JSP
Java Server Pages

A “prettier” form of Servlet technology

Servlets

- Sun expected Servlets to replace CGI
- Basically …
- They didn't.

ASP & PHP

- Microsoft's Active Server Page technology proved more popular – though limited to Microsoft IIS server
- PHP came from nowhere and easily outgrew servlets – in number of sites, if not importance of sites

ASP & PHP advantage

- If done well, separates
  - HTML display concerns
  - Coding concerns

OK, a large fraction of ASP and PHP pages fail to exploit potential for separation

Sun's response …

- Java Server Pages
  - Basically a HTML document
  - Embedded
    - Directives
    - Actions
    - Scriptlets
  - These embedded elements invoke processing
    - Return data strings that get embedded into the HTML

JSP & Servlets

- JSP is a servlet technology
- JSP page "compiled into a servlet" before use.
- Often combined with other servlets
  - Servlets do the "heavy lifting", results left in shared data structures
  - Servlet forwards control to JSP page
  - JSP page adds the "pretties“
    - Data extracted from shared data structures and embedded in amongst the “pretties”

JSP as a Servlet

- JSP pages run as servlets
- So need a servlet engine
- Will again use Glassfish (AppServer) as our engine

- Deployment of JSP identical to deployment of servlet –
  - Same directory structure
  - Similar XML deployment descriptor (*not actually needed in simple examples*)
  - Use of "war" file
  - …

Oh, yes … ***beans***

- Data returned for use in a script has to be packaged in structures that the script can access.
- Need simple access scheme (to minimize code in JSP page)
- "JavaBeans" style adopted
- Most of "Java things" used in a JSP page will be instances of "bean" classes

Beans?

- Beans?
  - Originally conceived as scheme that would make it easier to create editor programs (authoring systems) that can manipulate Java GUI components
  - Scheme included Java reflection architecture (mechanisms used to allow editor program to determine functionality of GUI component – so allowing it to create appropriate menus for developer to view etc etc)
  - Major aspect of interest here ***Conventions for naming access functions of a "data structure" – follow the conventions, and code using these structures can be generated auto-magically***

Beans: get… set…

- Standard naming
  - getXXX()
    - Read value of XXX property (field in data structure or something calculated)
  - setXXX(…)
    - Write new value into XXX property
- Can become quite automated
  - Later see examples of JSP form handlers
    - Each named form element corresponds to property in a structure (instance of "bean" class)
    - Single JSP directive can set the lot – expanded code gets Form parameters and invokes matching setXXX() for each property
    - JSP script can then ask Bean to validate data, save to DB etc

Just one more thing before we start

- *Be patient.!*
- *JSP pages are slow to start first time they are used*
  - *Your computer hasn't crashed!*
  - *JSP has to be processed to create source code of servlet*
  - *Servlet has to be compiled*
  - *(You can specify a load on startup option, so that no customer gets annoyed by slow startup)*

Oh, yes, another thing …

- *Servlets generated for JSP pages are created in subdirectories of the "work" directory in main Tomcat /Appserver work area.*
- *This directory soon fills up with lots of things best hidden in kitty-litter – you are responsible for keeping this area clean, it's not automatic.*

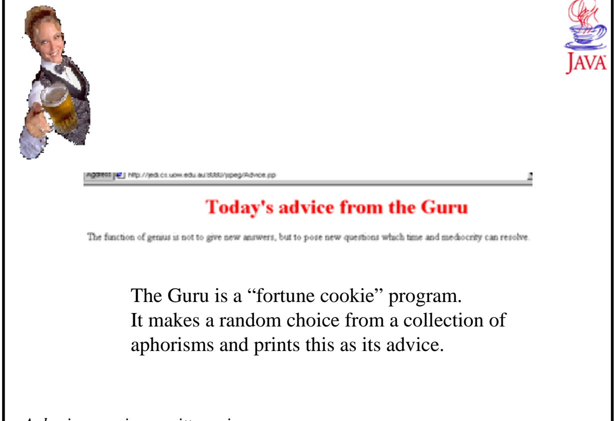
NetBean's style

- As with servlets, simplest to develop and test in NetBean's environment using embedded Tomcat.

The adventures first, explanations take too much time

The Guru

A better class of
"Hello World"
Program



The Guru is a "fortune cookie" program.
It makes a random choice from a collection of aphorisms and prints this as its advice.

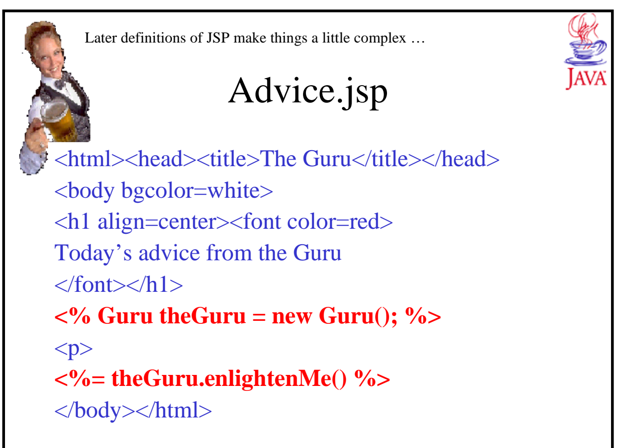
Aphorism: a wise or witty saying

Advice.jsp & Guru.java

- JSP
  - Using "scriptlets" – small fragments of Java used to invoke operations of an object
- A simple semi-bean class "Guru"
  - Only one method defined – String enlightenMe()
- No XML deployment file needed
- Usual directory configuration; in "webapps"
  - Jspeg
    - Advice.jsp
    - WEB-INF
      - Classes
        - » Your Java package
          - - Guru.java/Guru.class

Simplified

- *Example relies on default parameters applying to JSP pages*
- *Typical JSP page has to contain additional directives to set these parameters*
- *Don't base real web-apps on simplified code, use more realistic examples shown later.*



Guru.java

```
import java.util.Random;
public class Guru {
    private static Random rr;
    private static final String[ ] Sayings = {
    “Women like silent men. They think they're listening.”,
    “It has been discovered that C++ provides a remarkable facility for concealing the trivial details of a program - such as where its bugs are. “
            …
    “Inside every older person is a younger person wondering what the hell happened “
    };
    public String enlightenMe() {
            int select = rr.nextInt(Sayings.length);
            return Sayings[select];
    }
    {
            rr = new Random();
    }
}
```

Generated servlet : imports

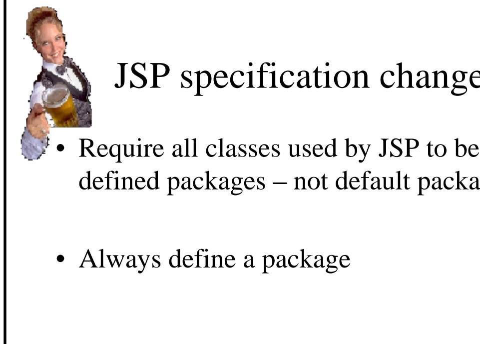
```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.PrintWriter;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.util.Vector;
import org.apache.jasper.runtime.*;
import java.beans.*;
import org.apache.jasper.JasperException;
```

Generated servlet

```
public class _0002fAdvice_0002ejspAdvice_jsp_0 extends
    HttpJspBase {
    static {   }
    public _0002fAdvice_0002ejspAdvice_jsp_0( ) {   }
    private static boolean _jspx_inited = false;
    public final void _jspx_init() throws JasperException {  }
    public void _jspService(HttpServletRequest request,
                    HttpServletResponse  response)
            throws IOException, ServletException {
        …
    }
}
```

jspService method

```
public void _jspService(HttpServletRequest request,
    HttpServletResponse  response) throws … {
        JspFactory _jspxFactory = null; PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null; ServletConfig config = null;
        JspWriter out = null; Object page = this; String  _value = null;
        try {
            …
        } catch (Exception ex) {
            if (out.getBufferSize() != 0) out.clearBuffer();
            pageContext.handlePageException(ex);
        } finally { out.flush();
          _jspxFactory.releasePageContext(pageContext);
        }
  }
```

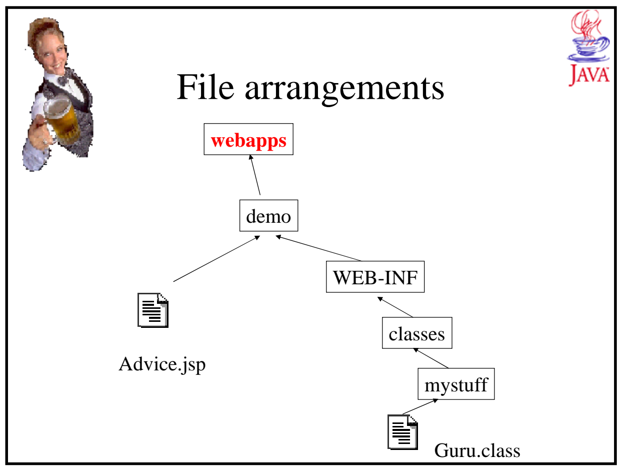


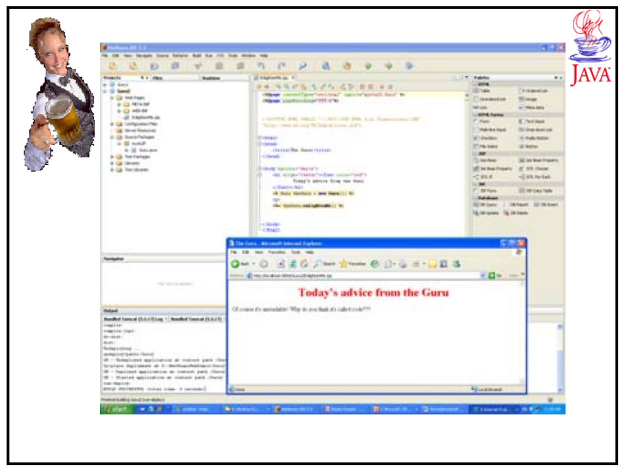
Import your package – "mystuff"

