JSP and JDBC

- JSP's and Scope
- A Shopping cart application using JSP and JavaBeans
- A Simple JSP/JDBC Example
- A JSP/JDBC Example using connection pooling

Much of this lecture is from a book entitled "Pure JSP" by Goodwill published by SAMS

Page Scope

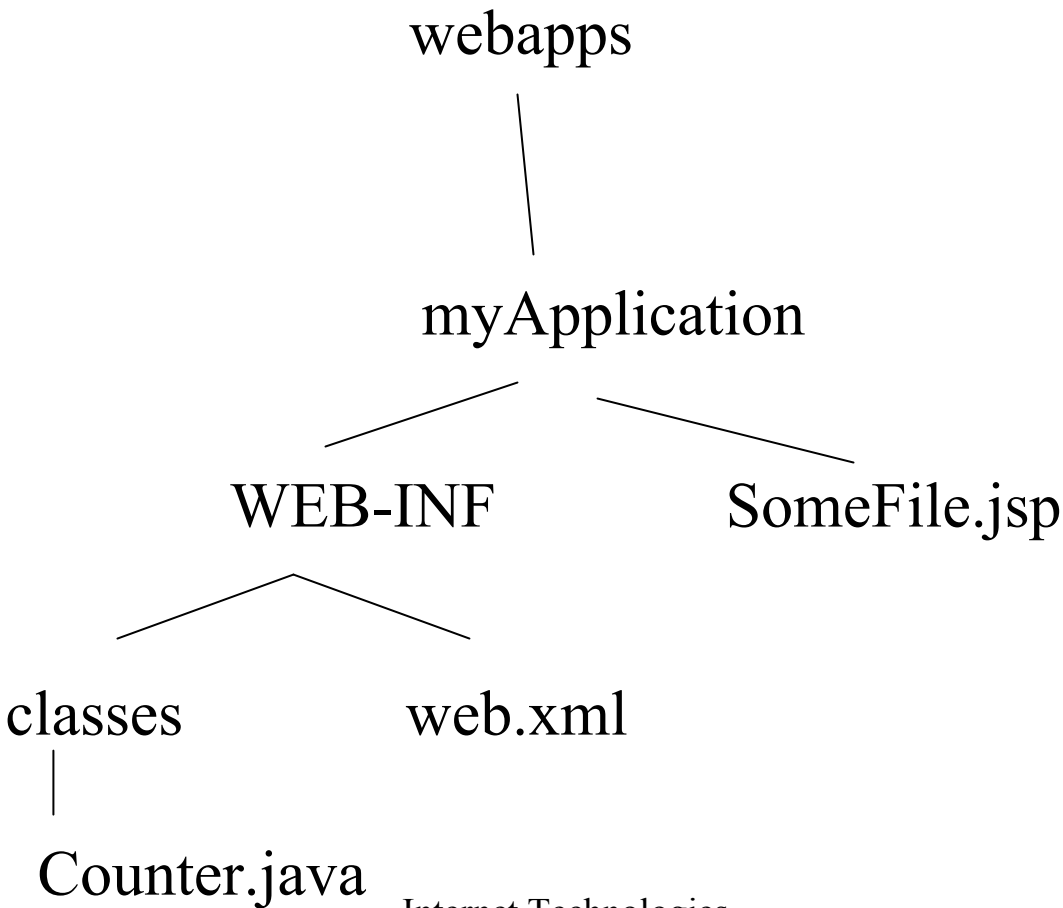
Beans with page scope are accessible only within the page where they were created.

A bean with page-level scope is not persistent between requests or outside the page

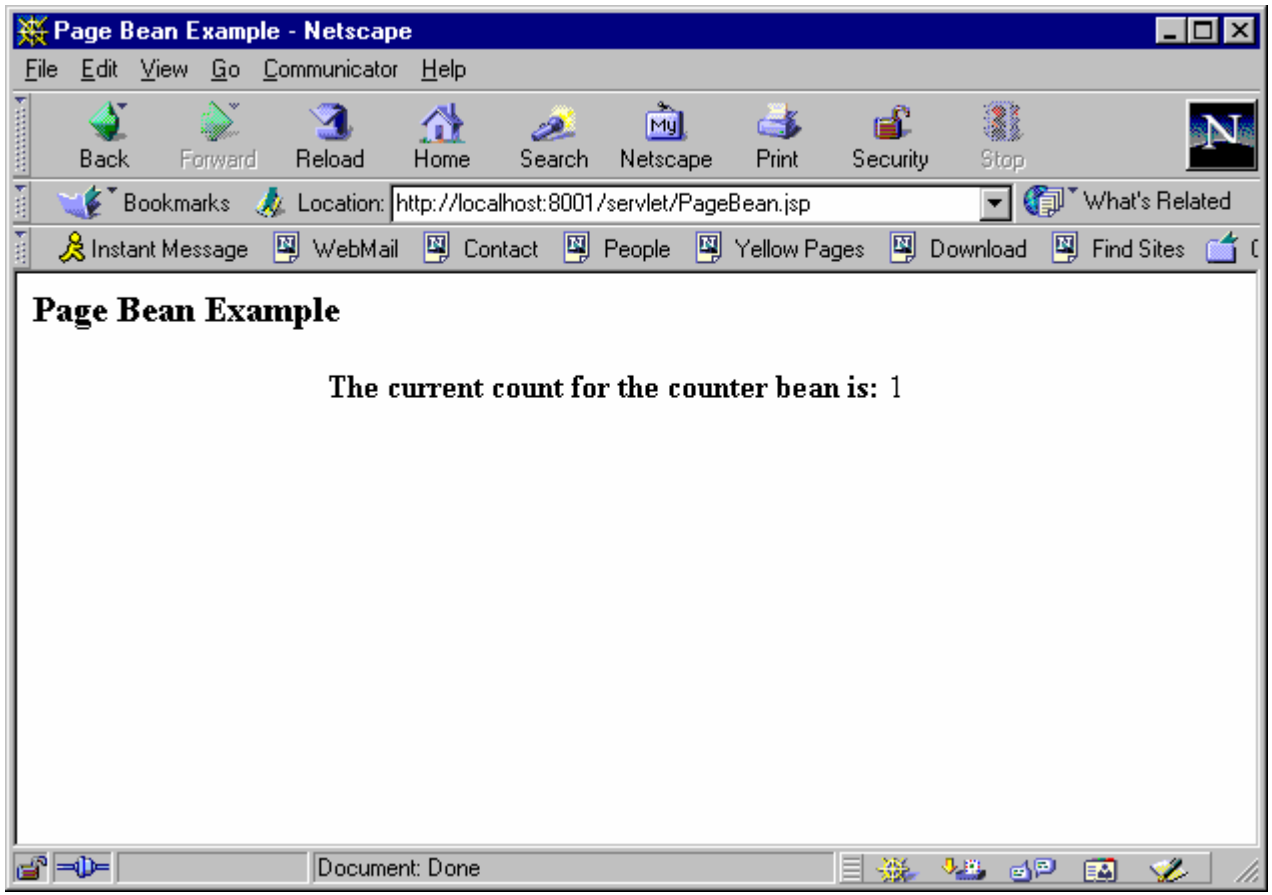
Page Scope Example

```
/* A simple bean that counts visits. */
import java.io.*;

public class Counter implements Serializable  {
    private int count = 1;
    public Counter() {}
    public int getCount() { return count++; }
    public void setCount(int c) { count = c; }
}
```

Under Tomcat



```
<%-- Use the Counter bean with page scope. --%>
<%-- The Counter class must be imported. Its in the
WEB-INF/classes directory --%>

<%@ page import="Counter" %>
<jsp:useBean id = "ctr" scope = "page" class = "Counter" />
<html>
  <head>
    <title>Page Bean Example</title>
  </head>
  <body>
    <h3>Page Bean Example </h3>
    <center>
      <b>The current count for the counter bean is: </b>
      <jsp:getProperty name = "ctr" property ="count" />
    </center>
  </body>
</html>
```



The count never changes.

