promoList=new ArrayList<String>();
        if(promo.equals("ing1")){
            promoList.add("DonaldDuck");
            promoList.add("MinnieMouse");
            promoList.add("Pluto");//...
        }
        else if(promo.equals("ing2")){
            promoList.add("MickeyMouse");
            promoList.add("Daisy Duck");
            promoList.add("Goofy"); //...
        }
        else {return null;}
        Return promoList;
        } }
```

MVC :Step4

- 5.The controller uses the model data for its response
  6.The controller transmits its response to the view (JSP)

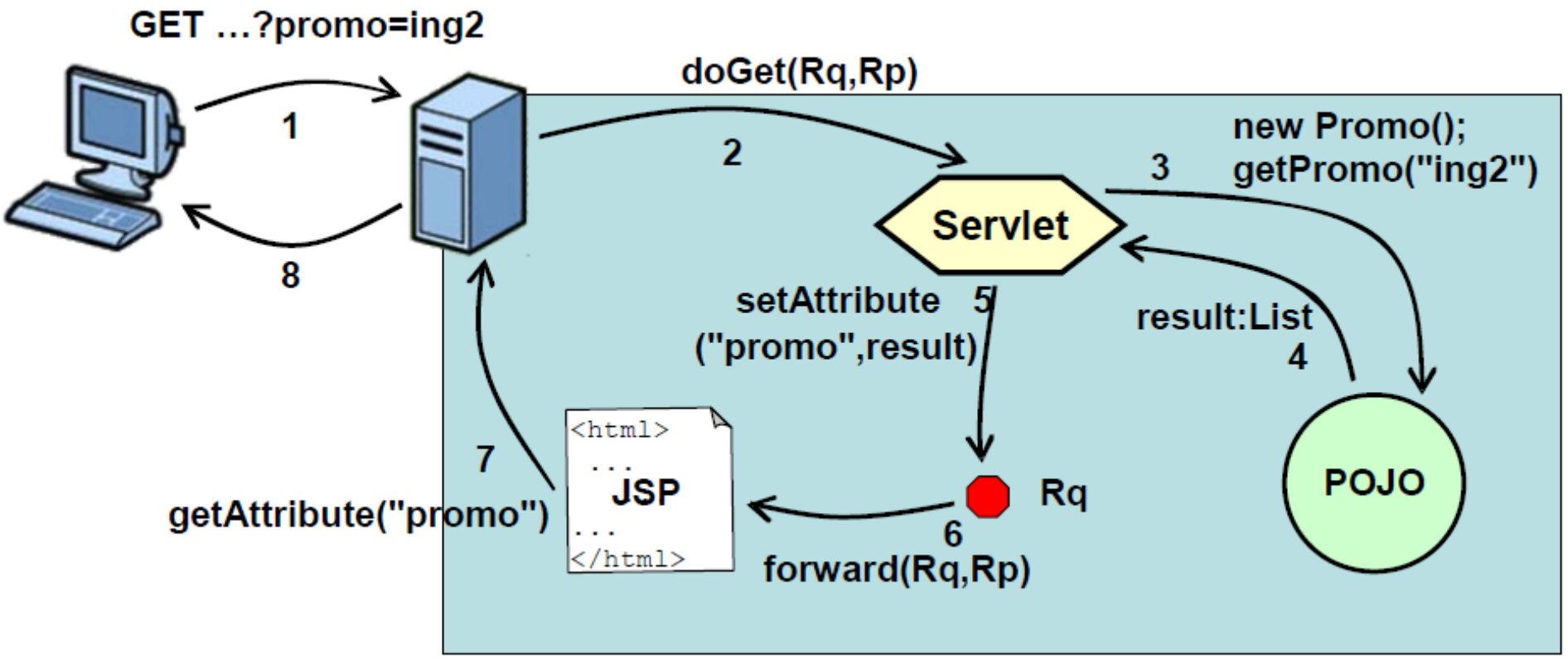


Controller : SelectPromo.java

```
Package arel;
import...;
public class SelectPromo extends javax.servlet.http.HttpServlet
                                 Implements javax.servlet.Servlet{
//...
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                 throwsServletException, IOException {

    String promoName = request.getParameter("promo");
    Promo promo = new Promo();
    List<String> result = promo.getPromo(promoName);

    request.setAttribute("promo", result);    // On ajoute l'attribut promo à la
    RequestDispatcher view = request.getRequestDispatcher("result.jsp");//requête


    view.forward(request, response); // On forward la requête à la JSP
    }
}
```

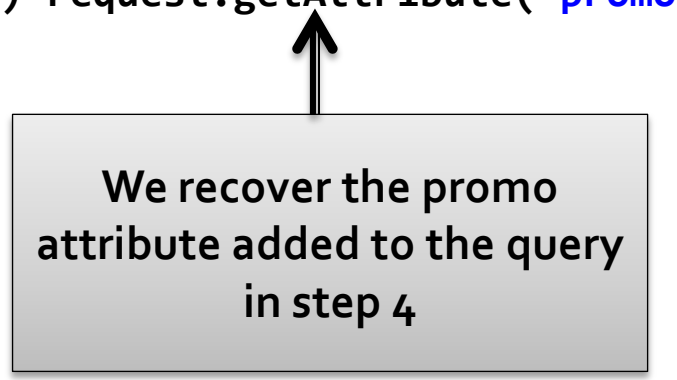
MVC :Step5

7.the JSP (view) processes the response sent by the controller
8.The resultant HTML page is received by the client



View :result.jsp

```
<%@ page import="java.util.*" %>
<%@ page language="java"  contentType="text/html;charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPEhtmlPUBLIC"-//W3C//DTDHTML4.01Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type"
        content="text/html;charset=ISO-8859-1">
        <title>Result</title>
    </head>
    <body>
        <%
        List<String> promoList = (List<String>) request.getAttribute("promo");
        Iteratorit=promoList.iterator();
        while (it.hasNext()){
        out.print("<br/>"+it.next());
        }
        %>
    </body>
</html>
```



Ex :ARELV6 –liste des promos