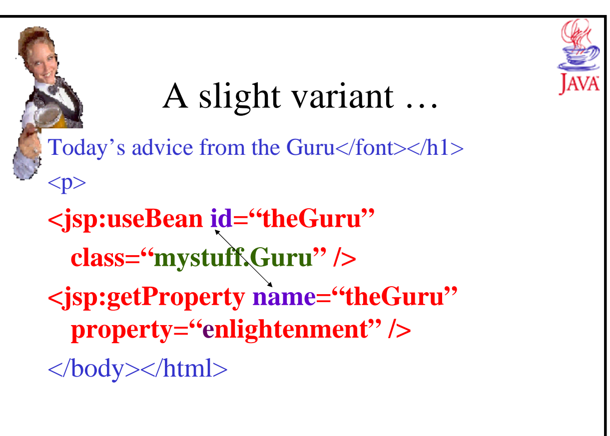
```
<%@ page import="mystuff.Guru" %>
<html><head><title>The Guru</title></head>
<body bgcolor=white>
<h1 align=center><font color=red>
Today's advice from the Guru
</font></h1>
<% Guru theGuru = new Guru(); %>
<p>
<%= theGuru.enlightenMe() %>
</body></html>
```

Guru class – member of package

```
package mystuff;
import java.util.Random;
public class Guru {
  private static Random rr;
  private static final String[ ] Sayings = {
    };
  public String enlightenMe() {
      int select = rr.nextInt(Sayings.length);
      return Sayings[select];
  }
  {
      rr = new Random();
  }
}
```

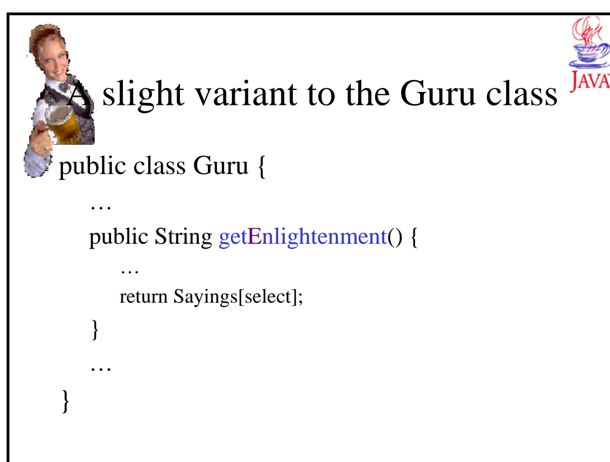






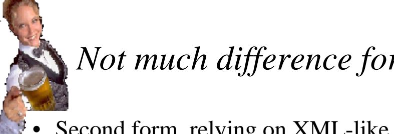
A slight variant …

```
Today's advice from the Guru</font></h1>
<p>
<jsp:useBean id="theGuru"
  class="mystuff.Guru" />
<jsp:getProperty name="theGuru"
  property="enlightenment" />
</body></html>
```



A slight variant to the Guru class

```
public class Guru {
    …
    public String getEnlightenment() {
        …
        return Sayings[select];
    }
    …
}
```



Not much difference for now…

- Second form, relying on XML-like “action tags” rather than scriptlet code, is “better”.
- As move to more complex examples, will try to use tag style more.
- Advantage?
  - Well, in more complex cases these tags do make a difference and do reduce the amount of code in the JSP page.

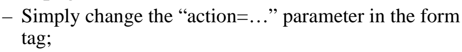
SubscriberRecord - 1

- Remember this from servlets …

SubscriberRecord - 1

- Phase 1:
  - JSP page uses “beanified” SubscriberRecord class
    - Instantiate a SubscriberRecord bean that can be passed to other JSP pages or to Servlets
    - Fill the data fields
    - For this first step, just print out contents of fields

Files …



- “infopage.html”
  - Simply change the “action=…” parameter in the form tag; should now reference Subscriber.jsp
  - *Well, also carefully check the names of input fields (lots of case sensitivity restrictions …)*
- Subscriber.jsp
  - Create SubscriberRecord object
  - Fill it with data from form
  - Print its contents
- SubscriberRecord.java
  - Some modifications to comply with “bean” style guidelines.

JSP page

```
<%@ page language="java"
   contentType="text/html"  session="false" %>
<%@ page import="mystuff.*" %>
<jsp:useBean scope="request" id="userInfo"
   class="mystuff.SubscriberRecord">
   <jsp:setProperty name="userInfo"
   property="*" />
</jsp:useBean>
<html><head><title>OK</title></head><body>
<ul>
```

JSP page

```
<ul>
<li>
<jsp:getProperty name="userInfo" property="givenName" />
<li>
<jsp:getProperty name="userInfo" property="familyName" />
<li>
<jsp:getProperty name="userInfo" property="age" />
<li>
<jsp:getProperty name="userInfo" property="sex" />
<li>
<jsp:getProperty name="userInfo" property="givenName" />
<li>
<jsp:getProperty name="userInfo" property="email" />
</ul></body></html>
```

SubscriberRecord.java

```
public class SubscriberRecord  {
   private String  GivenName;
   …
   public boolean isValid() { … }
   public String getGivenName() { return GivenName; }
   public void setGivenName(String aName) { … }
   public String getFamilyName() { return FamilyName; }
   public void setFamilyName(String aName) { … }
   public String getEmail() { return Email; }
   public void setEmail(String aName) { … }
   public String getSex() { return Sex; }
   public void setSex(String gender) {  … }
   public int getAge() { return  Age; }
   public void setAge(String AgeStr) { … }
   …
}
```

Gotcha (1) !

- Watch those rules about first letter capitalization.
- Form field must be “sex”
- Methods must be “setSex”, “getSex”
- Property name in action tag on JSP page must be “sex”

- *But if it was “SEx” you would use “SEx” everywhere.*

Gotcha (2) !

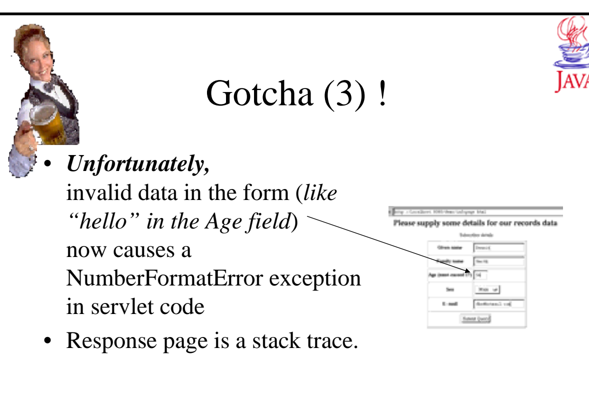
```
public int getAge() { return  Age; }
public void setAge(String AgeStr) { … }
```

- Age is not set.
- Code generation system uses signature of “get” method to guide its operation.
  - int getAge()
  - Therefore, must be void setAge(int)

OK try …

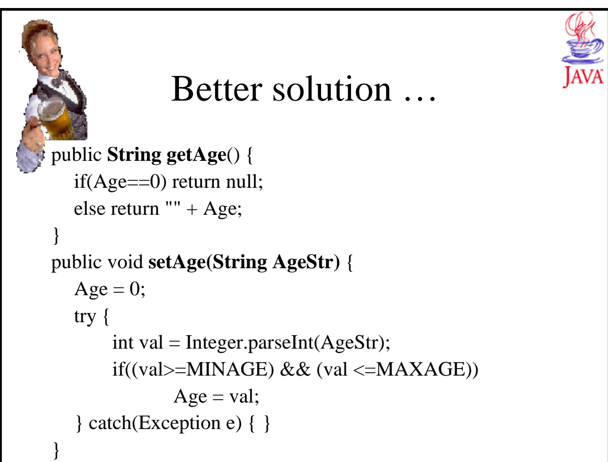
```
public void setAge(int val) {
   if((val>=MINAGE) && (val <=MAXAGE))
         Age = val;
}
public int getAge() { return  Age; }
```

- Generated code
  - grabs string from form’s <input name=age … > (age=33& etc of GET or POST param string)
  - Does an Integer.parseInt(…) to get int argument for setAge
  - Invokes public void setAge(int)



Gotcha (3) !

- ***Unfortunately,*** invalid data in the form (*like "hello" in the Age field*) now causes a NumberFormatError exception in servlet code
- Response page is a stack trace.



Better solution …

```
public String getAge() {
    if(Age==0) return null;
    else return "" + Age;
}
public void setAge(String AgeStr) {
    Age = 0;
    try {
        int val = Integer.parseInt(AgeStr);
        if((val>=MINAGE) && (val <=MAXAGE))
            Age = val;
    } catch(Exception e) { }
}
```

Exceptions …

- Default response is exception printout and stack trace being returned to client browser.
- Can trap exceptions and divert them to a special "exception handling" page.
  - Pick up data about exception in "Exception" object that can be interrogated.

Bouncing exceptions to a reporter page

- <%@page isErrorPage="false" errorPage="exceptions.jsp" %>

exceptions.jsp

```
<%@ page isErrorPage="true" %>
<html><head><title>Errors!</title></head>
<body bgcolor=red>
<h1 align=center>Run time errors</h1>
<p>The program hit a problem and an exception was thrown.
<p>Tell the programmer that the exception was
<%= exception.toString() %>
</body></html>
```

Move on to SubscriberRecord-2

- Processing becomes more elaborate:
  - Create a SubscriberRecord
  - Fill it in
  - Ask it whether it was "valid"
  - If it wasn't,
    forward SubscriberRecord object to
    another page where generate error report
  - If valid
    get it to save itself to DB
    report membership number

SubscriberRecord-2

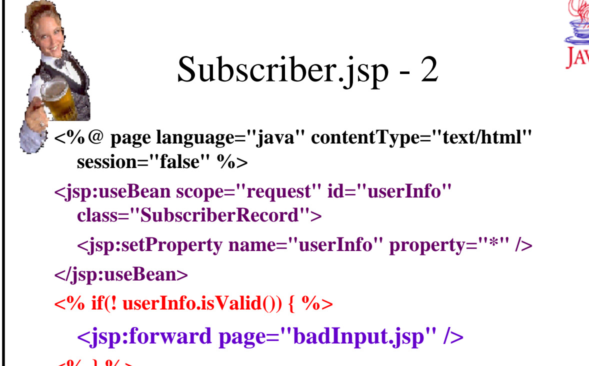
- In terms of JSP, main new features are
  - collaboration amongst generated servlets,
  - Passing of generated "bean" amongst servlets
- As for the "SubscriberRecord" bean, new feature is
  - Ability to work with database.