One Page May Call Another

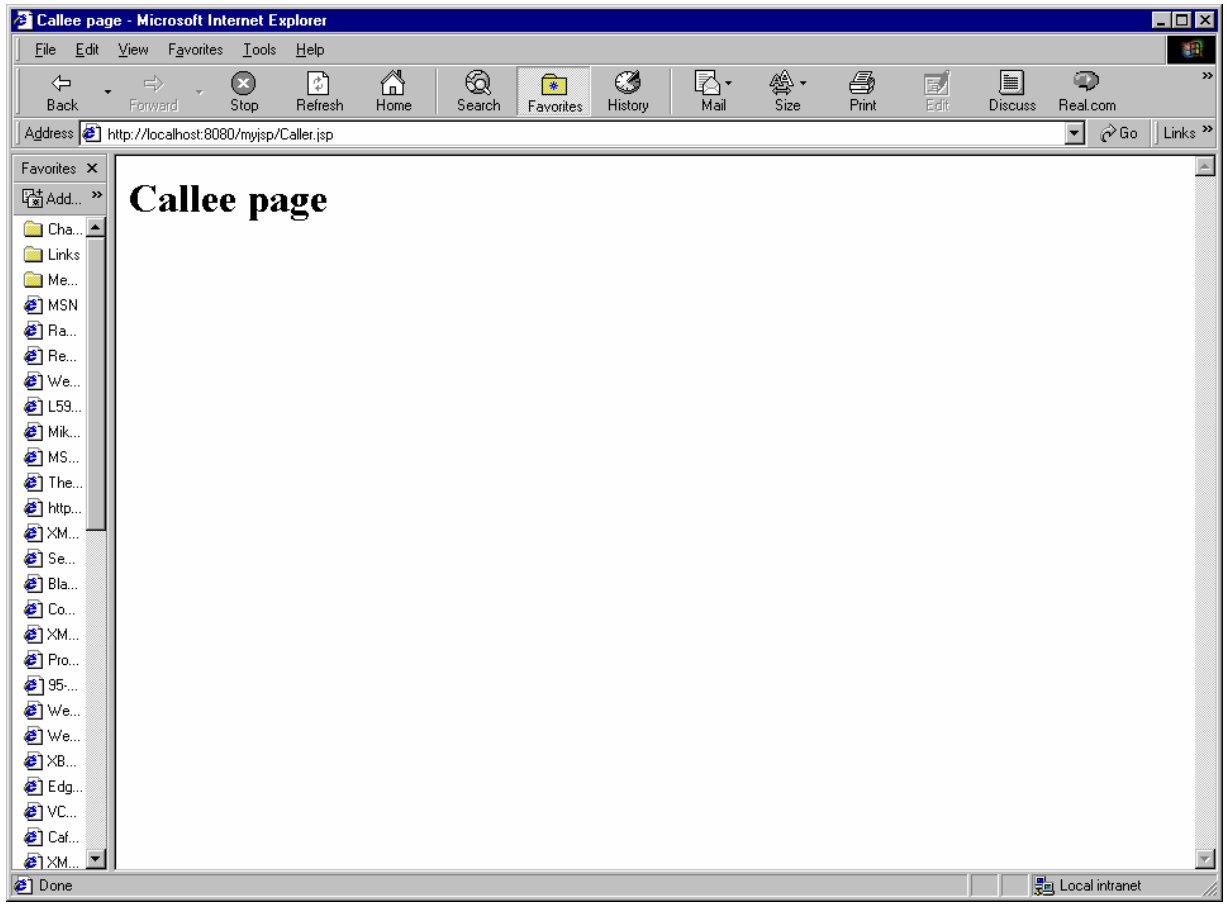
```
<%-- Caller page      Caller.jsp --%>

<html>
  <head>
    <title>Caller page </title>
  </head>
  <body>
    <h1> Caller page </h1>
    <jsp:forward page = "Callee.jsp" />
  </body>
</html>
```

Any response data is cleared and control passes to the new page.

Callee.jsp

```
<%-- Callee page --%>

<html>
   <head>
      <title>Callee page </title>
   </head>
   <body>
      <h1> Callee page </h1>
   </body>
</html>
```

After Visiting Caller.jsp



Request Scope

- One page may call another and the bean is still available.
- Its considered one request.
- The second page will use an existing bean before creating a new one.
- When the current request is complete the bean is reclaimed by the JVM.

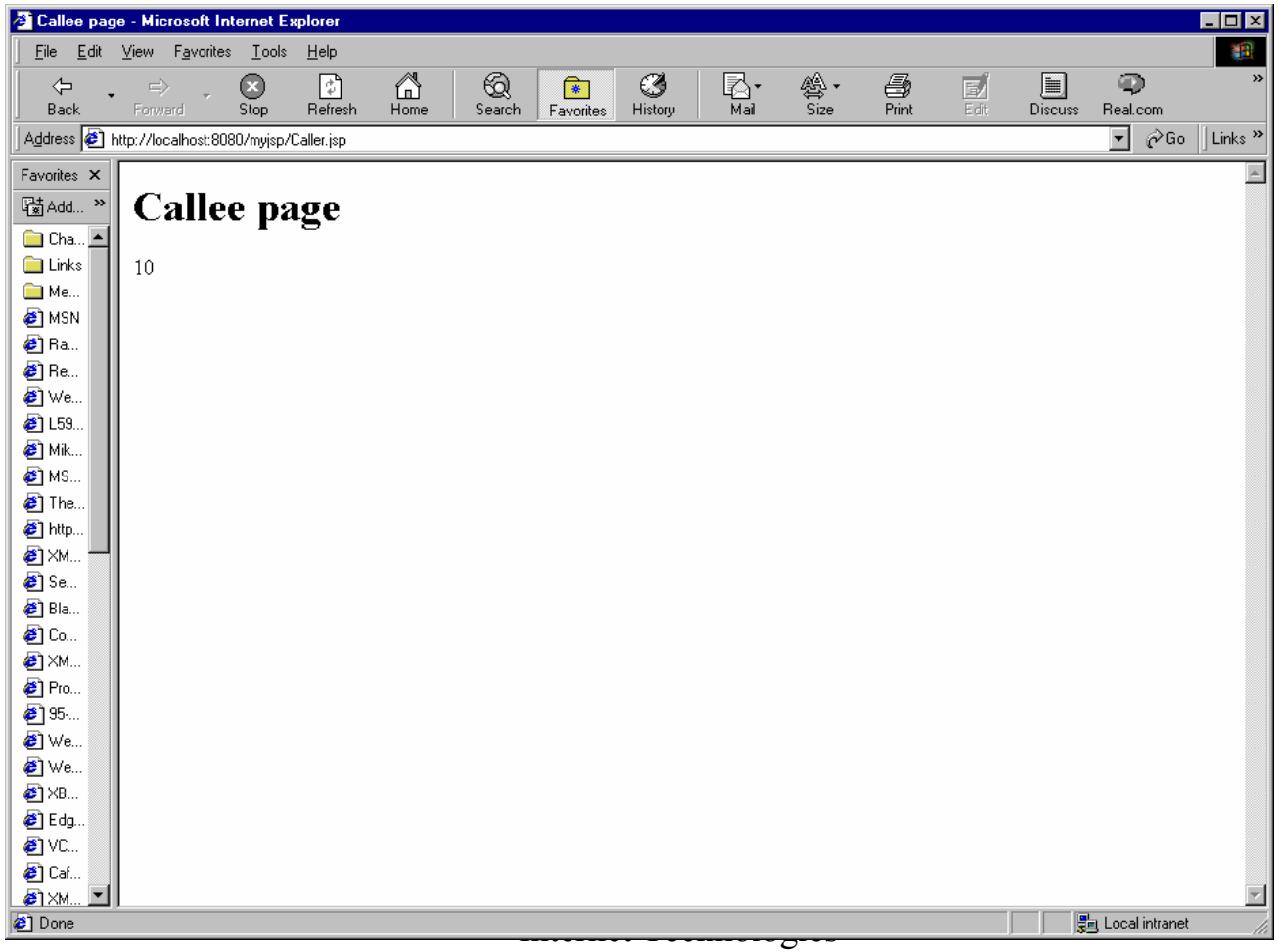
Request Scope Caller.jsp

```
<%-- Caller page --%>
<%@ page import="Counter" %>

<jsp:useBean id = "ctr" scope = "request" class = "Counter" />
<html>
   <head>
      <title>Caller page </title>
      <jsp:setProperty name = "ctr" property = "count" value = "10" />
   </head>
   <body>
      <h1> Caller page </h1>
      <jsp:forward page = "Callee.jsp" />
   </body>
</html>
```

