Using the JSP Standard Tag Library (JSTL) with JSF

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/JSF-Tutorial/

For live training on JSF 1.x or 2.0, please see courses at http://courses.coreservlets.com/.

Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - Java 5, Java 6, intermediate/beginning servlets/JSP, advanced servlets/JSP, Struts, JSF, Ajax, GWT, custom courses.
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, Ruby/Rails

Contact hall@coreservlets.com for details

Agenda

- **Obtaining JSTL documentation and code**
- **The JSTL Expression Language**
- **Looping Tags**
  - Looping a certain number of times
  - Looping over data structures
  - Improving Ajax MVC data-handling examples
- **Conditional Evaluation Tags**
  - Single choice
  - Multiple choices
- **Database Access Tags**
- **Other Tags**

Overview and Installation

JSTL Overview

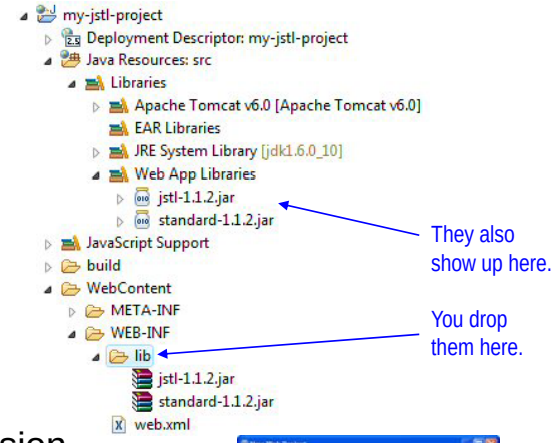
- **Full JSTL**
  - Contains many common and useful JSP custom tags
  - Particularly useful when you are using MVC, but the data contains a varying number of entries
  - Based on the Struts looping and logic tags
  - Not part of the JSP 1.2, 2.0, or 2.1 specs
    - It is a separate specification that requires a separate download
- **Expression language**
  - The JSP expression language came from JSTL, but is now part of JSP 2.0, JSP 2.1, and JSF
- **The JSTL Specification is available in PDF**
  - http://jcp.org/en/jsr/detail?id=52

Downloading JSTL

- **JSTL 1.1**
  - Home page
    - http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html
  - Download
    - http://jakarta.apache.org/site/downloads/downloads_taglibs-standard.cgi
  - Documentation (Javadoc)
    - http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/
  - Note
    - Many server packages (e.g., MyFaces) and IDEs (e.g., MyEclipse) *already* have JSTL 1.1 bundled
- **JSTL 1.2**
  - As of 10/2008, only available as part of Java EE 5
    - Not as a separate download for use in other servers

Enabling JSTL with Eclipse or MyEclipse

- **Eclipse**
  - Download from Apache site
  - Drag jstl.jar and standard.jar onto WEB-INF/lib
    - They also get listed in " Web App Libraries"
    - I usually rename JAR files to show exact version
- **MyEclipse**
  - Follow Eclipse directions above, or
  - File → New →Web Project, select JSTL 1.1 option



Enabling JSTL Manually

- **Put in Web App**
  - Drop JAR files in the WEB-INF/lib folder of your Web app
- **Put in CLASSPATH**
  - Add the JAR files to the CLASSPATH used by your development environment
  - Do *not* add to the CLASSPATH used by the server

The JSTL Expression Language

- **Accessed via `${expression}`**
- **Similar to JavaScript and XPath**
- **Provides shorthand notation to access:**
  - Attributes of standard servlet objects
  - Bean properties
  - Map, List, and Array elements
- **Is standard part of JSP 2.0 and 2.1**
  - In JSTL, EL can be used only in attributes of JSTL tags
  - In JSP 2.0, the EL can be used anywhere
    - *web.xml* element and page directive attribute let you disable the EL for backward compatibility
- **EL covered in separate lecture**

Iteration Tags

JSF Constructs for Handling Variable-Length Data

Simplest for Page Author

Most Control for Page Author

- **Bean**
  - <h:outputText value="#{cust.withdrawals}" escape="true"/>
- **Custom tag (non-looping)**
  - <mytags:withDrawalTable customer="${cust}" border="..." headerStyles="..." .../>
- **h:dataTable**
- **t:dataList**
- **JSTL loop**
- **JSP scripting loop**

Looping Tags: Summary

- **Looping with explicit numeric values**

```
<c:forEach var="name" begin="x" end="y" step="z">
  Blah, blah ${name}
</c:forEach>
```

- **Looping over data structures**
  - Can loop down arrays, strings, collections, maps

```
<c:forEach var="name"
           items="array-or-collection">
  Blah, blah ${name}
</c:forEach>
```

- **Looping down delimited strings**
  - `forTokens`

Looping Tags: Motivation

- **JSP without JSTL**

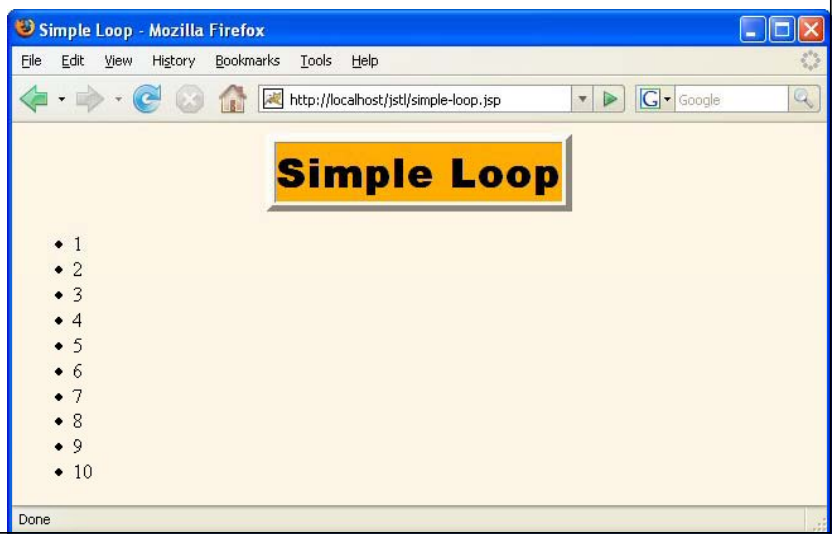
```
<UL>
<%
for(int i=0; i<messages.length; i++) {
  String message = messages[i];
%>
<LI><%= message %>
<% } %>
</UL>
```