SubscriberRecord - 2

- Get a mix of "scriptlets" and "actions"
- Difficult to avoid scriptlet code at places where have to make conditional tests and select different processing steps.¶
- The example doesn't use an error page to handle exceptions, all problems that might occur in "bean" are handled there – it simply sets "invalid" status flags etc.

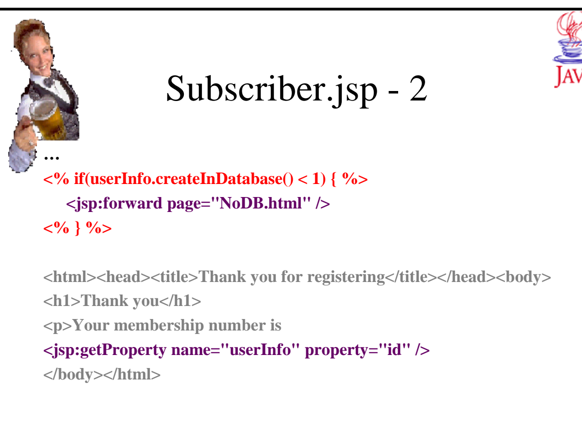
¶Well you can avoid scriptlet, you use "custom tags" such as the logic tags from Apache.org's "struts" tag set or the "standard tags" from JSTL.

Subscriber.jsp - 2



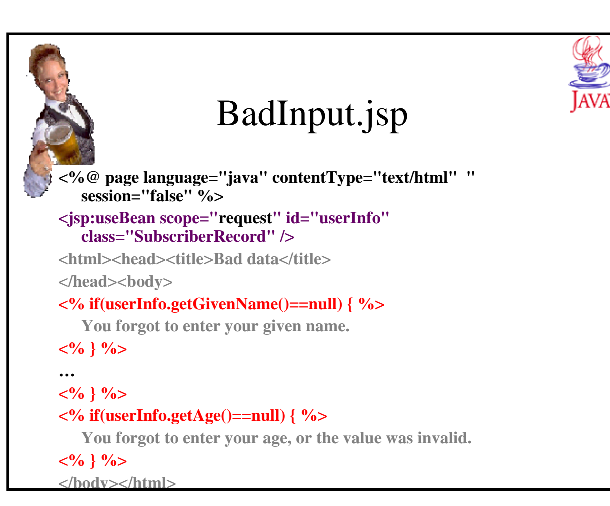
```
<%@ page language="java" contentType="text/html"
    session="false" %>
<jsp:useBean scope="request" id="userInfo"
    class="SubscriberRecord">
    <jsp:setProperty name="userInfo" property="*" />
</jsp:useBean>
<% if(! userInfo.isValid()) { %>
    <jsp:forward page="badInput.jsp" />
<% } %>
```

Subscriber.jsp - 2



```
...
<% if(userInfo.createInDatabase() < 1) { %>
    <jsp:forward page="NoDB.html" />
<% } %>

<html><head><title>Thank you for registering</title></head><body>
<h1>Thank you</h1>
<p>Your membership number is
<jsp:getProperty name="userInfo" property="id" />
</body></html>
```

BadInput.jsp



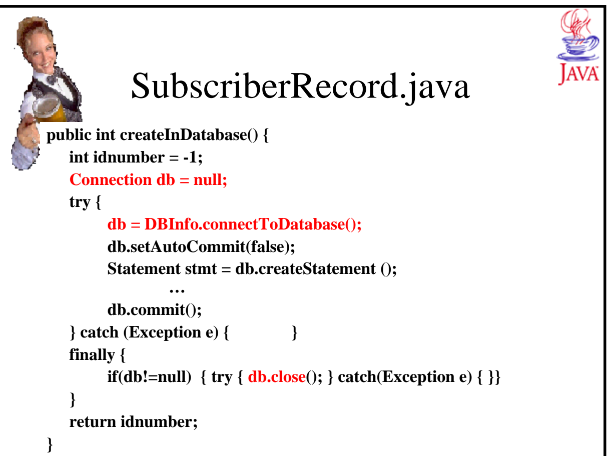
```
<%@ page language="java" contentType="text/html"  "
    session="false" %>
<jsp:useBean scope="request" id="userInfo"
    class="SubscriberRecord" />
<html><head><title>Bad data</title>
</head><body>
<% if(userInfo.getGivenName()==null) { %>
    You forgot to enter your given name.
<% } %>
...
<% } %>
<% if(userInfo.getAge()==null) { %>
    You forgot to enter your age, or the value was invalid.
<% } %>
</body></html>
```

SubscriberRecord.java

- The earlier Servlet example had a database connection that belonged to the servlet and which was passed to a SubscriberRecord that had to save or load its state.
- Could do the same for JSP page
  - Declare a "page variable" to hold a reference to the connection
  - Provide an "initialization" function for the page, this function would open the connection.
  - Pass db connection as argument in scriptlet code that invokes DB operations.

SubscriberRecord.java

- Chose instead to let the SubscriberRecord open its own connection to a database when it needed it.
- Might be too costly in a system that handled many registrations; but not a problem if registration rate is only a few per hour.



SubscriberRecord.java

```
public int createInDatabase() {
    int idnumber = -1;
    Connection db = null;
    try {
        db = DBInfo.connectToDatabase();
        db.setAutoCommit(false);
        Statement stmt = db.createStatement ();
            …
        db.commit();
    } catch (Exception e) {        }
    finally {
        if(db!=null)  { try { db.close(); } catch(Exception e) { }}
    }
    return idnumber;
}
```

OK, time to get more technical

Things your find in JSP pages

Beans

Scopes for Beans

Sessions

etc etc

Page contents

- Directives
- Action elements
- Scripting elements
- Template text

Directives

- Essentially, these provide information to the component that generates the servlet from the JSP page.
  - page attributes
    - Is a Session to be used for state maintenance?
    - How are output data buffered?
    - Is there a page that can be invoked if an exception occurs?
    - …
  - includes
    - Fragments of prepared text from other files
  - taglib
    - Specify library containing custom action elements that supplement the standard JSP tags

Action elements

- These are the XML-like tags that can appear.
- There is a **standard set of action elements** – provided via the **jsp** tags
- These standard tags support
  - common actions on "beans"
  - some actions on the servlet request, and response objects.
  - Control collaboration with other jsp pages and/or servlets
- Standard action elements include:
  - <jsp:useBean>
  - <jsp:getProperty>, <jsp:setProperty>
  - <jsp:include>, <jsp:forward>
  - <jsp:param>

Custom action elements

- You aren't limited to the jsp action elements, you can define your own.
- An action element appears as something like

  <alib:thing1 parameters, more parameters>

  Block of text

  </alib:thing1>
- This block is mapped into code in the generated servlet that is (conceptually) along the lines –

```
import alib.*;
…
    alib.Thing1 t001 = new alib.Thing1(params);
    t001.getStarted();
    t001.handleBodyText("Block of text");
    t001.finishUp();
```

Custom tags

- You have to define your "alib" tag library (~ Java package of tag classes)
- You have to define classes that do whatever special processing your require.
  - These classes must extend various base tag classes that are provided as part of the JSP system.
- Typically, processing occurs mostly at end tag, there generate block of HTML text derived from information in parameters and body data.

- *Well, really, you don't define your own custom tags.*
- *You make use of other peoples' tag libraries such as the "Struts" library from www.apache.org*

Scripting elements

- Scriptlet

  some Java code

  The code is dropped into the generated servlet.

  Can include variable declarations – they are defined as local variables of the "service" function.
- Expression

  `<%= Java expression %>`

  gets translated into "out.print(Java expression)"
- Declarations
  - …

Scripting elements contd.

- Declarations –
  - You can declare instance data members and member functions for the generated servlet.
  - What might you want?
    - Suppose that you decided that the servlet should own a database connection, and that this would be opened at initialization time and closed and destroy time
    - Then you would define
      - java.sql.Connection dbConnection;
      - public void jspInit() { /* connect to database */ … }
      - public void jspDestroy() { /* close that connection */ … }

Scripting elements contd.