Request Scope Callee.jsp

```
<%-- Callee page --%>
<%@ page import="Counter" %>
<jsp:useBean id = "ctr" scope = "request" class = "Counter" />
<html>
  <head>
    <title>Callee page </title>
  </head>
  <body>
    <h1> Callee page </h1>
    <jsp:getProperty name = "ctr" property ="count" />
  </body>
</html>
```

After Visiting Caller.jsp



Session Scope

Beans with session scope are accessible within pages processing requests that are in the same session as the one in which the bean was created.

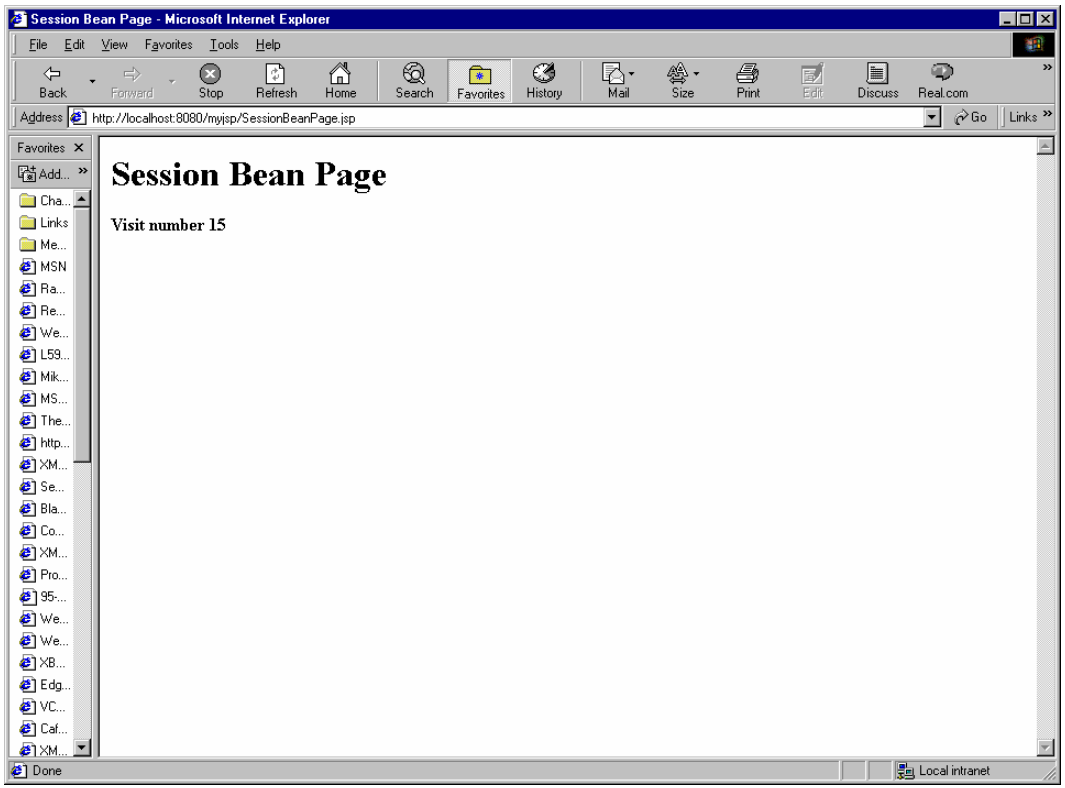
Session lifetime is typically configurable and is controlled by the servlet container. Currently, my session ends when the browser exits.

Multiple copies of the same browser each get their own session bean.

Session Scope Example

```
<%-- SessionBeanPage.jsp --%>
<%@ page import="Counter" %>
<jsp:useBean id = "ctr" scope = "session" class = "Counter" />
<html>
  <head>
    <title>Session Bean Page </title>
  </head>
  <body>
    <h1> Session Bean Page </h1>
    <B>Visit number
        <jsp:getProperty name = "ctr" property = "count"/>
    </B>
  </body>
</html>
```

Session Scope Example



The counter increments on each hit till browser exits. New browser back to 1.

Application Beans

A bean with a scope value of application has an even broader and further reaching availability than session beans.

Application beans exist throughout the life of the JSP container itself, meaning they are not reclaimed until the server is shut down.

Session beans are available on subsequent requests from the same browser. Application beans are shared by all users.

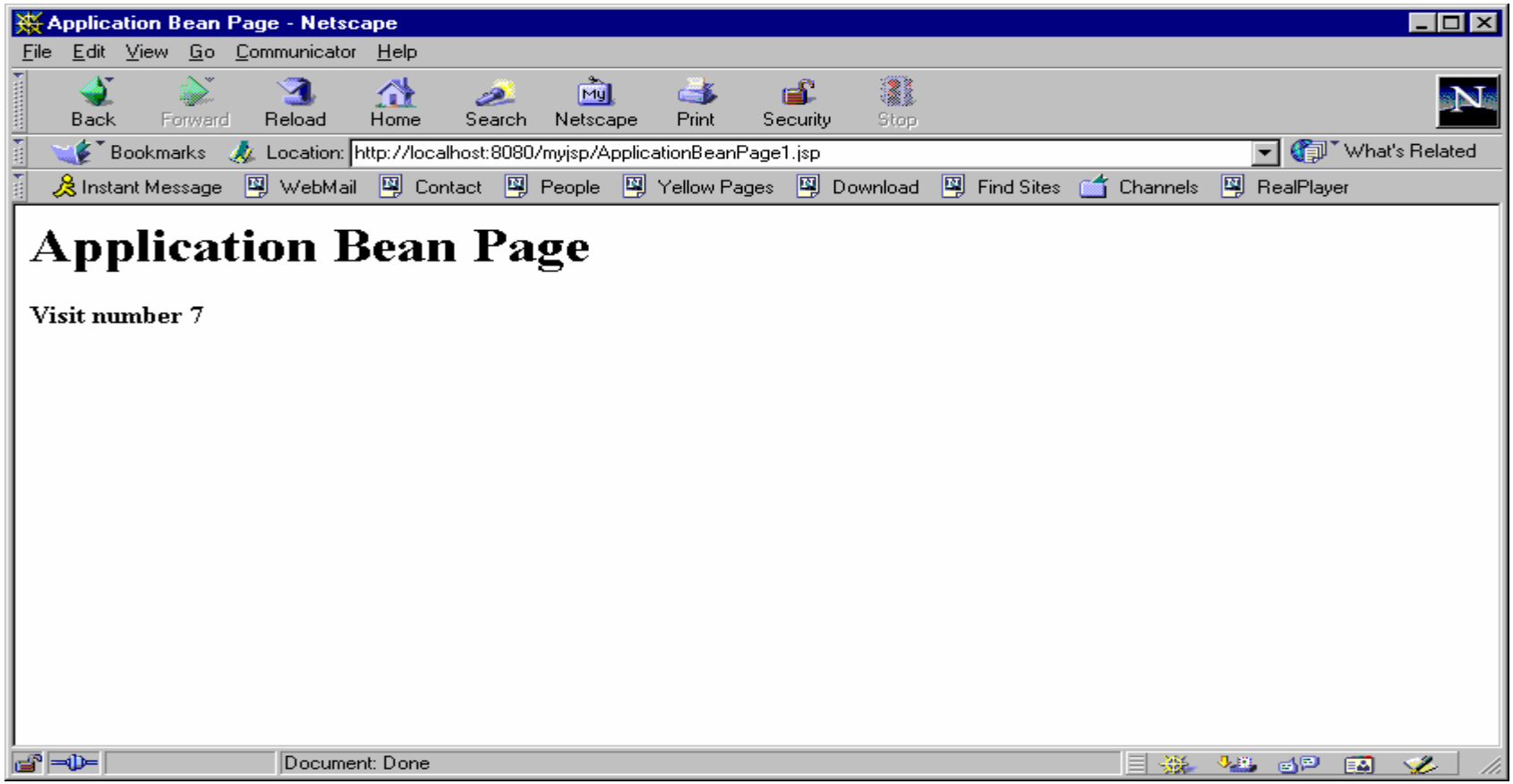
Application Bean Example 1

```
<%-- ApplicationBeanPage1.jsp --%>
<%@ page import="Counter" %>
<jsp:useBean id = "ctr" scope = "application" class = "Counter" />
<html>
  <head>
    <title>Application Bean Page </title>
  </head>
  <body>
    <h1> Application Bean Page </h1>
    <B>Visit number <jsp:getProperty name = "ctr“
                          property = "count"/> </B>
  </body>
</html>
```