- **JSP with JSTL**

```
<UL>
<c:forEach var="message" items="${messages}">
  <LI>${message}
</c:forEach>
</UL>
```

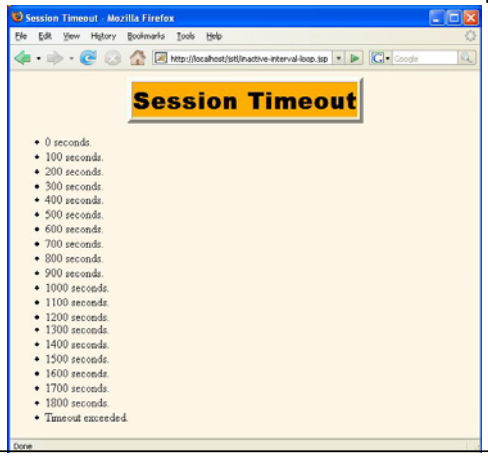

Looping with Simple Numeric Values

```
<%@ taglib prefix="c"
           uri="http://java.sun.com/jsp/jstl/core"%>
<UL>
<c:forEach var="i" begin="1" end="10">
  <LI>${i}
</c:forEach>
</UL>
```



Looping with a Designated Step Size

```
<%@ taglib prefix="c"
           uri="http://java.sun.com/jsp/jstl/core"%>
<UL>
<c:forEach
   var="seconds"
   begin="0"
   end="${pageContext.session.maxInactiveInterval}"
   step="100">
  <LI>${seconds} seconds.
</c:forEach>
  <LI>Timeout exceeded.
</UL>
```



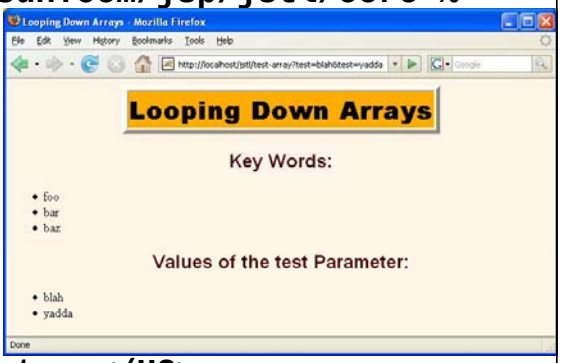
Looping Down Arrays (Servlet)

```
public class ArrayServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    String[] words = { "foo", "bar", "baz"};
    request.setAttribute("words", words);
    String address = "/WEB-INF/results/array-loop.jsp";
    RequestDispatcher dispatcher =
      request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
  }
}
```

- **Note**
  - url-pattern in web.xml is /test-array

Looping Down Arrays

```
<%@ taglib prefix="c"
           uri="http://java.sun.com/jsp/jstl/core"%>
<H2>Key Words:</H2>
<UL>
<c:forEach var="word"
           items="${words}">
  <LI>${word}
</c:forEach>
</UL>

<H2>Values of the test Parameter:</H2>
<UL>
<c:forEach var="val"
           items="${paramValues.test}">
  <LI>${val}
</c:forEach>
</UL>
```



Note for Older Servers

- **Examples in this lecture assume JSP 2.0+**
  - JSTL also runs in older servers, but you have to replace direct output of ${foo} with <c:out value="$foo"/>
- **Previous example (JSP 2.0+)**

```
<UL>
<c:forEach var="word" items="${words}">
  <LI>${word}
</c:forEach>
</UL>
```

- **Previous example (JSP 1.2)**

```
<UL>
<c:forEach var="word" items="${words}">
  <LI><c:out value="${word}"/>
</c:forEach>
</UL>
```

${foo} vs. <c:out value="${foo}"/>

- **c:out runs on older servers**
  - Oracle 9i, BEA WebLogic 8.x, IBM WebSphere 5.x
- **c:out escapes HTML (XML) characters**
  - <c:out value="<h1>"/> outputs <h1>
    - Very important if value being output comes from end user.
    - Disable with <c:out value="…" escapeXml="false"/>
- **c:out lets you supply a default value**
  - <c:out value="${foo}" default="explicit value"/> or <c:out value="${foo}" default="${calculatedValue}"/>
  - The default is output when ${foo} evaluates to null

Looping Down Collections: Notes

- **Can loop down collections other than arrays**
  - "items" can refer to an array, collection (List, Map, Set, etc.), comma-separated String, Iterator, or Enumeration
- **Can access sub-elements of local variable**
  - <c:forEach var="bean" items="${collectionOfBeans}">
    ${bean.property}
    </c:forEach>
  - <c:forEach var="collection"
    items="${collectionOfCollections}">
    ${collection[part]}
    </c:forEach>
  - For details on accessing collections and sub-elements, see separate lecture on the JSP expression language

Looping Down Arrays: Accessing SubElements (Servlet)

```
public class ArrayServlet2 extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    Name[] names =
      { new Name("Bill", "Gates"),
        new Name("Larry", "Ellison"),
        new Name("Sam", "Palmisano"),
        new Name("Scott", "McNealy"),
        new Name("Eric", "Schmidt"),
        new Name("Jeff", "Bezos") };
    request.setAttribute("names", names);
    String[][] sales =
      { {"2005", "12,459", "15,622"},
        {"2006", "18,123", "17,789"},
        {"2007", "21,444", "23,555"} };
    request.setAttribute("sales", sales);
    String address = "/WEB-INF/results/array-loop2.jsp";
    RequestDispatcher dispatcher =
      request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
  }
}
```