- Declarations appear as

```
<%!
  java.sql.Connection dbConnection;
  public void jspInit() {
      …
  }
  // etc
%>
```

Template text

- All the standard HTML and content text fragments in the JSP page are referred to as "template text".
- These text fragments are simply packed into a large number of output statements in the generated service function.

  out.print(*bit more html*);

  out.print(*bit more content text*);

  out.print(*more html*);

  out.print(*and still more html*);

A JSP page

```
<%@ page language="java" contentType="text/html"   session="false" %>
<jsp:useBean scope="request" id="userInfo" class="SubscriberRecord">
        <jsp:setProperty name="userInfo" property="*" />
</jsp:useBean>
<% if(! userInfo.isValid()) { %>
        <jsp:forward page="badInput.jsp" />
<% } %>
<% if(userInfo.createInDatabase() < 1) { %>
        <jsp:forward page="NoDB.html" />
<% } %>

<html><head><title>Thank you for registering</title></head><body>
<h1>Thank you</h1>
<p>Your membership number is
<jsp:getProperty name="userInfo" property="id" />
</body></html>
```

Directive

Action tag

Scriptlet

Template text

Directives

- **<%@ directiveName attr1="value" attr2="value" %>**
  - **include**
    - URI for file to include
      - Can include chunks of HTML/content text from other files
  - **taglib**
    - Prefix
      - Group name for the tags, equivalent to "jsp:"
    - URI for tag library
      - Tag library should be in the WEB-INF directory and should be described more in the web.xml file

Page directive

- <%@ page … %>
  - Lots of attributes!
  - Can use more than one page directive.
- Attributes (*some less important ones first*):
  - language
    - Defaults to Java, and for most systems there is only Java; some JSP implementations allow Javascript code fragments instead
  - extends
    - You can define your own servlet base class, a refinement of HttpJspPage; but it would be unusual to want to do this
  - info
    - Text describing your JSP page (for use in sophisticated development environment)
  - isThreadSafe
    - Set to false if really want a "single thread model" servlet

Page directive

- More attributes of a page directive …
  - isErrorPage (*true/**false***)
    - Error page has "Exception" object defined, script reports errors as encoded in this object
  - contentType
    - Used to set content type ("text/html", or if you are ambitious "img/gif")
  - import
    - Reference java package used in code in page
  - session (***true**/false*)
    - Default is JSP pages are parts of session, with cookie used to carry session id

Page directive

- Still more attributes
  - errorPage
    - URL of JSP error page
  - autoflush (***true**/false*)
  - buffer (buffer=nkb, default 8kb)

Page buffering

- Output from JSP normally buffered before being sent to client
- Allows you to start generating output, then maybe modify a header (can't change http headers if output "committed" by some of it being written to back to web-server and thence to client)
- By default, output buffers flush and then can be refilled,
can change so that buffer full causes an error exception (*device to prevent generation of excessive output*)

Session

- Default is for JSP pages to check for session, and start session if none defined.
- Session relies on cookie based session identifier, *you should explicitly encode session identifier into all URLs in order to cater for clients who don't like cookies.*
- Turn off sessions if not really needed (many informational pages don't need session support)

jsp: actions, useBean

- <jsp:useBean … />
- Or
- <jsp:useBean …> …body … </jsp:useBean>

- Basically, instantiate bean; parameters set properties. Body code would be related initialization steps
  - Scriptlet style
  - Invocation of other action tags
  - (Can have some template text, just becomes more outputs to response output stream)

jsp:useBean

- Parameters
  - class
    - String with fully qualified name for bean class (if bean is class defined in WEB-INF/classes, then no package name qualification needed)
  - id
    - Variable name for the object that is being created (for use in scriptlet code etc)
  - scope
    - Page, request, session, application
  - Also
    - beanName
    - type

Bean scope is important, more shortly!

useBean

- Checks scope specified for instance of specified class associated with identifier
  - If already exists, return reference
  - Else create object

- (Object could exist – something bound to session could have been set by any other servlet/jsp-page involved in same session)

jsp:getProperty

- <jsp:getPropety name="…" property="…">
- Parameters
  - name
    - Object's name, as specified in <jsp:useBean id=".."
  - property
    - Property name, remembering rules regarding capitalization, matching "get" function etc

jsp:setProperty

- <jsp;setProperty name=… property=…
  param=… value=… />
- Parameters
  - name, property object identifier and property;
  - param value
    - Alternative ways of specifying value;
      - param data comes from a HttpRequest parameter, either use param="…" to specify parameter or follow defaults of matching names
      - value data comes from a bit of scriptlet code

jsp:setProperty

- setXXX(XType …)
  - Usually will have a String value from form parameter, so need String to XType converstion
    - Boolean.valueOf(String)
    - Integer.valueOf(String)
    - …

jsp:forward jsp:include

- <jsp:forward page="…">
  - Discard any buffered partial response and bounce request to specified page
- <jsp:include page="…">
  - Flush buffer with partial response (committing and sending headers)
  - Invoke other page, letting it output some data
  - Resume processing

Bean scope

- Bean is just an object instantiated from specified class.
- How is bean object defined?
  - Local variable of "doService()" function of generated servlet, this is **page scope** and is the default
  - Variable added to session object using session's setAttribute function,
this is **session scope**;
variable can be retrieved from session by other servlets or jsp's in related group
  - Variable added to HttpRequest object using its setAttribute function,
this is **request scope**,
gets passed with request if JSP forwards or includes other servlets
  - Variable added to servlet context object using its setAttribute function
this is **application scope.**

Servlet, Beanie, and JSP Examples

Strategy

- Code in the JSP page should be limited to that controlling aspects of data display; code expressing business logic, and code for tasks such as database access, should be in supporting beans (or servlets).
- Action tags are preferred over scriptlet coding.

Soccer league example

- From earlier example:
  - Have simple database with results of games in some imaginary soccer league
    - Table: Team1, Team2, Score1, Score2
      - No primary key – would need something based on location and date of match, but not bothering with such detail
  - Support queries (only in this example)
    - List all
      - Draws
      - Home wins
      - Away wins
      - All games

Soccer league system

- JSP page:
  - Gets query
  - Creates a bean that will organize search
  - Passes search type as parameter to bean
  - Requests that bean perform search
- Helper bean (SoccerSearchBean)
  - Connect to database
  - Run a search, collect matching results as simple SoccerGame structures stored in a vector
- JSP page:
  - Picks up iterator for results vector
  - Prints details of each successive SoccerGame returned by iterator

No permanent DB connection? OK, it's going to be used lightly!

Soccer

- Components
  - Soccer.html
    - Simple static page that allows user to request search
  - Soccer.jsp
    - Supposedly highly graphic attractive page that presents results of search
  - Beans (and other support classes)
    - SoccerSearchBean
    - SoccerGame
    - DBInfo
  - *Look at simpler parts first …*

HTML page

- Simply alinks to JSP page with query strings specifying search option …

```
<html><head><title>Soccer searcher</title></head>
<body bgcolor=white>
<h1 align=center>Search the little soccer league table
</h1><p align=center>
<a
href="http://localhost:8080/jspeg/Soccer.jsp?searchType=all">Li
  st all games</a><br>
<a
href="http://localhost:8080/jspeg/Soccer.jsp?searchType=drawn">
  List drawn games</a>
<br>
…
```

DBInfo

- You have to store names of data-sources, user-account names, passwords somewhere.
- Here using simple static class, which also provides connecToDatabase() function

```
import java.sql.*;
public class DBInfo {
  public static final String userName = "…";
  …
  private static final String dbURL = "…";
  public static final Connection connectToDatabase() {
    Connection dbConnection = null;
    try { …
      dbConnection = DriverManager.getConnection(…);
    } catch(Exception e) {  }
    return dbConnection;
```

SoccerGame

- Owns
  - Details of one game
- Does
  - Fills its data from a row in a result-set
  - Allows read access to its data members

SoccerGame.java

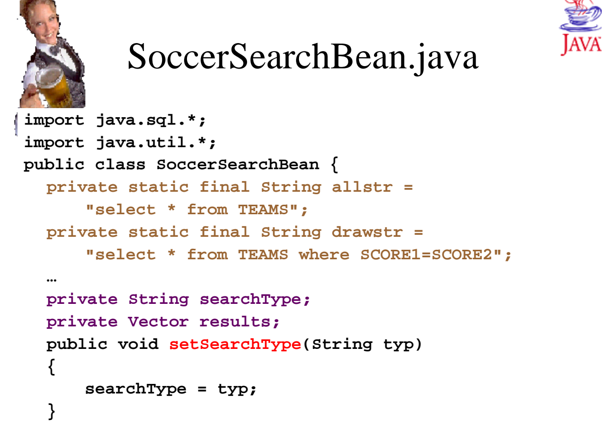
```
import java.sql.*;

public class SoccerGame {
  private String team1;
  private String team2;
  private int    score1;
  private int score2;

  public String getTeam1() { return team1; }
  public String getTeam2() { return team2; }
  public String getScore1() { return "" + score1; }
  public String getScore2() { return "" + score2; }

  …
```

SoccerGame.java

```
public class SoccerGame {
  …

  public void loadFromResultSet(ResultSet rset)
  throws SQLException
  {
     team1 = rset.getString("TEAM1");
     team2 = rset.getString("TEAM2");
     score1= rset.getInt("SCORE1");
     score2 = rset.getInt("SCORE2");
  }

}
```

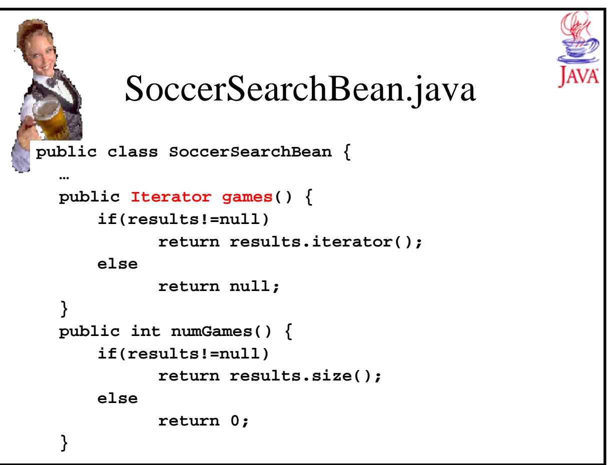
SoccerSearchBean.java

- Owns
  - String data member to hold type of next search
  - Vector to hold collection of retrieved SoccerGame objects
  - Some string constants for SQL queries
- Does
  - Allows setting of search type
  - Perform search, collecting results in memory
  - Reports on number of items found for search
  - Returns an iterator allowing access to retrieved items



SoccerSearchBean.java

```
import java.sql.*;
import java.util.*;
public class SoccerSearchBean {
  private static final String allstr =
      "select * from TEAMS";
  private static final String drawstr =
      "select * from TEAMS where SCORE1=SCORE2";
  …
  private String searchType;
  private Vector results;
  public void setSearchType(String typ)
  {
      searchType = typ;
  }
```



SoccerSearchBean.java

```
public class SoccerSearchBean {
  …
  public Iterator games() {
      if(results!=null)
          return results.iterator();
      else
          return null;
  }
  public int numGames() {
      if(results!=null)
          return results.size();
      else
          return 0;
  }
```