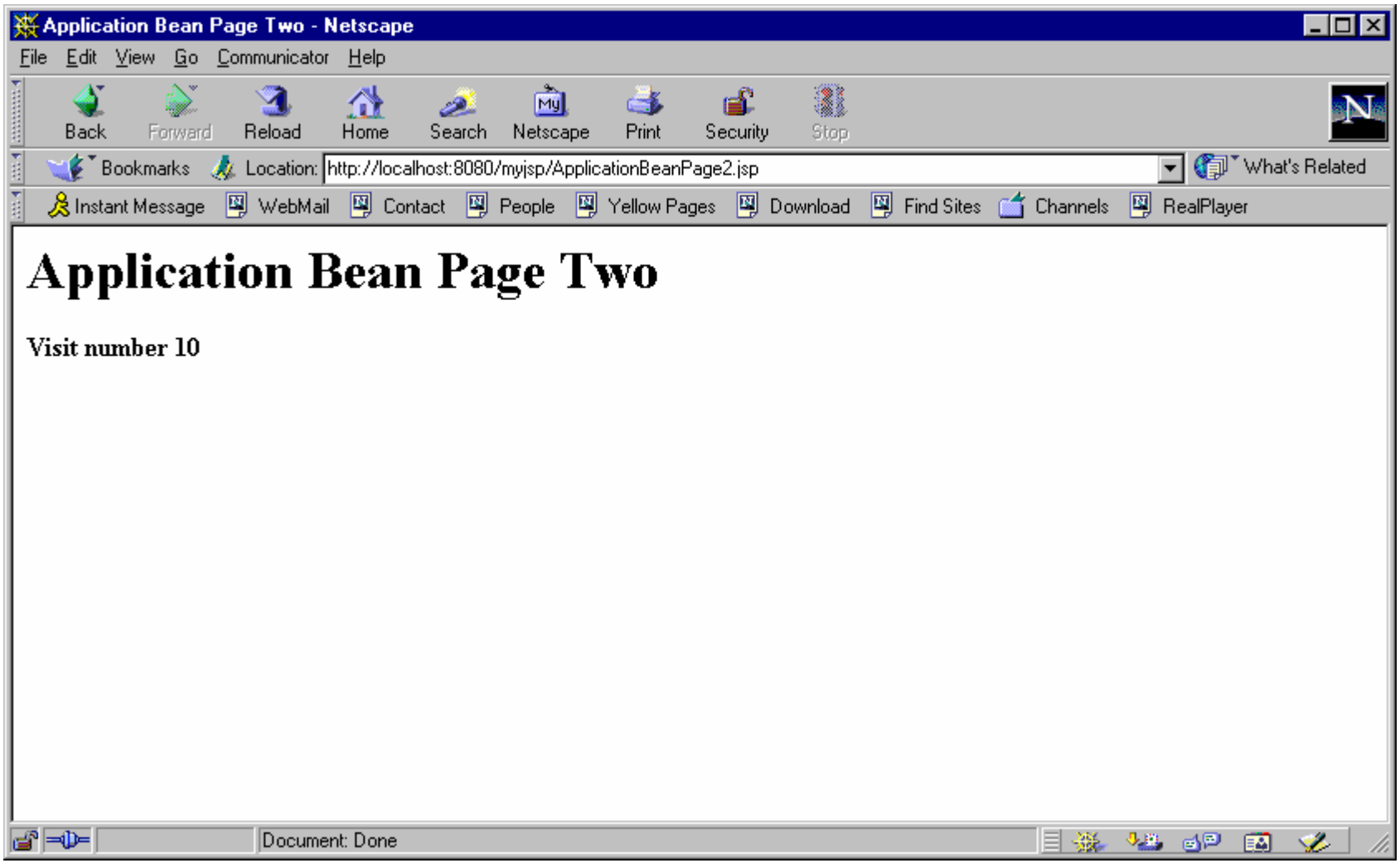
Application Bean Example 2

```
<%-- ApplicationBeanPage2.jsp --%>
<%@ page import="Counter" %>

<jsp:useBean id = "ctr" scope = "application" class = "Counter" />
<html>
  <head>
    <title>Application Bean Page Two </title>
  </head>
  <body>
    <h1> Application Bean Page Two </h1>
    <B>Visit number <jsp:getProperty name = "ctr“
                          property = "count"/> </B>
  </body>
</html>
```

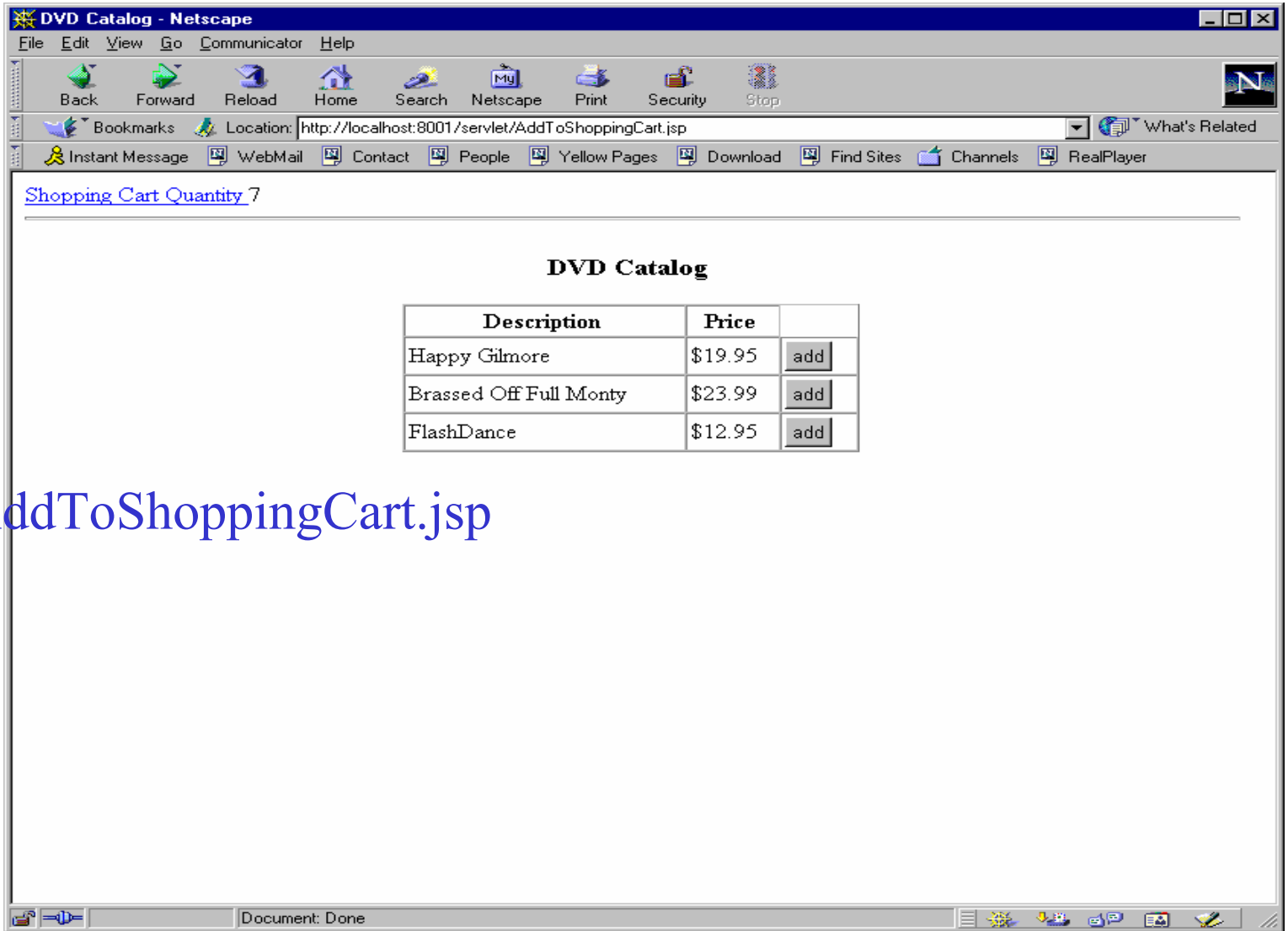
After several visits with IE5 we visit with Netscape.