Note: url-pattern is /test-array2

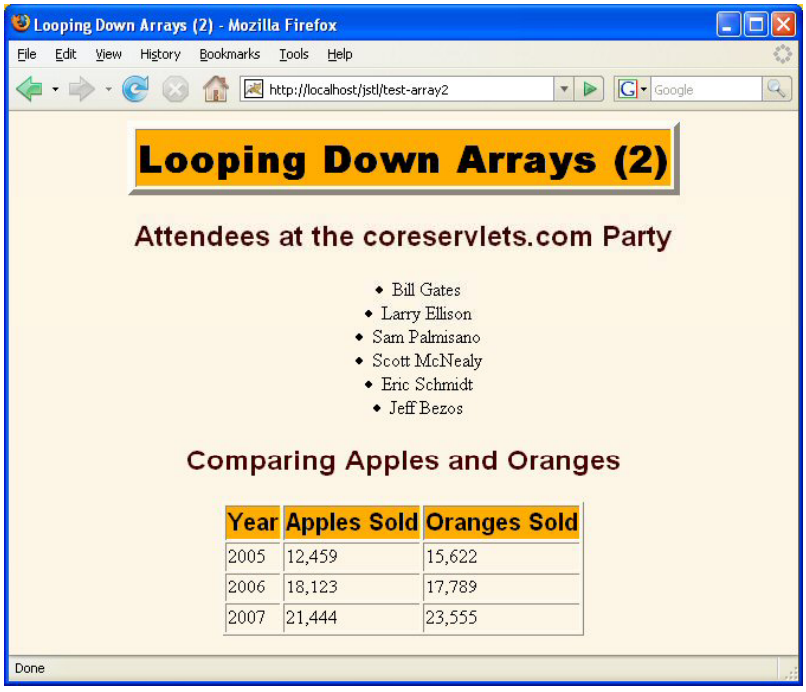
Looping Down Arrays: Accessing SubElements (Bean)

```
public class Name {
  private String firstName;
  private String lastName;

  public Name(String firstName, String lastName) {
    setFirstName(firstName);
    setLastName(lastName);
  }

  public String getFirstName() { return(firstName); }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() { return(lastName); }

  public void setLastName(String lastName) {
    this.lastName = lastName;
  }
}
```

Looping Down Arrays: Accessing SubElements (JSP)

```
<H2>Attendees at the coreservlets.com Party</H2>
<UL>
<c:forEach var="name" items="${names}">
  <LI>${name.firstName} ${name.lastName}
</c:forEach>
</UL>
<H2>Comparing Apples and Oranges</H2>
<TABLE BORDER="1">
  <TR><TH>Year</TH>
      <TH>Apples Sold</TH>
      <TH>Oranges Sold</TH></TR>
<c:forEach var="row" items="${sales}">
  <TR><c:forEach var="col" items="${row}">
        <TD>${col}</TD>
      </c:forEach>
</c:forEach>
</TABLE>
```


Looping Down Arrays: Accessing SubElements (Result)

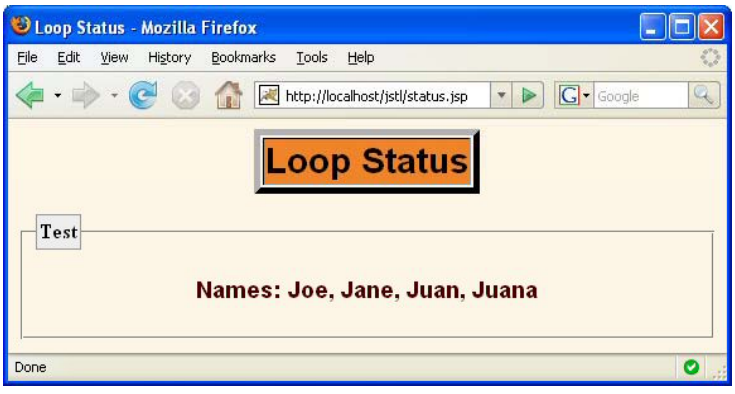


Loop Status

- **You can assign varStatus**
  - <c:forEach var="name" items="${names}" varStatus="status">
- **You can check status.subProperty**
  - Usually using c:if (covered later in this lecture)
- **Status subproperties**
  - index (int: the current index)
  - first (boolean: is this the first entry?)
  - last (boolean: is this the first entry?)
  - begin (Integer: value of 'begin' attribute)
  - end (Integer: value of end' attribute)
  - step (Integer: value of 'step' attribute)

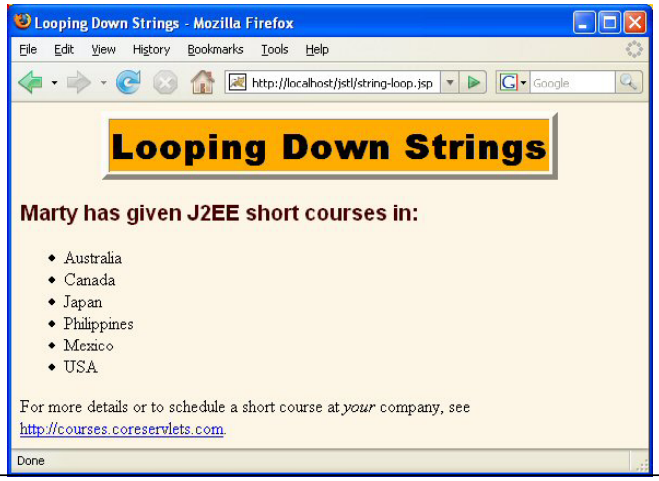
Loop Status: Example

```
<%@ taglib prefix="c"
          uri="http://java.sun.com/jsp/jstl/core" %>
<% String[] names = {"Joe", "Jane", "Juan", "Juana"};
   request.setAttribute("names", names); %>
<h2>Names:
<c:forEach var="name" items="${names}" varStatus="status">
  ${name}<c:if test="${!status.last}">,</c:if>
</c:forEach>
</h2>
```



Looping Down Comma-Delimited Strings

```
<%@ taglib prefix="c"
           uri="http://java.sun.com/jsp/jstl/core"%>
<UL>
<c:forEach
  var="country"
  items="Australia,Canada,Japan,Philippines,Mexico,USA">
  <LI>${country}
</c:forEach>
</UL>
```



Looping Down Arbitrarily-Delimited Strings

```
<%@ taglib prefix="c"
           uri="http://java.sun.com/jsp/jstl/core" %>
<UL>
<c:forTokens var="color"
    items="(red (orange) yellow)(green)((blue) violet)"
    delims="()">
  <LI>${color}
</c:forTokens>
</UL>
```

- **Point:**
  forTokens built on forEach: you can build your own custom tags based on JSTL tags