SoccerSearchBean.java

```
public void doSearch() {
   results = new Vector();
   try {
     Connection db = DBInfo.connectToDatabase();
     Statement stmt = db.createStatement();
     String request = allstr;
     if("drawn".equals(searchType))
       request = drawstr;
     else
       …;
     ResultSet rset = stmt.executeQuery(request);
     …
```

SoccerSearchBean.java

```
try {
  …
  ResultSet rset = stmt.executeQuery(request);
  while(rset.next()) {
      SoccerGame sg = new SoccerGame();
      sg.loadFromResultSet(rset);
      results.addElement(sg);
  }
  rset.close();
  stmt.close();
  db.close();
}
catch(Exception e) { … }
```

JSP page – “pretties”

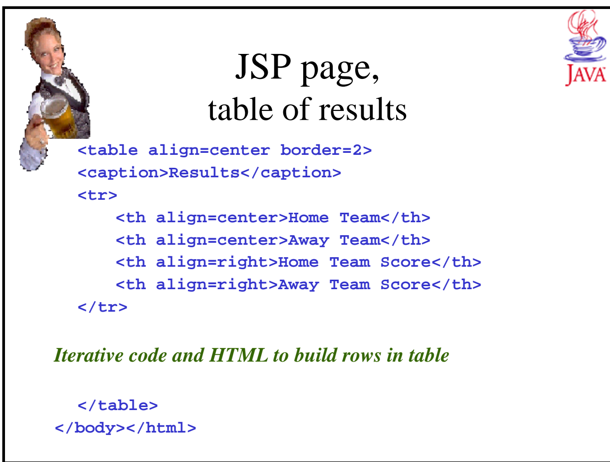
```
<%@ page import="java.util.*" %>
<html><head>
<title>Soccer League Results</title></head>
<body bgcolor=white>
<!--
  Imagine that this is page contains lots of HTML
  directives to build a really pretty page.  A page
  with a tiled picture background (soccer balls ad
  infinitum); assorted advertisements strategically
  placed.  All created by some creative artist
  utilizing an interactive editing program.
  Embedded in amongst that auto-generated HTML will be
  a few fragments of JSP scripting: actions,
  scriptlets, etc.
-->
<h1 align=center><font color=red>Testing soccer</font></h1>
```

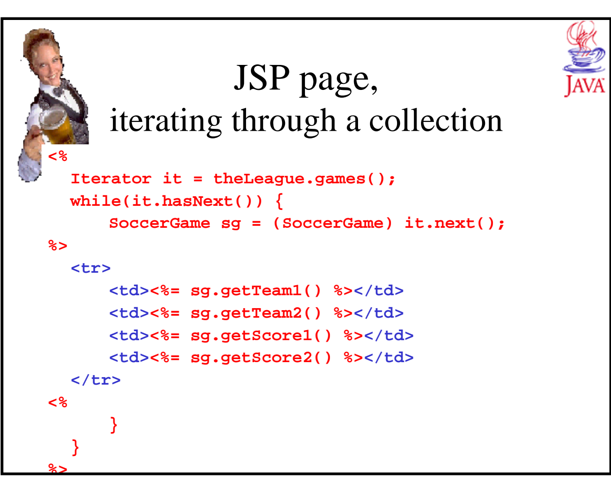
JSP page - work

- Actions:
  - Use a SoccerSearchBean
  - Set property for searchType
- Scriptlet
  - SoccerSearchBean runs search
  - Query count of results
    - If zero, select short HTML “no results” output
    - If non-zero, HTML to set up table then scriptlet code filling rows of table

JSP page, set up and select output

```
<p>
<jsp:useBean id="theLeague"
   class="SoccerSearchBean" />
<jsp:setProperty name="theLeague"
   property="searchType" />
<%
   theLeague.doSearch();
%>
<%if(theLeague.numGames()==0) { %>
   <p>There haven't been any such games yet. But
   the season is young; come back again soon.
<%
   }
   else {
%>
```





Soccer example

- Not too bad
- Code in JSP
  - Exploit jsp tagset to extract parameters and use them to set bean data members
  - Invoke action
  - Select one of two outputs formats
  - Iterating through collection
- Still could probably do better
  - Lots of scriptlet code, error opportunities – look at that double close “}” at end of last code fragment!

Improvements

- Servlet handles part of work?
  - Conceptually have two different dynamic response pages
    - One reporting a search failure
      - In more sophisticated example, would have some reason for failure of search
    - One reporting results of a successful search
      - Gets passed collection of data to display
- Overall system better if servlet handled the initial processing, created data and forwarded data to selected JSP

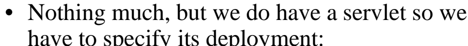
Soccer: Servlet, Beans, JSP pages

- Static HTML start page sends request to Servlet
- Servlet:
  - Creates a SoccerSearchBean
  - Gets search parameter, loads into bean
  - Gets bean to run the search and collect the results in a Vector of SoccerGame objects
  - Checks the result
    - If no matches found
      - Compose a suitable message, forward along with request & response to "failure JSP page"
    - If matches found
      - Forward the SoccerSearchBean along with request & response to reporter page

Components

- Soccer.html
  - Links now reference servlet URL as specified in web.xml deployment
- MatchReport.jsp
- NoResult.jsp
- web.xml
- PreprocessServlet.java
- SoccerGame.java, SoccerSearchBean.java and DBInfo.java
  - unchanged

web.xml



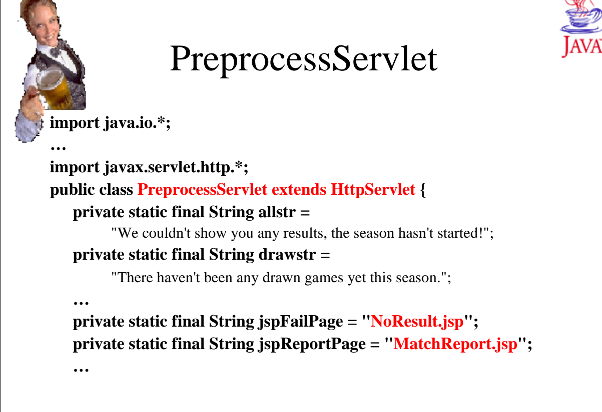
- Nothing much, but we do have a servlet so we have to specify its deployment:

```
<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
   2.2//EN"
   "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
   <servlet>
       <servlet-name>SoccerServlet</servlet-name>
       <servlet-class>PreprocessServlet</servlet-class>
   </servlet>
   <servlet-mapping>
       <servlet-name>SoccerServlet</servlet-name>
       <url-pattern>/SoccerInfo</url-pattern>
   </servlet-mapping>
</web-app>
```

PreprocessServlet

- doGet()
  - Pick up search type,
  - operations using SoccerSearchBean,
  - Decision on which JSP handles final output
- *Servlet could have additional role of owning a persistent database connection, avoiding costly opening of connection for each individual search.*

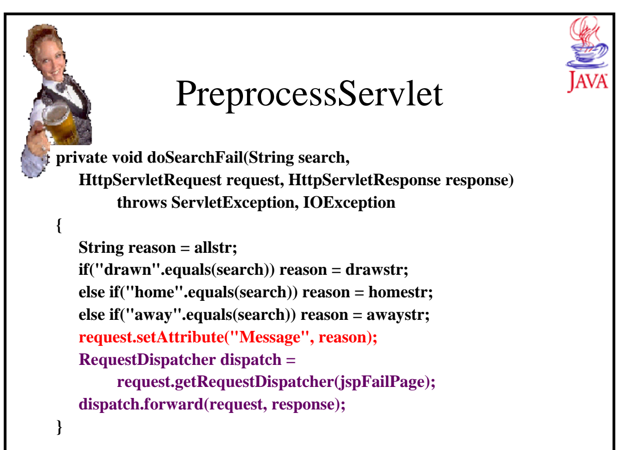
PreprocessServlet

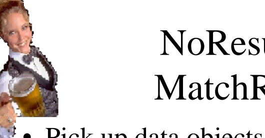
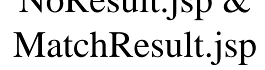


```
import java.io.*;
…
import javax.servlet.http.*;
public class PreprocessServlet extends HttpServlet {
    private static final String allstr =
        "We couldn't show you any results, the season hasn't started!";
    private static final String drawstr =
        "There haven't been any drawn games yet this season.";
    …
    private static final String jspFailPage = "NoResult.jsp";
    private static final String jspReportPage = "MatchReport.jsp";
    …
```

PreprocessServlet

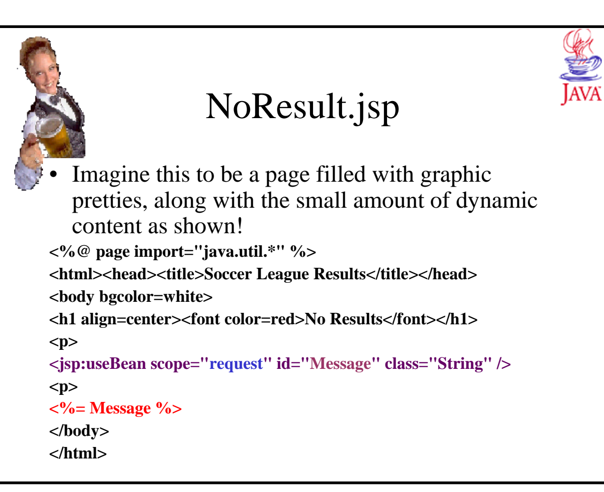
```
public void doGet (HttpServletRequest request,
    HttpServletResponse response)
        throws ServletException, IOException
{
    String search = request.getParameter("searchType");
    SoccerSearchBean ssb = new SoccerSearchBean();
    ssb.setSearchType(search);
    ssb.doSearch();
    if(ssb.numGames()==0)
        doSearchFail(search, request, response);
    else
        doSuccess(ssb, request, response);
}
```



NoResult.jsp & MatchResult.jsp

- Pick up data objects passed from servlet
  - Servlet attached them to request (as attributes) before forwarding the request and response
  - JSP can get at them by specifying a jsp:Usebean action involving a "request scope" data item.
- JSP page can then process final data



MatchReport.jsp

```
<%@ page import="java.util.*" %>
<html><head><title>Soccer League Results</title></head>
<body bgcolor=white>
<h1 align=center><font color=red>Search Results</font></h1>
<p>
<jsp:useBean scope="request" id="theLeague" class="SoccerSearchBean" />
<table align=center border=2>
<caption>Results</caption>
<tr>
    <th align=center>Home Team</th>
    <th align=center>Away Team</th>
    <th align=right>Home Team Score</th>
    <th align=right>Away Team Score</th>
</tr>
…
```

MatchReport.jsp

```
…
<%
    Iterator it = theLeague.games();
    while(it.hasNext()) {
        SoccerGame sg = (SoccerGame) it.next();
%>
    <tr>
        <td><%= sg.getTeam1() %></td>
        <td><%= sg.getTeam2() %></td>
        <td><%= sg.getScore1() %></td>
        <td><%= sg.getScore2() %></td>
    </tr>
<%
    }
%>
</table></body></html>
```

Better!