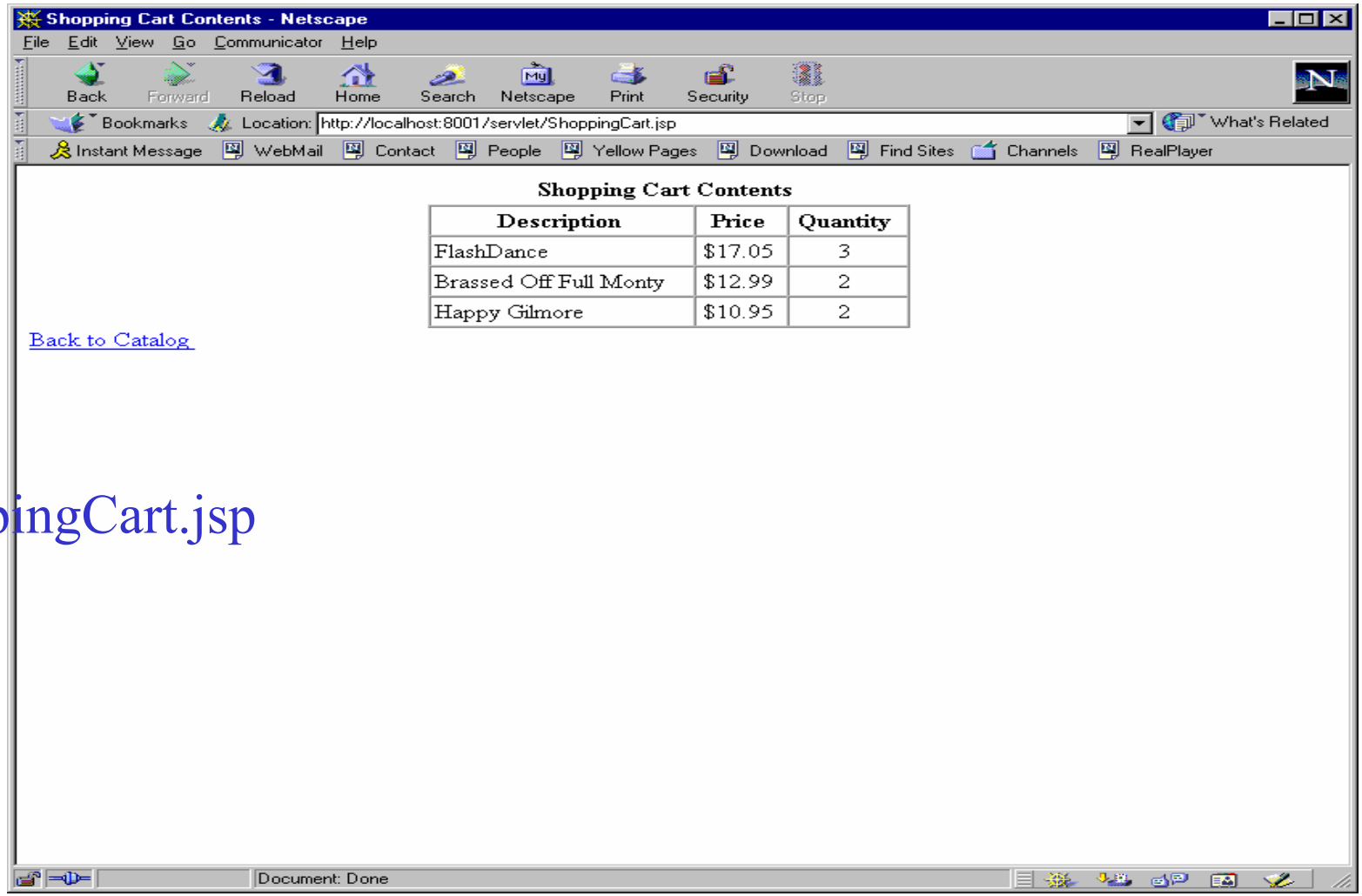
After visiting from a different machines with a different browsers, we still keep count.



A Shopping Cart



AddToShoppingCart.jsp



ShoppingCart.jsp

The Bean – ShoppingCart.java

```
// Adapted from From James Goodwill Pure JSP
// ShopingCart.java

import java.util.*;

public class ShoppingCart implements Serializable {

    protected Hashtable items = new Hashtable();

    public ShoppingCart() {}
```

```
public void addItem(String itemId, String description, float price, int quantity) {

        // pack the item as an array of Strings
        String item[] = { itemId, description, Float.toString(price),
                          Integer.toString(quantity)};

        // if item not yet in table then add it
        if(! items.containsKey(itemId)) {

            items.put(itemId, item);
        }
        else { // the item is in the table already
              String tempItem[] = (String[])items.get(itemId);
              int tempQuant = Integer.parseInt(tempItem[3]);
              quantity += tempQuant;
              tempItem[3] = Integer.toString(quantity);

        }
}
```

Get a reference to a hashtable entry.

Change it.

```
public void removeItem(String itemId) {

        if(items.containsKey(itemId)) {

            items.remove(itemId);
        }
    }
    public void updateQuantity(String itemId, int quantity) {

        if(items.containsKey(itemId)) {

            String[] tempItem = (String[]) items.get(itemId);
            tempItem[3] = Integer.toString(quantity);

        }
    }
 public Enumeration getEnumeration() {

        return items.elements();
    }
```

```
public float getCost() {

    Enumeration enum = items.elements();
    String[] tempItem;
    float totalCost = 0.00f;

    while(enum.hasMoreElements()) {

      tempItem = (String[]) enum.nextElement();
      totalCost += (Integer.parseInt(tempItem[3]) *
                    Float.parseFloat(tempItem[2]));
    }
    return totalCost;
}
```

```
public int getNumOfItems() {

        Enumeration enum = items.elements();
        String tempItem[];
        int numOfItems = 0;
        while(enum.hasMoreElements()) {

          tempItem = (String[]) enum.nextElement();
          numOfItems += Integer.parseInt(tempItem[3]);
        }
        return numOfItems;
    }
```

```
public static void main(String a[]) {

        ShoppingCart cart = new ShoppingCart();

        cart.addItem("A123", "Bike", (float)432.46, 10);
        cart.addItem("A124", "Bike", (float)732.46, 5);
        System.out.println(cart.getNumOfItems());
        System.out.println(cart.getCost());
        cart.updateQuantity("A123", 2);
        System.out.println(cart.getNumOfItems());
        cart.addItem("A123", "Bike", (float)432.46, 4);
        System.out.println(cart.getNumOfItems());
    }

}
```

```
D:\Apache Tomcat 4.0\webapps\myjsp\WEB-INF\classes>java ShoppingCart
15
7986.9004
7
11
```

AddToShoppingCart.jsp

```
<!-- Adapted from From James Goodwill Pure JSP -->
<%@ page errorPage = "errorpage.jsp" %>
<%@ page import="ShoppingCart" %>
<%@ page language = "java" %>

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

<html>
  <head>
    <title>DVD Catalog </title>
  </head>
```

```
<%

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

    if(id != null) {

        String desc = request.getParameter("desc");
        Float price = new Float(request.getParameter("price"));

        cart.addItem(id, desc, price.floatValue(), 1);
    }

%>
```

We have access to the request object.