Looping Tags in Action: Ajax Data Handling

Original Version

- **In "Ajax Data Handling" lecture, showed example that displayed lists of cities**
  - JavaScript
    - Sent out request to server
    - Took data back in XML, JSON, or string format
    - Extracted information and produced HTML table
    - Inserted table into page
  - HTML
    - Loaded JavaScript files
    - Had button that trigged the action
  - Servlet
    - Produced List of City objects
  - JSP
    - Put city info into XML, JSON, or string format
      - Repeated entries for each city in List
  - Notes and source code for original example
    - http://courses.coreservlets.com/Course-Materials/ajax.html

HTML Code

```
...
<fieldset>
  <legend>Getting XML Data from Server...</legend>
  <form action="#">
   <label for="city-type-1">City Type:</label>
   <select id="city-type-1">
     <option value="top-5-cities">
       Largest Five US Cities</option>
     <option value="second-5-cities">
       Second Five US Cities</option>
     ...
    </select>
   <br/>
   <input type="button" value="Show Cities"
          onclick='xmlCityTable("city-type-1",
                                "xml-city-table")'/>
  </form>
  <p/>
  <div id="xml-city-table"></div>
</fieldset>...
```


Servlet Code

```
public class ShowCities extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setHeader("Cache-Control", "no-cache");
    response.setHeader("Pragma", "no-cache");
    String cityType = request.getParameter("cityType");
    List<City> cities = findCities(cityType);
    request.setAttribute("cities", cities);
    String format = request.getParameter("format");
    String outputPage;
    if ("xml".equals(format)) {
      response.setContentType("text/xml");
      outputPage = "/WEB-INF/results/cities-xml.jsp";
    } ...
    RequestDispatcher dispatcher =
      request.getRequestDispatcher(outputPage);
    dispatcher.include(request, response);
  }
```

Original JSP Code (before-jstl/cities-xml.jsp)

```
<?xml version="1.0" encoding="UTF-8"?>
<cities>
  <headings>
    <heading>City</heading>
    <heading>Time</heading>
    <heading>Population</heading>
  </headings>
  <city>
    <name>${cities[0].name}</name>
    <time>${cities[0].shortTime}</time>
    <population>${cities[0].population}</population>
  </city>
  ...    <!- Repeated for cities 1, 2, and 3 -->
  <city>
    <name>${cities[4].name}</name>
    <time>${cities[4].shortTime}</time>
    <population>${cities[4].population}</population>
  </city>
</cities>
```


Problems with JSP Code

- **Repetitive**
  - Almost-identical city entries repeated five times
  - If the city format changes, all five entries need to change
- **Presumes exactly five cities**
  - Will not work with a different number of cities
    - Even though both the servlet code and the client-side JavaScript code can handle any number of cities
- **Solution**
  - Use JSTL

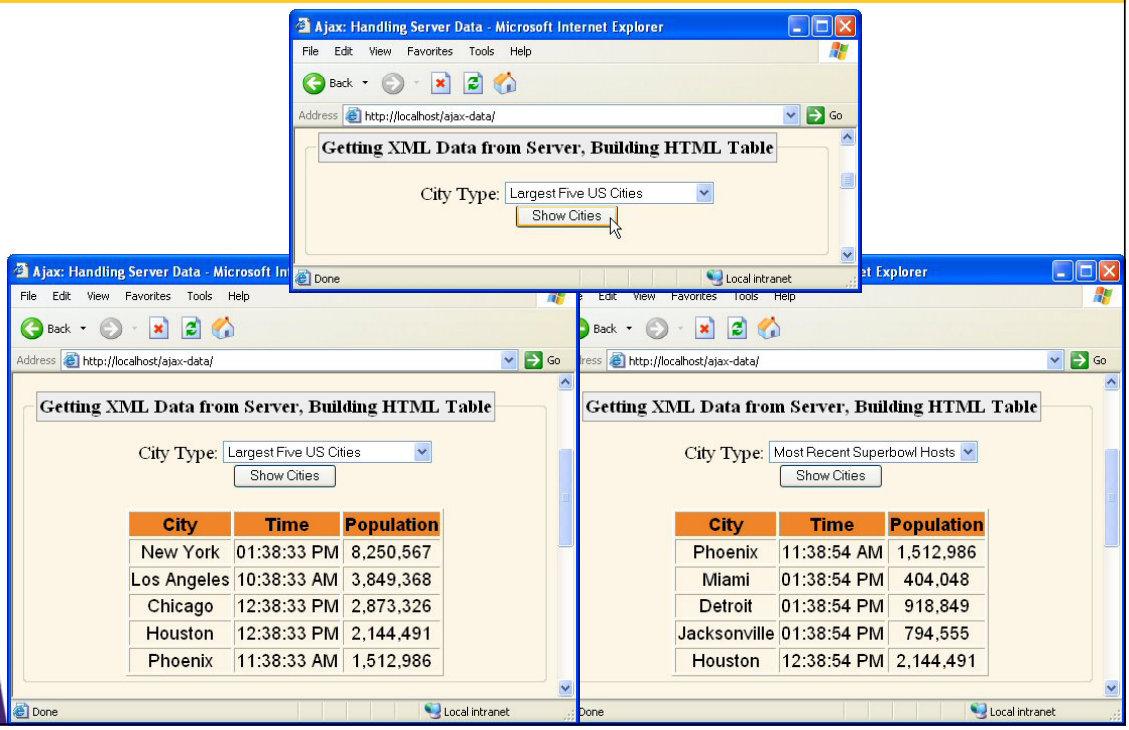
Updated JSP Code (cities-xml.jsp)

```
<?xml version="1.0" encoding="UTF-8"?>
<cities>
  <headings>
    <heading>City</heading>
    <heading>Time</heading>
    <heading>Population</heading>
  </headings>
<%@ taglib prefix="c"
           uri="http://java.sun.com/jsp/jstl/core" %>
<c:forEach var="city" items="${cities}">
  <city>
    <name>${city.name}</name>
    <time>${city.shortTime}</time>
    <population>${city.population}</population>
  </city>
</c:forEach>
</cities>
```