- JSP pages now focus solely on data presentation
  - No application logic
  - No complex code
- But still have scriptlet code for accessing data held in SoccerGame objects
- Plus the code for the iterative loop.
- What else can we do to improve?

Expressions v. getProperty

- Those <%= sg.getTeam1() %> expressions?
- What is wrong with a <jsp:getProperty … />?

<jsp:getProperty …

- <jsp:getProperty name="x" property="y" />
- Not translated simply into:
  - x.getY();
- It's a complex bit of code along lines of:
  - Look up something called "x" in page context and find out what it is?.
  - Use reflection to find whether it has a getY() function.
  - Build a method object that will call getY() on appropriate x-thing
  - Run method object.

<jsp:getProperty and script variables …

- For auto-coding system to work, the name "x" must be associated with the "x-object" in some table describing the page.
- Can't simply have

<jsp:getProperty name="sg" property="team1" />

Promoting variable to page scope

- But can "promote" a script variable so that it is known in page context,
  then can use <jsp: tags.
- Here is how:

```
…
SoccerGame sg = (SoccerGame) it.next();
pageContext.setAttribute("sg",sg,PageContext.PAGE_SCOPE);
%>
<tr><td>
    <jsp:getProperty name="sg" property="team1" />
</td>
```

And that is it …

- You can't further simplify the code.
- Still have to have that scriptlet code for the iterative construct:

```
<%
  Iterator it = theLeague.games();
  while(it.hasNext()) {
      SoccerGame sg = (SoccerGame) it.next();
      pageContext.setAttribute("sg",sg,PageContext.PAGE_SCOPE);
%>
<tr>
  …
  <jsp:getProperty name="sg" property="score2" />
  </td>
</tr>
<% } %>
```

And that is it, if you stick to solely the jsp actions

- Well, you can go further.
- But not if limited to the standard “jsp” action set.
- Which brings us to “custom actions” or “tag libraries”.

Tag libraries

Why?

Why? Well, would you believe …

```
<logic:iterate id="sg" collection="<%= theLeague.games() %>" >
<tr>
   <td><bean:write name="sg" property="team1" /></td>
   <td><bean:write name="sg" property="team2" /></td>
   <td><bean:write name="sg" property="score1" /></td>
   <td><bean:write name="sg" property="score2" /></td>
</tr>
 </logic:iterate>
```

Safer for use in page edited by creative designers!

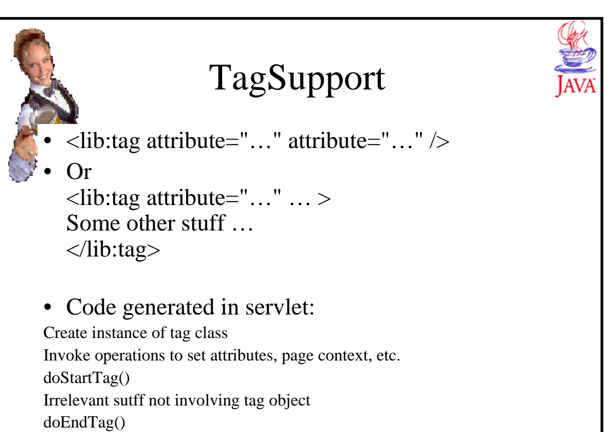
- Action tag style suits JSP
  - JSP "code" should be "get property and place here"
- Iterate tag far less likely to be messed up than scriptlet when page re-edited by creative designer
  - Clear, matching begin end tags
  - XML-tag style probably understood by editing environment

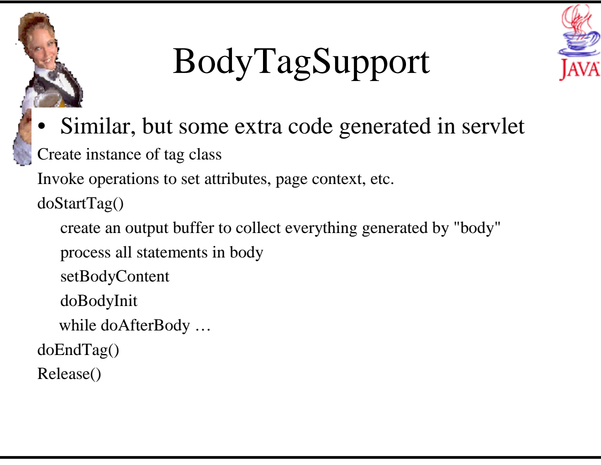
Iterate?

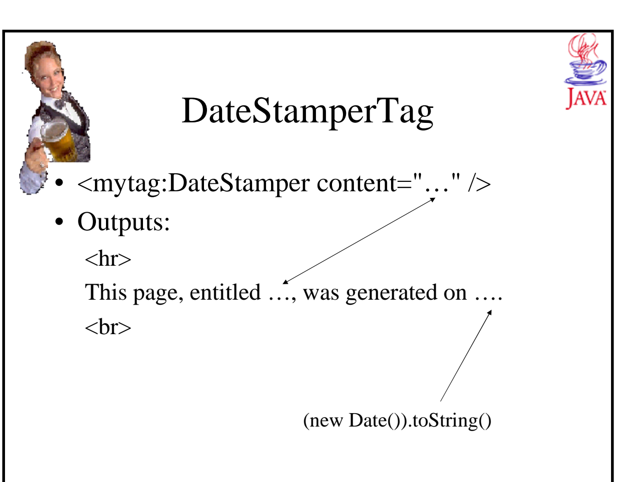
- Comes from Apache's "struts" taglib.
- Struts:
  - Beans:
    - Utility
  - Html
    - Tags associated with composition of HTML forms
  - Logic
    - Conditional test, iterative constructs etc
  - Template
    - Assist transfer of data amongst JSP pages by adding data to request

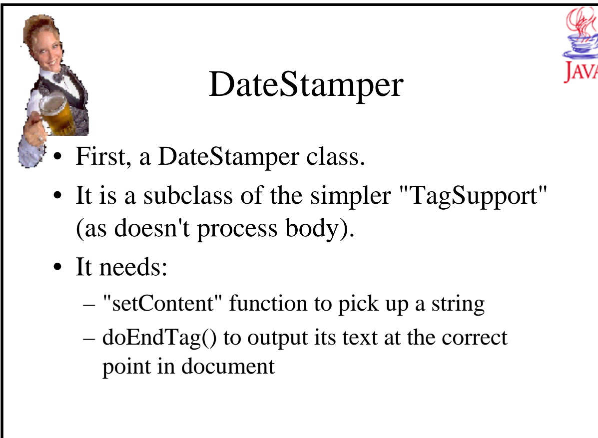
First though, our own tag

- Tags classes, for implementing action tags in JSP page, are created by extending library classes provided in the package javax.servlet.jsp.tagext
- TagSupport
- BodyTagSupport







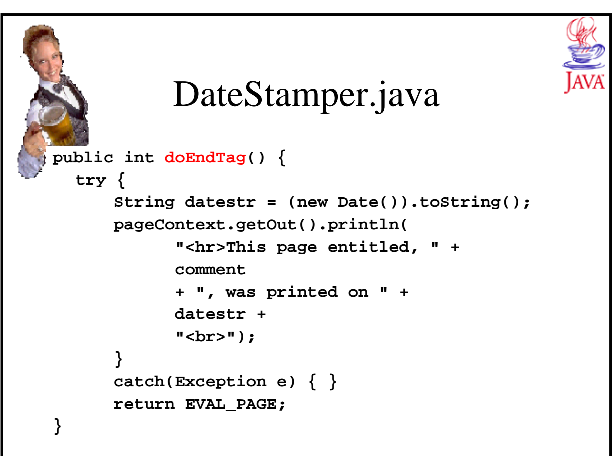


DateStamper.java

```
package mine;
import java.util.*;
import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class DateStamper extends TagSupport
{
    protected String comment = null;
    …
}
```

DateStamper.java

```
public class DateStamper extends TagSupport {
    …
    public String getComment() {
            return comment;
    }
    public void setComment(String cm) {
        comment  = cm;
    }
    public int doEndTag() {
        try { … }
        catch(Exception e) { }
        return EVAL_PAGE;
    }
}
```



DateStamper.java

```java
public int doEndTag() {
   try {
        String datestr = (new Date()).toString();
        pageContext.getOut().println(
              "<hr>This page entitled, " +
              comment
              + ", was printed on " +
              datestr +
              "<br>");
        }
        catch(Exception e) { }
        return EVAL_PAGE;
}
```

DateStamper.doEndTag()

- Gets a (buffered) output stream from the PageContext
  - PageContext maintains a stack of buffered output streams,
  - Output from an inner action tag is saved and can be made available to body processing routine of an enclosing tag

DateStamper.doEndTag()

- Returns
  - EVAL_PAGE
  - SKIP_PAGE

Deployment of tag …

- Like most advanced Java, deployment is harder than coding.
  - JSP page must specify that it wants to use tags from "mytag" library
  - web.xml must specify where an xml document describing library can be found
  - "tag library descriptor" file must contain specification of tags
  - Code for library must be in CLASSPATH when JSP compiled into servlet.