It will be null on the first visit.

Add an item to the bean.

JSP uses java as the scripting language. Think of this code as being a part of the resulting servlet.

```
<a href = "ShoppingCart.jsp"> Shopping Cart Quantity </a>
<%= cart.getNumOfItems() %>
<hr>
<center>
      <h3> DVD Catalog </h3>
</center>
 <table border = "1" width = "300“
                  cellspacing = "0" cellpadding = "2" align = "center" >
    <tr>
        <th> Description </th>
        <th> Price </th>
     </tr>
```

This is a JSP expression. It's evaluated and sent to the response object.

It will change every time The servlet is hit within this session.

```
<tr>
    <form action = "AddToShoppingCart.jsp" method = "post" >
        <td>Happy Gilmore</td>
        <td>$19.95</td>
        <td>
          <input type = "submit" name = "submit" value = "add">
        </td>

        <input type = "hidden" name = "id" value = "1" >
        <input type = "hidden" name = "desc" value = "Happy Gilm
        <input type = "hidden" name = "price" value = "10.95" >
    </form>
</tr>
```

If this form is clicked we execute this same servlet but pass back the hidden form fields.

```
<tr>

    <form action = "AddToShoppingCart.jsp" method = "post" >

        <td>Brassed Off Full Monty</td>
        <td>$23.99</td>
        <td>
          <input type = "submit" name = "submit" value = "add">
        </td>

        <input type = "hidden" name = "id" value = "2" >
        <input type = "hidden" name = "desc" value =
                        "Brassed Off Full Monty" >
        <input type = "hidden" name = "price" value = "12.99" >
    </form>
</tr>
```

Hidden data going back to the server.

```
<form action = "AddToShoppingCart.jsp" method = "post" >

            <td>FlashDance</td>
            <td>$12.95</td>
            <td>
              <input type = "submit" name = "submit" value = "add">
            </td>

            <input type = "hidden" name = "id" value = "3" >
            <input type = "hidden" name = "desc" value =
                                                    "FlashDance" >
            <input type = "hidden" name = "price" value = "17.05" >
        </form>
      </tr>
    </table>
   </body>
<html>
```

ShoppingCart.jsp

```
<!-- Adapted from From James Goodwill Pure JSP -->

<%@ page errorPage = "errorpage.jsp" %>
<%@ page import = "java.util.*" %>
<%@ page import="ShoppingCart" %>
<%@ page language = "java" contentType="text/html" %>

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

We have access to the same bean.

```
<html>
  <head>
    <title> Shopping Cart Contents </title>
  </head>
  <body>
    <center>
    <table width = "300" border = "1" cellspacing = "0"
            cellpadding = "2" border = "0" >
    <caption>
       <b> Shopping Cart Contents </b>
    </caption>
    <tr>
      <th> Description </th>
      <th> Price </th>
      <th> Quantity </th>
    </tr>
```

```
<%

    Enumeration enum = cart.getEnumeration();
    String tempItem[];

    while(enum.hasMoreElements()) {

      tempItem = (String[]) enum.nextElement();

%>
```

This loop executes within the servlet.

Notice that the loop has not ended but we are done with java for the moment…

```
<tr>
    <td>
        <%= tempItem[1] %>
    </td>
     <td align = "center">
         <%= "$" + tempItem[2] %>
     </td>
     <td align = "center">
          <%= tempItem[3] %>
     </td>
</tr>
      <% } %>
   </table>
  </center>
  <a href = "AddToShoppingCart.jsp">Back to Catalog </a>
 </body>
</html>
```

We start generating html to the response object (in the loop).

End of loop. Java code is Enclosed within the <% .. %>

A Simple JSP/JDBC Example

stocks.mdb database schema

There are three tables. Both customer and stocks have a one-to-many relationship with portfolios. The database stocks.mdb was registered with the ODBC driver as "CoolStocks"

customer	stocks	portfolio
id	symbol	id
lname	company	symbol
fname	price	num_shares

Register w/ODBC

Create an ODBC data source.
Click on the Start button.
Choose Settings, Control Panel
Double-click on ODBC Data Sources
Choose the System DSN tab
Click Add
Click on the desired driver (MSAccess)
Click on the Finish button
Enter a Data Source Name (I called my database CoolStocks and that name appears in the java code below)
Click on the Select button
Locate the directory and file containing your database. This will be the “stock.mdb” file created by Microsoft Access.
Click OK

A Simple JSP/JDBC Example

<TITLE>JSP JDBC Example 1</TITLE>
</HEAD>

<BODY>
<!– Adapted from James Goodwill's Pure JSP →
<!-- Set the scripting language to java and -->
<!-- import the java.sql package -->
<%@ page language="java" import="java.sql.*" %>
<%@ page import= "java.io.*" %>

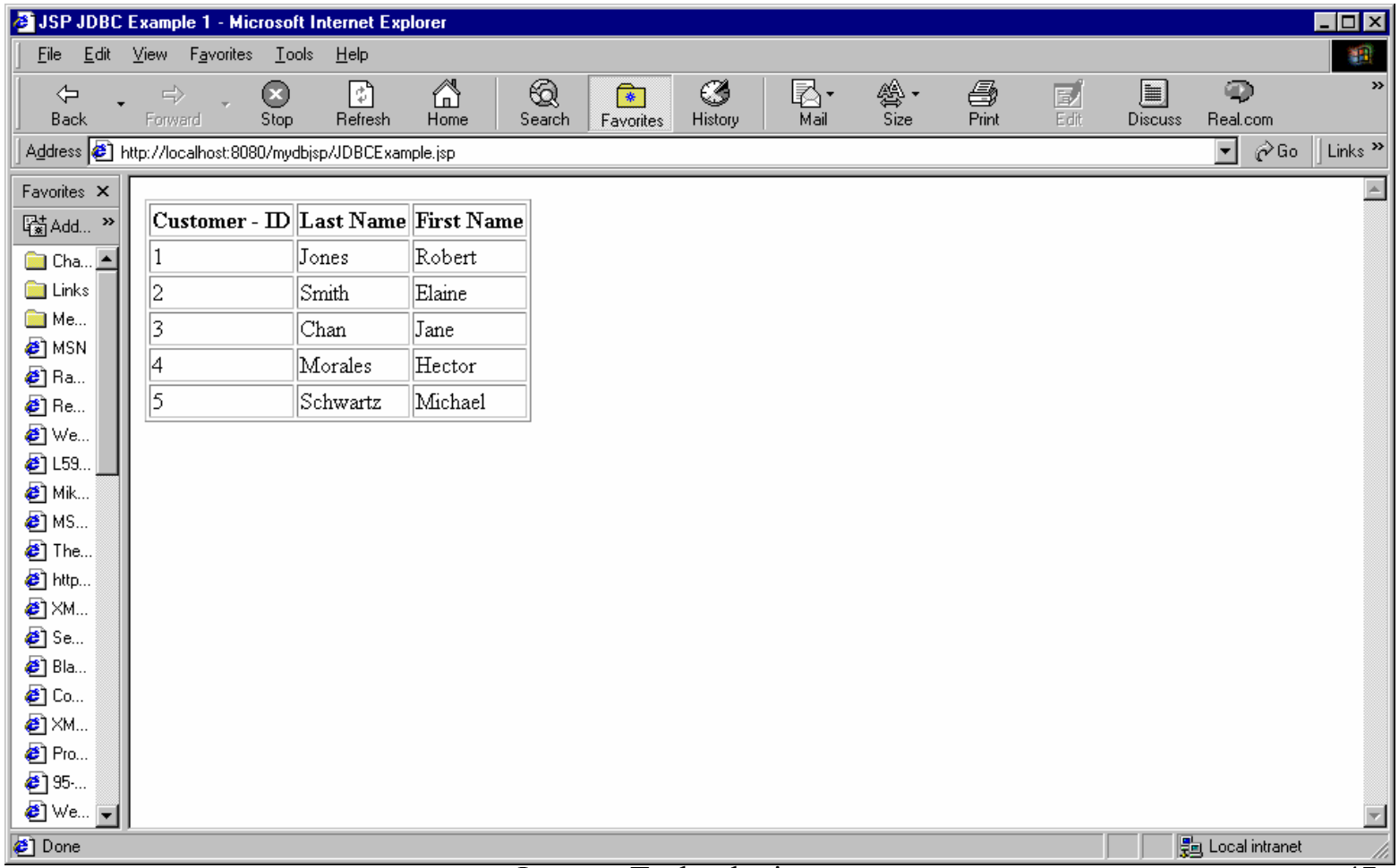
```
<%
   Connection con = null;
   try {
     // Load the Driver class file
     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