Alternative Version (Using XML Syntax: cities-xml.jspx)

```
<?xml version="1.0" encoding="UTF-8"?>
<cities xmlns:c="http://java.sun.com/jsp/jstl/core">
  <headings>
    <heading>City</heading>
    <heading>Time</heading>
    <heading>Population</heading>
  </headings>
<c:forEach var="city" items="${cities}">
  <city>
    <name>${city.name}</name>
    <time>${city.shortTime}</time>
    <population>${city.population}</population>
  </city>
</c:forEach>
</cities>
```

XML Data: Results



JSON Version: Original (before-jstl/cities-json.jsp)

```
{ headings: ["City", "Time", "Population"],
  cities: [["${cities[0].name}", "${cities[0].shortTime}",
            "${cities[0].population}"],
           ["${cities[1].name}", "${cities[1].shortTime}",
            "${cities[1].population}"],
           ["${cities[2].name}", "${cities[2].shortTime}",
            "${cities[2].population}"],
           ["${cities[3].name}", "${cities[3].shortTime}",
            "${cities[3].population}"],
           ["${cities[4].name}", "${cities[4].shortTime}",
            "${cities[4].population}"]]
}
```

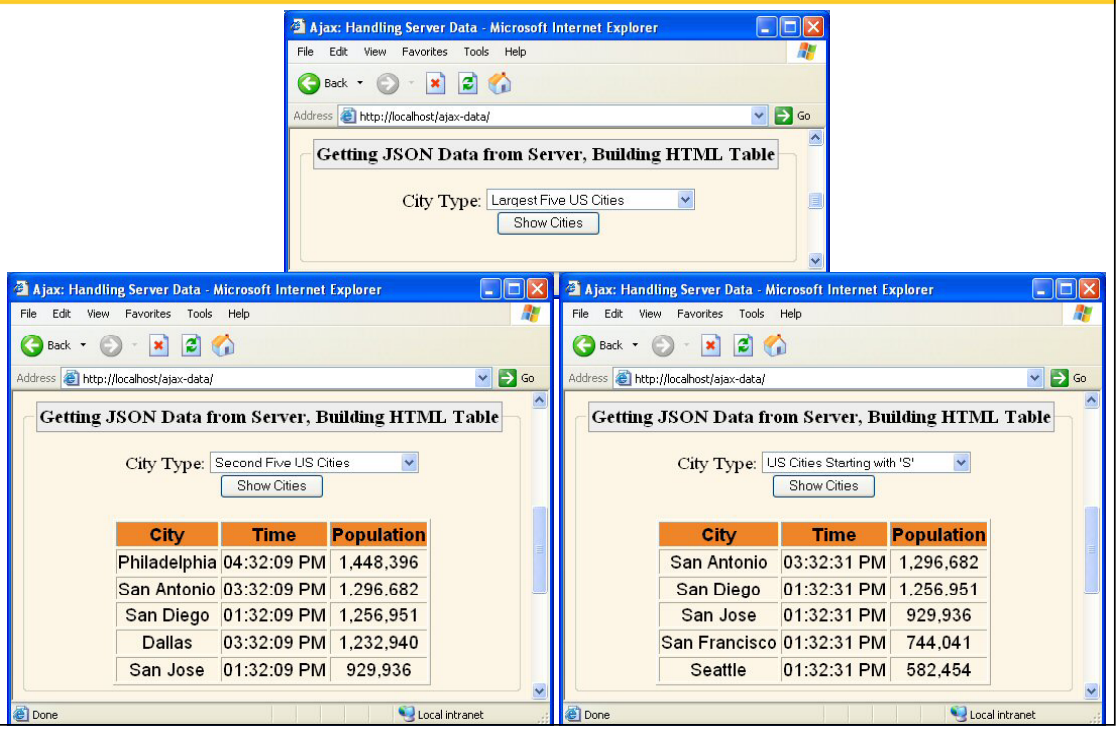
JSON Version: Updated (cities-json.jsp)

```
<%@ taglib prefix="c"
           uri="http://java.sun.com/jsp/jstl/core" %>
{ headings: ["City", "Time", "Population"],
  cities: [
  <c:forEach var="city" items="${cities}" varStatus="status">
    ["${city.name}","${city.shortTime}","${city.population}"]
    <c:if test="${!status.last}">,</c:if>
  </c:forEach>
  ]
}
```

This is important because Internet Explorer crashes for arrays and JSON objects with trailing commas. E.g., these crash in IE:

```
var nums = [1, 2, 3, ];
var obj = { a: 1, b: 2, c: 3, };
```


JSON Version: Results



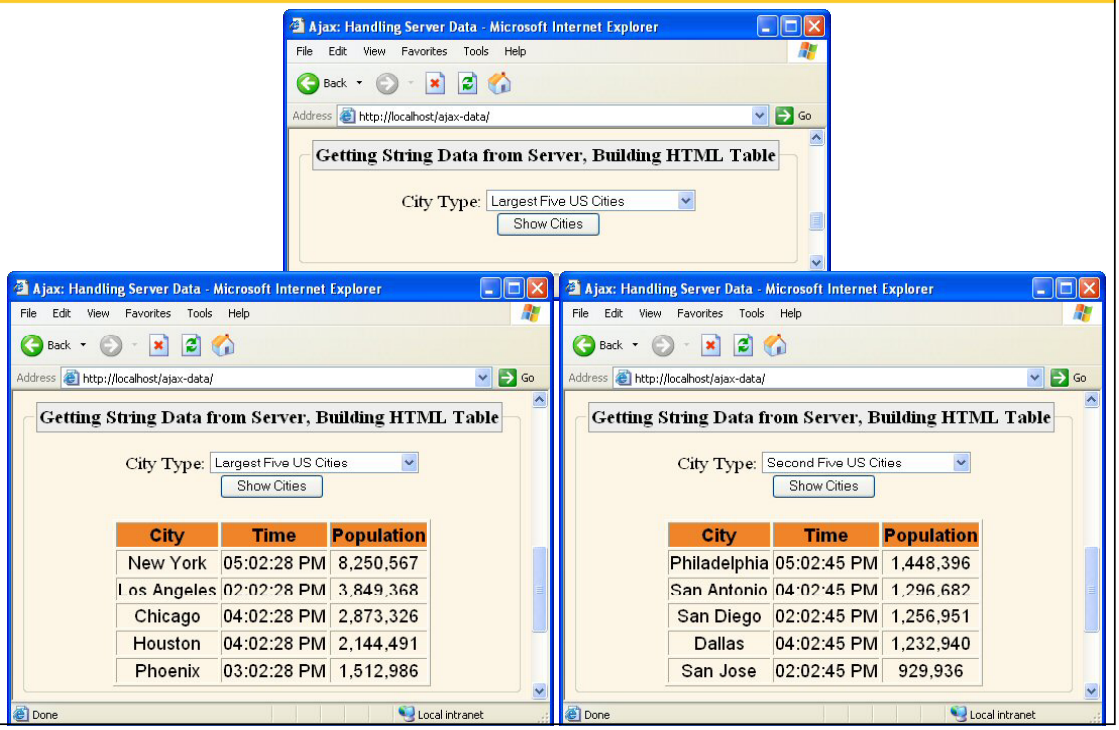
Delimited String Version: Original (before-jstl/cities-string.jsp)

```
City#Time#Population
${cities[0].name}#${cities[0].shortTime}#${cities[0].population}
${cities[1].name}#${cities[1].shortTime}#${cities[1].population}
${cities[2].name}#${cities[2].shortTime}#${cities[2].population}
${cities[3].name}#${cities[3].shortTime}#${cities[3].population}
${cities[4].name}#${cities[4].shortTime}#${cities[4].population}
```