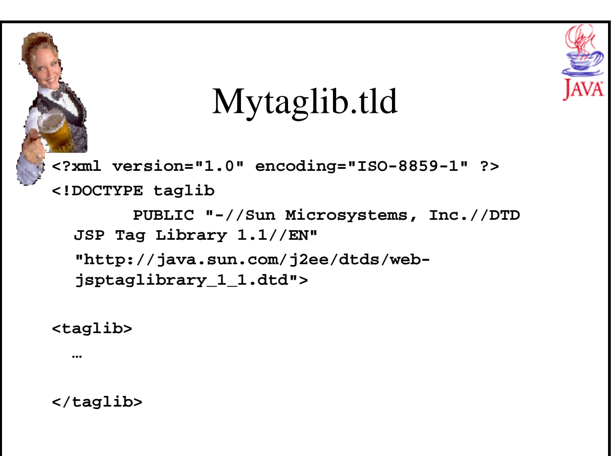
Deployment for DateStamper example

- JSP page

```
<%@ page session="false" %>
<%@ taglib uri="/mytaglib" prefix="mytag" %>
<html><head><title>My Tag Test</title></head>
<body bgcolor=white>
<h1 align=center>Test Document</h1>
<p>
This is a test
<mytag:DateStamper comment="Test Document" />
</body></html>
```

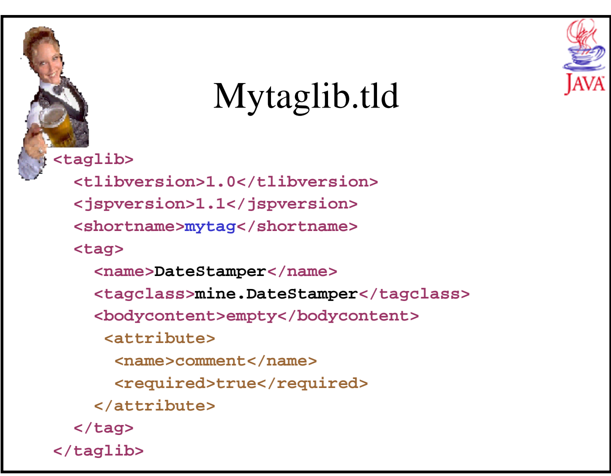
Web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC '-//Sun Microsystems,
  Inc.//DTD Web Application 2.2//EN'
  'http://java.sun.com/j2ee/dtds/web-app_2.2.dtd'>
<web-app>
  <taglib>
    <taglib-uri>
      /mytaglib
    </taglib-uri>
    <taglib-location>
      /WEB-INF/tlds/mytaglib.tld
    </taglib-location>
  </taglib>
</web-app>
```



Mytaglib.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
        PUBLIC "-//Sun Microsystems, Inc.//DTD
  JSP Tag Library 1.1//EN"
  "http://java.sun.com/j2ee/dtds/web-
  jsptaglibrary_1_1.dtd">

<taglib>
  …

</taglib>
```



Mytaglib.tld

```
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>mytag</shortname>
  <tag>
    <name>DateStamper</name>
    <tagclass>mine.DateStamper</tagclass>
    <bodycontent>empty</bodycontent>
     <attribute>
      <name>comment</name>
      <required>true</required>
    </attribute>
  </tag>
</taglib>
```

Code

- In WEB-INF/classes have subdirectory "mine" containing the source.
- Typically would have many tags in a library, would package them in a "jar" file and place this in WEB-INF/lib
- For simple example like this, sufficient to copy DateStamper.class into WEB-INF/classes directory

More likely to use existing tag libraries

- Struts beans:
  - Bean properties
    - Extends <jsp:getProperty, <jsp:setProperty with more elaborate variants
  - Bean creation
    - Extends <jsp:useBean
  - Bean output
    - Output of text from "bean.toString()" or "bean.getProperty()"
    - Optional of "internationalization", generation of locale customized outputs

Lots of other classes in org.apache.struts.taglib.bean

- CookieTag
- HeaderTag
- ParameterTag
- ResourceTag
- These primarily "helper" objects used when creating beans,
  used to get initialization data from cookie, form parameters, headers, …

More struts, "html"

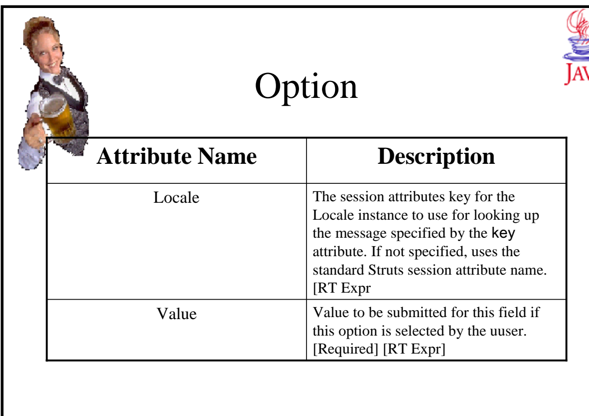
- Large set of tags relating to construction of HTML forms.
- Not clear when these would be preferable to visual editor style insertion of form elements,
  possibly advantageous if have JSP pages being generated programmatically

HTML tags

- Button   renders button
- Cancel   renders cancel button
- Checkbox   renders checkbox
- …
- Textarea   renders textarea
- Each tag takes a host of required and optional arguments …

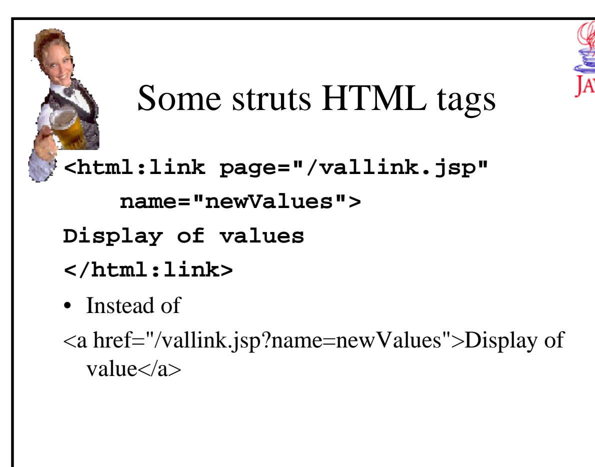
HTML Option (for Select)

Attribute Name	Description
bundle	The servlet context attributes key for the MessageResources instance to use. If not specified, defaults to the application resources configured for our action servlet. [RT Expr]
Disabled	Set to true if this option should be disabled. [RT Expr]
Key	If specified, defines the message key to be looked up in the resource bundle specified by bundle for the text displayed to the user for this option. If not specified, the text to be displayed is taken from the body content of this tag. [RT Expr]



Option

Attribute Name	Description
Locale	The session attributes key for the Locale instance to use for looking up the message specified by the key attribute. If not specified, uses the standard Struts session attribute name. [RT Expr
Value	Value to be submitted for this field if this option is selected by the uuser. [Required] [RT Expr]

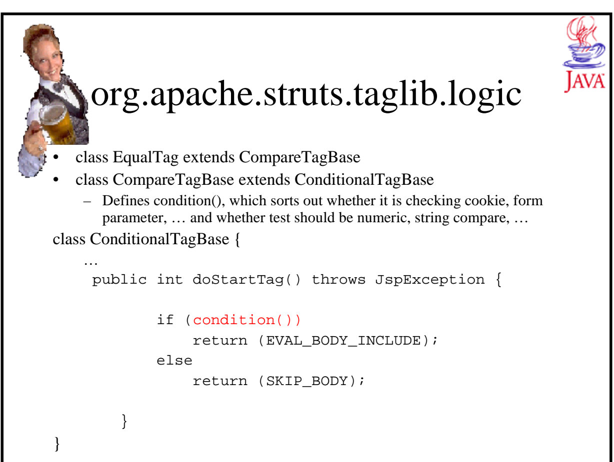


Logic tags

- These are likely to be most useful of the struts collection
- Logic
  - Value comparison
  - Substrings
  - Forward and redirect
  - Iterate

Equal tag …

- Attributes (specify in tag line)
  - "value" for test
  - Identification of thing to be tested …
    - Cookie
    - Parameter
    - Property of Bean
    - …
- Code for doStartTag checks condition and returns "SKIP_BODY" or "EVAL_BODY_INCLUDE"



org.apache.struts.taglib.logic

- class EqualTag extends CompareTagBase
- class CompareTagBase extends ConditionalTagBase
  - Defines condition(), which sorts out whether it is checking cookie, form parameter, … and whether test should be numeric, string compare, …

```
class ConditionalTagBase {
    …
    public int doStartTag() throws JspException {

            if (condition())
                return (EVAL_BODY_INCLUDE);
            else
                return (SKIP_BODY);

        }
}
```

Iterate

- Iterate is more complex:
  - Needs to process body
  - Needs to introduce "loop variable" into scope of JSP page
- Definition involves extra helper class – Tag Extra Information class
  - This supplies data describing "loop variable"

Iterate

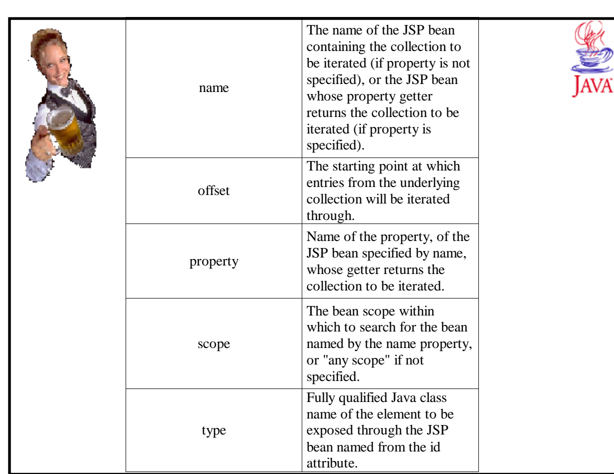
- Repeats the nested body content for every element of a collection
- Collection must be specified as:
  - runtime expression ("collection" attribute)
  - JSP bean ("name" attribute)
  - Property of a bean ("name" and "property" attributes)

Iterate

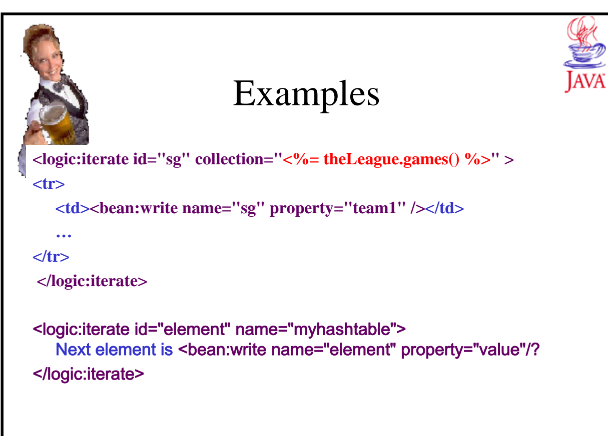
- Collection must be one of
  - An array of Java objects
  - A java.util.Collection, e.g. Vector
  - A java.util.Enumeration.
  - A java.util.Iterator.
  - A java.util.Map.