     // Make a connection to the ODBC datasource Movie Catalog
     con = DriverManager.getConnection("jdbc:odbc:CoolStocks");

     // Create the statement
     Statement statement = con.createStatement();

     // Use the created statement to SELECT the DATA
     // FROM the customer Table.
     ResultSet rs = statement.executeQuery("SELECT * " +
       "FROM customer");
     // Iterate over the ResultSet
%>
```

```
<!-- Add an HTML table to format the results -->
<TABLE BORDER="1">
<TR>
<TH> Customer - ID</TH><TH>Last Name</TH>
<TH>First Name</TH>
<%
while ( rs.next() ) {

  // get the id, convert to String
  out.println("<TR>\n<TD>" + rs.getString("id") + "</TD>");

  // get the last name
  out.println("<TD>" + rs.getString("lname") + "</TD>");

  // get the first name
  out.println("<TD>" + rs.getString("fname") + "</TD>\n</TR");
}
```

```
 // Close the ResultSet
  rs.close();
}
catch (IOException ioe) {
  out.println(ioe.getMessage());
}
catch (SQLException sqle) {
  out.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe) {
  out.println(cnfe.getMessage());
}
catch (Exception e) {
  out.println(e.getMessage());
}
```

```
finally {
      try {
        if ( con != null ) {

          // Close the connection no matter what
          con.close();
        }
      }
      catch (SQLException sqle) {

        out.println(sqle.getMessage());
      }
    }

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

It Works!



Customer - ID	Last Name	First Name
1	Jones	Robert
2	Smith	Elaine
3	Chan	Jane
4	Morales	Hector
5	Schwartz	Michael

An Example Using Connection Pooling

The example above opens a connection every time there is a visit.

Goodwill presents another approach in chapter 14.

PooledConnection.java

```
// Adapted from Goodwill's Pure JSP
import java.sql.*;

public class PooledConnection {

  // Real JDBC Connection
  private Connection connection = null;
  // boolean flag used to determine if connection is in use
  private boolean inuse = false;
```

```
// Constructor that takes the passed in JDBC Connection
// and stores it in the connection attribute.
public PooledConnection(Connection value) {
  if ( value != null ) {
    connection = value;
  }
}

// Returns a reference to the JDBC Connection
public Connection getConnection() {
  return connection;
}
```

```
// Set the status of the PooledConnection.
  public void setInUse(boolean value) {
    inuse = value;
  }
 // Returns the current status of the PooledConnection.
  public boolean inUse() {
    return inuse;
  }
  // Close the real JDBC Connection
  public void close() {
    try {
      connection.close();
    }
    catch (SQLException sqle) {
      System.err.println(sqle.getMessage());
    }
  }
}
```

ConnectionPool.java

```
// Adapted from James Goodwill's Pure Java

import java.sql.*;
import java.util.*;

public class ConnectionPool {

  // JDBC Driver Name
  private String driver = null;
  // URL of database
  private String url = null;
  // Initial number of connections.
  private int size = 0;
```

```
// Username
  private String username = new String("");
  // Password
  private String password = new String("");
  // Vector of JDBC Connections
  private Vector pool = null;

  public ConnectionPool() {

  }

  // Set the value of the JDBC Driver
  public void setDriver(String value) {
    if ( value != null ) {
      driver = value;
    }
  }
```

```
// Get the value of the JDBC Driver
  public String getDriver() {
    return driver;
  }

  // Set the URL Pointing to the Datasource
  public void setURL(String value ) {
    if ( value != null ) {
      url = value;
    }
  }

  // Get the URL Pointing to the Datasource
  public String getURL() {
    return url;
  }
```

```
// Set the initial number of connections
public void setSize(int value) {
  if ( value > 1 ) {
    size = value;
  }
}

// Get the initial number of connections
public int getSize() {
  return size;
}
// Set the username
public void setUsername(String value) {
  if ( value != null ) {
    username = value;
  }
}
```

```
// Get the username
  public String getUserName() {
    return username;
  }

  // Set the password
  public void setPassword(String value) {
    if ( value != null ) {
      password = value;
    }
  }
  // Get the password
  public String getPassword() {
    return password;
  }
```

```
// Creates and returns a connection
private Connection createConnection() throws Exception {

  Connection con = null;

  // Create a Connection
    con = DriverManager.getConnection(url,
    username, password);

  return con;
}
```

```
// Initialize the pool
public synchronized void initializePool() throws Exception {

    // Check our initial values
    if ( driver == null ) {
    throw new Exception("No Driver Name Specified!");
    }
    if ( url == null ) {
    throw new Exception("No URL Specified!");
    }
    if ( size < 1 ) {
    throw new Exception("Pool size is less than 1!");
  }
```

```
// Create the Connections
try {
  // Load the Driver class file
  Class.forName(driver);
  // Create Connections based on the size member
  for ( int x = 0; x < size; x++ ) {
      Connection con = createConnection();
      if ( con != null ) {
        // Create a PooledConnection to encapsulate the
        // real JDBC Connection
        PooledConnection pcon = new PooledConnection(con);
        // Add the Connection to the pool.
        addConnection(pcon);
    }
  }
}
```

```
catch (Exception e) {
      System.err.println(e.getMessage());
      throw new Exception(e.getMessage());
    }
  }
  // Adds the PooledConnection to the pool
  private void addConnection(PooledConnection value) {
    // If the pool is null, create a new vector
    // with the initial size of "size"
    if ( pool == null ) {
      pool = new Vector(size);
    }
    // Add the PooledConnection Object to the vector
    pool.addElement(value);
  }
```

```
public synchronized void releaseConnection(Connection con) {

    // find the PooledConnection Object
    for ( int x = 0; x < pool.size(); x++ ) {

      PooledConnection pcon =
        (PooledConnection)pool.elementAt(x);
      // Check for correct Connection
      if ( pcon.getConnection() == con ) {

        System.err.println("Releasing Connection " + x);
        // Set its inuse attribute to false, which
        // releases it for use
        pcon.setInUse(false);
        break;
      }
    }
}
```

```
// Find an available connection
  public synchronized Connection getConnection()
    throws Exception {
    PooledConnection pcon = null;
    // find a connection not in use
    for ( int x = 0; x < pool.size(); x++ ) {
      pcon = (PooledConnection)pool.elementAt(x);

      // Check to see if the Connection is in use
      if ( pcon.inUse() == false ) {

        // Mark it as in use
        pcon.setInUse(true);
        // return the JDBC Connection stored in the
        // PooledConnection object
        return pcon.getConnection();
      }
    }
```

```
  // Could not find a free connection socreate and add a new one
  try {
      // Create a new JDBC Connection
      Connection con = createConnection();
      // Create a new PooledConnection, passing it the JDBC Connection
      pcon = new PooledConnection(con);
      // Mark the connection as in use
      pcon.setInUse(true);
      // Add the new PooledConnection object to the pool
      pool.addElement(pcon);
  }
  catch (Exception e) {
    System.err.println(e.getMessage());
    throw new Exception(e.getMessage());
  }
  // return the new Connection
  return pcon.getConnection();
}
```

```
// When shutting down the pool, you need to first empty it.
public synchronized void emptyPool() {

  // Iterate over the entire pool closing the
  // JDBC Connections.
  for ( int x = 0; x < pool.size(); x++ ) {

    System.err.println("Closing JDBC Connection " + x);

    PooledConnection pcon =
      (PooledConnection)pool.elementAt(x);

    // If the PooledConnection is not in use, close it
    if ( pcon.inUse() == false ) {
      pcon.close();
    }
```

```
else {

        // If it is still in use, sleep for 30 seconds and
        // force close.
        try {

          java.lang.Thread.sleep(30000);
          pcon.close();
        }
        catch (InterruptedException ie) {

          System.err.println(ie.getMessage());
        }
      }
    }
  }
}
```