Delimited String Version: Updated (cities-string.jsp)

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
           prefix="c" %>City#Time#Population
<c:forEach var="city" items="${cities}">
${city.name}#${city.shortTime}#${city.population}</c:forEach>
```

String Version: Results



Logic Tags

Conditional Evaluation Tags

- One choice: `if`

```
<c:if test="${someTest}">
  Content
</c:if>
```

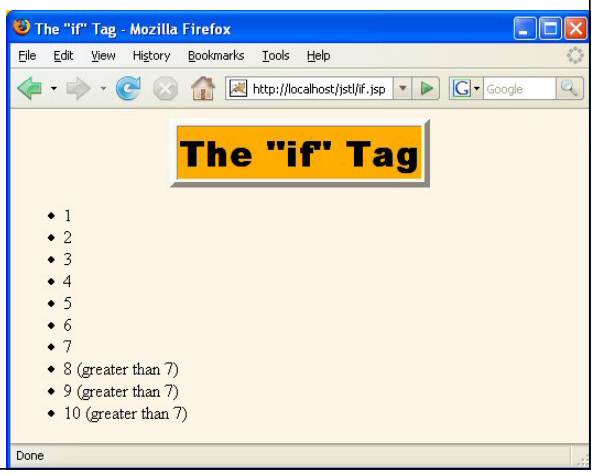
- Lots of choices: choose

```
<c:choose>
  <c:when test="test1">Content1</c:when>
  <c:when test="test2">Content2</c:when>
  ...
  <c:when test="testN">ContentN</c:when>
  <c:otherwise>Default Content</c:otherwise>
</c:choose>
```

- Caution: resist use of business logic!

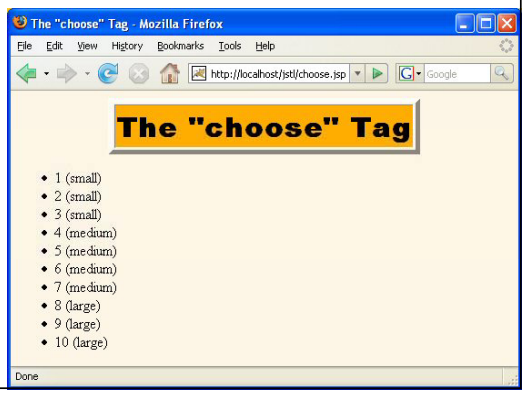
The "if" Tag

```
<%@ taglib prefix="c"
           uri="http://java.sun.com/jsp/jstl/core"%>
<UL>
<c:forEach var="i" begin="1" end="10">
  <LI>${i}
      <c:if test="${i > 7}">
      (greater than 7)
      </c:if>
</c:forEach>
</UL>
```



The "choose" Tag