JDBCPooledExample.jsp

```
<html>
  <body>
<%@ page errorPage="errorpage.jsp" %>
<%@ page import="java.util.*" %>
<%@ page import="java.sql.*" %>
<%@ page import= "java.io.*" %>
<%@ page import="ConnectionPool" %>

<!-- Instantiate the ConnectionPool bean with an id of "pool" -->
<jsp:useBean id="pool"
  scope="application"
  class="ConnectionPool" />
```

```
<%
  Connection con = null;
  try {
    // The pool is not initialized
    if ( pool.getDriver() == null ) {
      // initialize the pool
      pool.setDriver("sun.jdbc.odbc.JdbcOdbcDriver");
      pool.setURL("jdbc:odbc:CoolStocks");
      pool.setSize(5);
      pool.initializePool();
    }
    // Get a connection from the ConnectionPool
    con = pool.getConnection();
    // Create the statement
    Statement statement = con.createStatement();
```

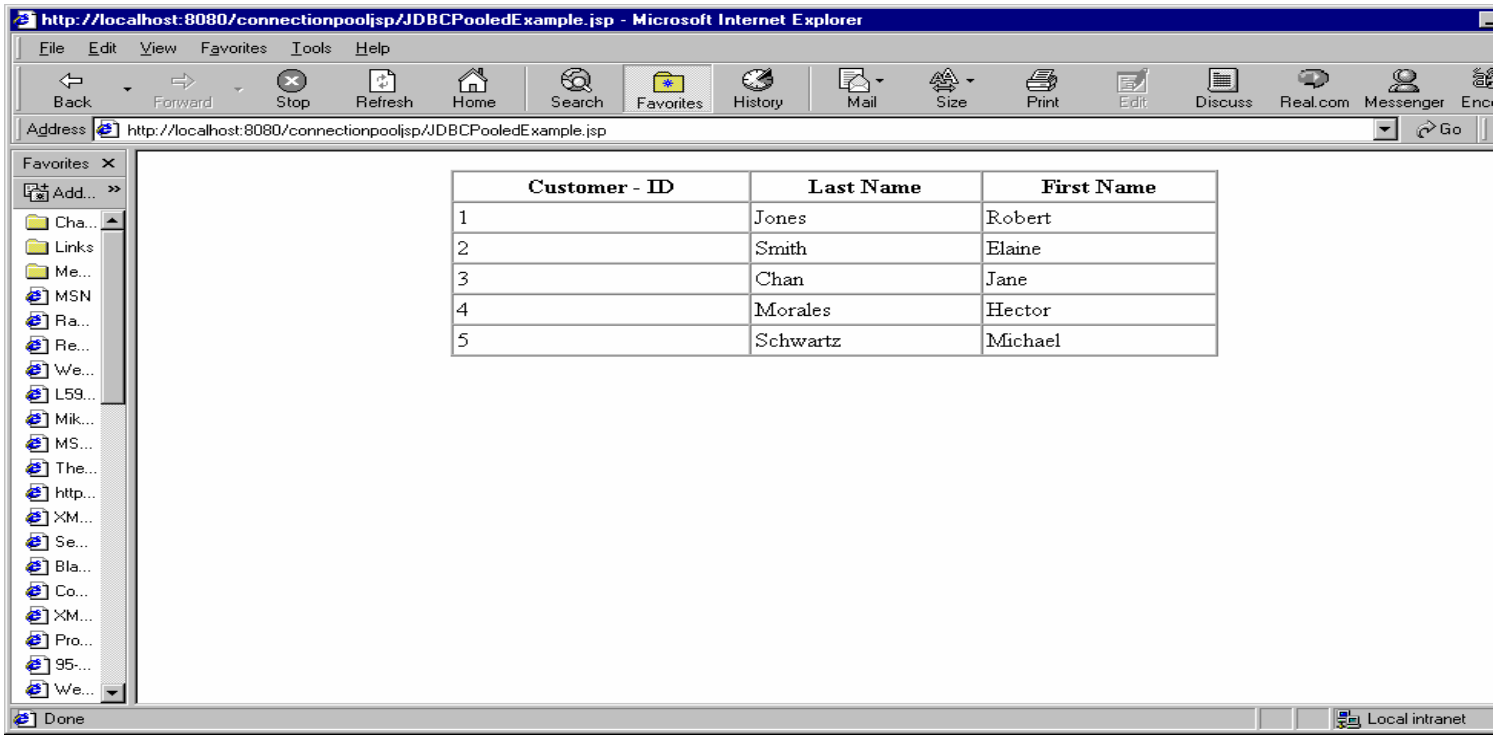
```
// Use the created statement to SELECT the DATA
// FROM the customer Table.
ResultSet rs = statement.executeQuery("SELECT * " +
  "FROM customer");

// Iterate over the ResultSet
%>
<!-- Add an HTML table to format the results -->
<center>
<table border="1" cellspacing="0" cellpadding="2"width="500">
  <tr>
   <TH> Customer - ID</TH><TH>Last Name</TH>
   <TH>First Name</TH>
```

```
<%
      while ( rs.next() ) {

       // get the id, convert to String
       out.println("<TR>\n<TD>" + rs.getString("id") + "</TD>");
       // get the last name
       out.println("<TD>" + rs.getString("lname") + "</TD>");
       // get the first name
       out.println("<TD>" + rs.getString("fname") + "</TD>\n</TR");
      }

   // Close the ResultSet
   rs.close();
   out.println("</table></center>");
  }
  catch (IOException ioe) {
   out.println(ioe.getMessage());
  }
```

```
catch (SQLException sqle) {
    out.println(sqle.getMessage());
  }
  catch (ClassNotFoundException cnfe) {
    out.println(cnfe.getMessage());
  }
  catch (Exception e) {
    out.println(e.getMessage());
  }
  finally {
    try {
      if ( con != null ) {
        // release the connection no matter what
        pool.releaseConnection(con);
      }
    }
```

```
catch (Exception e) {

      out.println(e.getMessage());
    }
  }
%>
  </body>
</html>
```

It works too!



Customer - ID	Last Name	First Name
1	Jones	Robert
2	Smith	Elaine
3	Chan	Jane
4	Morales	Hector
5	Schwartz	Michael

Summary

Connections take a long time to open.

Transactions are linked to connections and should not normally be shared.

If you have a very low traffic site then you might open and close the connection inside the servlet's doxxx() method.

Perhaps the best approach is to use connection pooling with different roles assigned to different connections. Each visitor has their own connection with an established role. The connections are only created during server startup.

Using EJB Entity beans would greatly simplify this project