```
<%@ taglib prefix="c"
           uri="http://java.sun.com/jsp/jstl/core"%>
<UL>
<c:forEach var="i" begin="1" end="10">
  <LI>${i}
    <c:choose>
      <c:when test="${i < 4}">
        (small)
      </c:when>
      <c:when test="${i < 8}">
        (medium)
      </c:when>
      <c:otherwise>
        (large)
      </c:otherwise>
    </c:choose>
</c:forEach>
</UL>
```



Using JSTL in JSF Pages

Output Pages vs. Direct-Access Pages

- **Output pages**
  - In pages that are the result of a navigation rule (i.e., listed under to-view-id), you can use JSTL as shown on previous slides
    - Since bean is *already* in request, session, or application
- **Direct access pages**
  - JSTL does not know about faces-config.xml, so you cannot refer to beans that are listed there but not yet instantiated
    - Using direct-access pages is rare in real life, but is very useful for testing when learning JSTL
    - Solution: use h:outputText to refer to bean once first
      - Declare bean containing array of numbers in faces-config.xml
      - First num: <h:outputText value="#{myBean.numbers[0]}/>
      - <c:forEach var="num" items="${myBean.numbers}>...

JSF Does Not Know About JSTL Variables

- **JSTL puts data in page scope**
- **This works**
  - <c:forEach var="name" items="${names}">
    <LI>${name}
    </c:forEach>
  - Example assumes this is an output page and "names" set earlier by JSF
- **This fails**
  - <c:forEach var="name" items="names">
    <LI><h:outputText value="#{name}"/>
    </c:forEach>
- **Supposedly fixed in JSTL 1.2**
  - But there are no standalone releases of JSTL 1.2
  - Use facelets instead

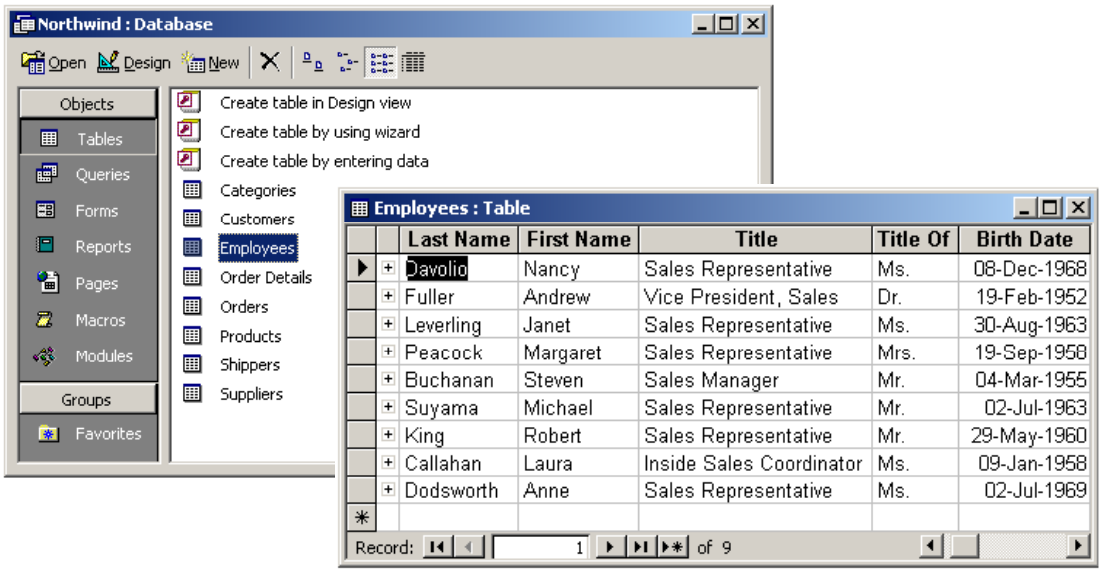
Database Access Tags

Database Access Tags

- **<sql:setDataSource>**
  - Specifies data source (can also be set by config settings)
- **<sql:query>**
  - Queries database and stores ResultSet in variable
  - Warning: this usage violates rule of keeping business logic out of presentation layer. Instead, do database access in servlet and pass results to JSP via MVC.
- **<sql:update>**
- **<sql:param>**
- **<sql:dateParam>**
- **<sql:transaction>**
  - Performs the enclosed <sql:query> and <sql:update> actions as a single transaction

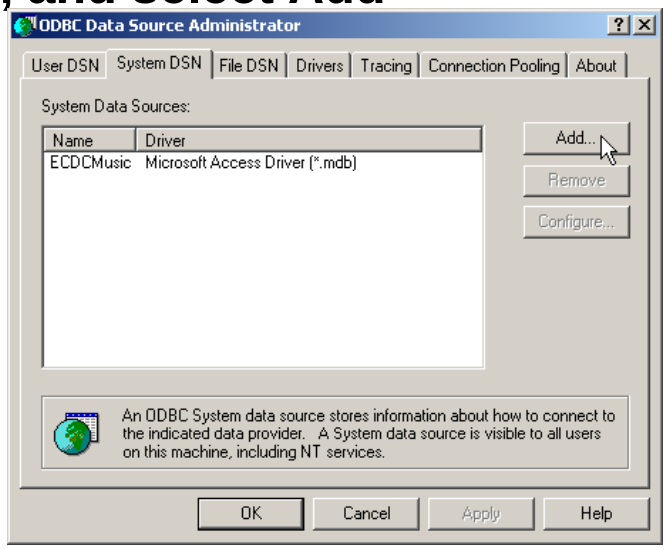
Aside: The Microsoft Access Northwind Database

- **Database that comes preinstalled with Microsoft Office**



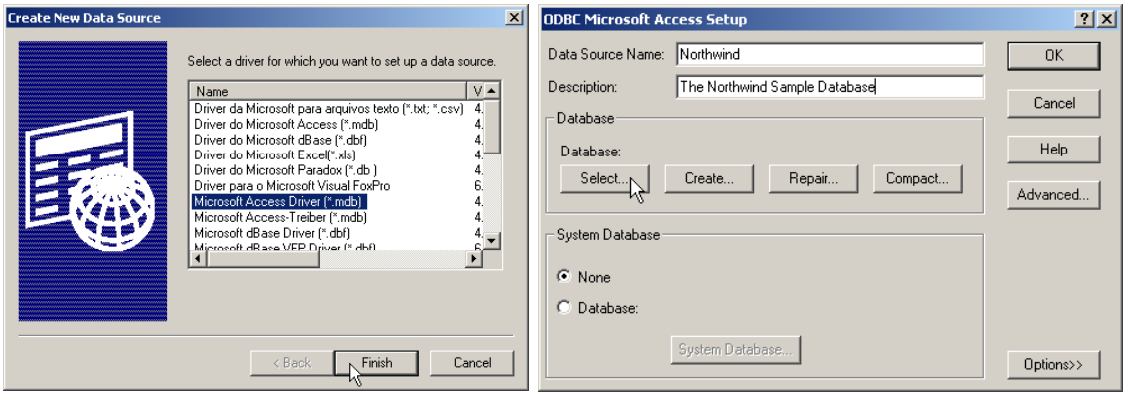
Using Microsoft Access via ODBC

- **Click Start, Settings, Control Panel, Administrative Tools, Data Sources, System DSN, and select Add**



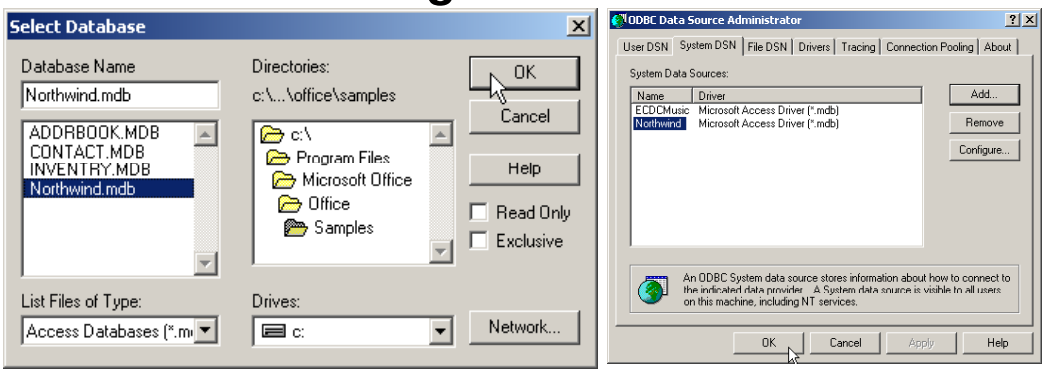
Using Microsoft Access via ODBC (Continued)

- **Select Microsoft Access Driver, Finish, type a name under Data Source Name, and hit Select**



Using Microsoft Access via ODBC (Continued)

- **Navigate to the Samples directory of MS Office, select Northwind.mdb, hit OK, then hit OK in following two windows**



Using Microsoft Access via ODBC (Continued)

- **Driver**
  - sun.jdbc.odbc.JdbcOdbcDriver
    - Comes bundled with JDK
- **URL**
  - jdbc:odbc:Northwind
    - Local access only; for testing. Not for serious applications.
- **Username**
  - Empty string
- **Password**
  - Empty string

sql:setDataSource

- **You can set a data source globally via configuration settings or application-scoped variables.**
  - Preferred approach in real applications
- **Or, you can set it on a specific page**

```
<%@ taglib prefix="sql"
           uri="http://java.sun.com/jstl/sql" %>
<sql:setDataSource
  driver="sun.jdbc.odbc.JdbcOdbcDriver"
  url="jdbc:odbc:Northwind"
  user=""
  password=""/>
```

sql:query

- **Form 1: use the sql attribute**

```
<sql:query var="results" sql="SELECT * FROM …"/>
```

- **Form 2: put the query in the body of the tag**

```
<sql:query var="results">
  SELECT * FROM …
</sql:query>
```

- **Options**
  - dataSource
  - maxRows
  - startRow
- **Caution**
  - Embedding SQL directly in JSP may be hard to maintain.

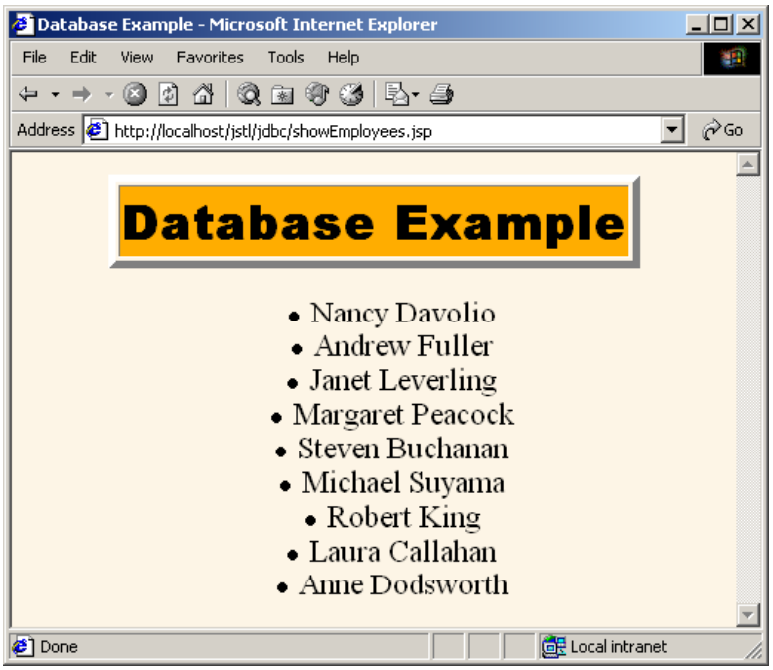
Simple Example

```
<%@ taglib prefix="c"
           uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="sql"
           uri="http://java.sun.com/jstl/sql" %>
<sql:setDataSource
  driver="sun.jdbc.odbc.JdbcOdbcDriver"
  url="jdbc:odbc:Northwind"
  user=""
  password=""/>
```

Simple Example (Continued)

```
<sql:query var="employees">
  SELECT * FROM employees
</sql:query>

<UL>
<c:forEach var="row" items="${employees.rows}">
  <LI>${row.firstname}
      ${row.lastname}
</c:forEach>
</UL>
```


Simple Example: Results



Other Tags

URL-Handling Tags

- **<c:import>**
  - Read content from arbitrary URLs
    - Insert into page
    - Store in variable
    - Or make accessible via a reader
  - Unlike <jsp:include>, not restricted to own system
- **<c:redirect>**
  - Redirects response to specified URL
- **<c:param>**
  - Encodes a request parameter and adds it to a URL
  - May be used within body of <c:import> or <c:redirect>

Formatting Tags

- **<fmt:formatNumber>**
  - Formats a numeric value as a number, currency value, or percent, in a locale-specific manner
- **<fmt:parseNumber>**
  - Reads string representations of number, currency value, or percent
- **<fmt:formatDate>**
- **<fmt:parseDate>**
- **<fmt:timeZone>**
- **<fmt:setTimeZone>**

Internationalization (I18N) Tags

- **<fmt:setLocale>**
  - Sets Locale
- **<fmt:setBundle>**
- **<fmt:bundle>**
- **<fmt:message>**
  - Retrieves localized message from a resource bundle
- **<fmt:param>**

XML Manipulation Tags

- **Core**
  - <x:parse>
- **XPath**
  - <x:if>
  - <x:choose>
  - <x:when>
  - <x:otherwise>
  - <x:forEach>
- **XSLT**
  - <x:transform>
  - <x:param>

Summary

- **JSTL overview**
  - JSTL is similar to the old Struts looping and logic tags, but better
  - JSTL is standardized, but not a standard part of JSP
    - You have to add JAR files to WEB-INF/lib
    - It is supposed to be included with all JSF implementations
- **Supports a concise expression language**
  - Lets you access bean properties and implicit objects
  - EL is standard part of JSP 2.0, JSP 2.1, and JSF
- **Looping tags**
  - Explicit numeric values
  - Arrays, collections, maps, strings, etc.
- **Conditional evaluation tags**
  - Single options
  - Multiple options

Summary (Continued)

- **Database Access Tags**
  - *Always* a bad idea when using MVC approach
    - With or without a framework like JSF or Struts
  - sql:setDataSource to specify a database
  - sql:query to perform query
  - Loop down results using iteration tags
- **Other tags**
  - Handling URLS
  - Internationalizing pages
  - Formatting strings
  - Manipulating XML
    - Bad idea when using MVC

Questions?