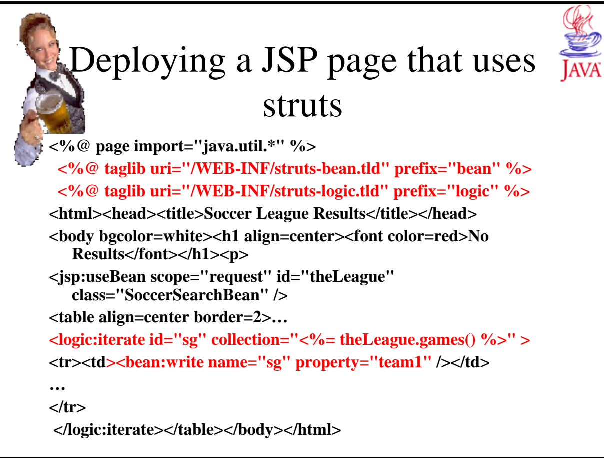
Iterate

Attribute Name	Description
collection	A runtime expression that evaluates to a collection
id	The name of a page scope JSP bean that will contain the current element of the collection.
indexId	The name of a page scope JSP bean that will contain the current index .
length	The maximum number of entries to be iterated through on this page (an integer, or the name of a JSP bean of type `java.lang.Integer`).



name	The name of the JSP bean containing the collection to be iterated (if property is not specified), or the JSP bean whose property getter returns the collection to be iterated (if property is specified).
offset	The starting point at which entries from the underlying collection will be iterated through.
property	Name of the property, of the JSP bean specified by name, whose getter returns the collection to be iterated.
scope	The bean scope within which to search for the bean named by the name property, or "any scope" if not specified.
type	Fully qualified Java class name of the element to be exposed through the JSP bean named from the id attribute.





Example

- Used struts-bean
  - <bean:write
- Used struts-logic
  - <logic:iterate

Struts components

- WEB-INF contains
  - struts-bean.tld
  - struts-logic.tld

<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>

- WEB-INF/lib contains
  - Struts.jar
- All struts components from archive downloaded from http://www.apache.org

Iterate – the tag implementation

```
public class IterateTag extends BodyTagSupport {
   protected Iterator iterator = null;
   …
   protected Object collection = null;
   public Object getCollection() { return
   collection; }
   public void setCollection(Object collection) {
       this.collection = collection;
    }
   public String getId() { return id; }
   public void setId(String id) { this.id = id; }
```

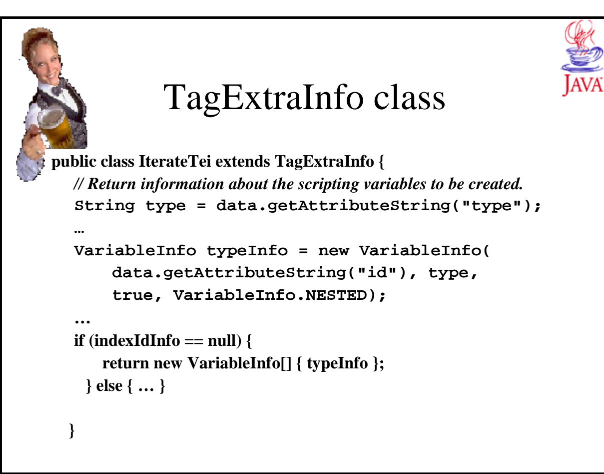
Iterate – the tag implementation

```
public int doStartTag() throws JspException {
   Object collection = this.collection;
   if (collection == null)
       collection =
           RequestUtils.lookup(pageContext, name,
             property, scope);
   …
   // Construct an iterator for this collection
   if (collection.getClass().isArray())
       collection =
           Arrays.asList((Object[]) collection);
   if (collection instanceof Collection)
       iterator = ((Collection)
           collection).iterator();
```

```
public int doStartTag() throws JspException {
    …
    // Calculate the starting offset
    …
    // Calculate the rendering length
    …
    // Skip to the starting offset
    …
    // Store the first value and evaluate,
    if (iterator.hasNext()) {
        Object element = iterator.next();
        …
        return (EVAL_BODY_TAG);
     }
    //or skip the body if none
    else return (SKIP_BODY);
}
```

doAfterBody

```
public int doAfterBody() throws JspException {
    // Render the output from this iteration
    // to the output stream
    if (bodyContent != null) {
        ResponseUtils.writePrevious(pageContext,
                 bodyContent.getString());
        bodyContent.clearBody();
    }
    // Decide whether to iterate or quit
    …
    if (iterator.hasNext()) {
        …
        return (EVAL_BODY_TAG);
    } else return (SKIP_BODY);
}
```



TagExtraInfo class

```
public class IterateTei extends TagExtraInfo {
    // Return information about the scripting variables to be created.
    String type = data.getAttributeString("type");
    …
    VariableInfo typeInfo = new VariableInfo(
        data.getAttributeString("id"), type,
        true, VariableInfo.NESTED);
    …
    if (indexIdInfo == null) {
        return new VariableInfo[] { typeInfo };
     } else { … }

}
```

Odds and ends

Some newer JSP aspects

JSPs – newer stuff

- Standardization of tag libraries
- “Expression language”
- “JSP documents”
- Alternative ways of defining custom tags
- …

JavaServer Pages Standard Tag Library (JSTL)

- Idea of just a few standard tags (jsp:…) and custom tag sets really did not work out well.
- Everyone needed
  - Iterator
  - Conditional
- Lots of people chose to have
  - SQL tags

http://www.jadecove.com/downloads/jstl-quick-reference.pdf

JSTL

- Numerous "in-house" tag libraries developed
- Competing third party products

- Problems:
  - Web site might end up using several different libraries
  - Maintenance problems
  - Incompatibilities

- Sun introduces "standard" tag library (building in part from some Apache work)

JSTL

- Classes implementing JSTL tags may be part of the servlet container's library – no need for extra ".jar" files in application.
  If needed (as with our Glassfish Appserver) add JSTL libraries to project
- Refer to
  http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html
- for a complete list of the JSTL tags and their attributes.

JSTL

- Core:
  - Variable definition and manipulation
  - Flow control
  - URL related
- SQL
  - Connections and queries
- Internationalization
  - Number and date formatting, message strings
- "XML"

Area	Function	Tags	Prefix
Core	Variable support	remove set	c
	Flow control	choose when otherwise forEach forTokens if	
	URL management	import param redirect param url param	
	Miscellaneous	catch out	

JSTL core

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
…
<c:set var="foo" scope="session" value="..."/>
…
<c:forEach var="Item" begin="0" items="…">… </c:forEach>
…
<c:if test="${!empty param.CustomerName}"> … </c:if>
```

JSTL

- Intent is that "standard" tags will replace use of in-house and 3rd-party tag libraries for routine operations.
- SQL tags – intended to quick prototyping, advice is not to use them in real applications

```
<sql:query var="books" dataSource="${applicationScope.bookDS}">
    select * from PUBLIC.books where id = ?
    <sql:param value="${bid}" />
</sql:query>
```

Expression language

- Previously could have fragments of scriptlet code used in tags – e.g. code to access data member of some “bean” attached to request, or code to get an iterator
- **<logic:iterate id="sg" collection="<%= theLeague.games() %>" >**
- Expression language – alternative, supposedly simpler scripting language that can be used for such purposes (claim is EL is easier for “web designers” to use than scriptlet code)

```
<c:if test="${sessionScope.cart.numberOfItems > 0}">
...
</c:if>
```

Expression Language

- Supposedly identical to Javascript on those parts that overlap.

EL

- ${ *EL code* }
- Code accessing data members of objects, performing relational tests etc; many implicit objects
  - servletContext, session, request, response, param, …
- One common use – accessing parameters with form data

EL - examples

<c:set var="bid" value="${param.bookId}"/>

- Picks up form input field bookId and places value in “bid”

```
<sql:query var="books“ dataSource="${applicationScope.bookDS}">
    select * from PUBLIC.books where id = ?
    <sql:param value="${bid}" />
</sql:query>
<c:forEach var="bookRow" begin="0“ items="${books.rowsByIndex}">
…
```

- Running SQL query then processing results within JSP page

Soccer – tag-style

- Add the JSTL library to library for project

```
<html><head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Soccer League Results</title>
  </head>
  <body bgcolor="white">
    <h <h1 align="center"><font color="red">Search Results</font></h1>
    <jsp:useBean id="theLeague" scope="request" class="soccer.SoccerSearchBean" />
    <table border="1" align="center">
        <tr>
          <th>Home Team</th><th>Away Team</th>
          <th>Home Score</th><th>Away Score</th>
        </tr>
        <c:forEach var="sg" items="${requestScope.theLeague.games}">
          <tr>
            <td><c:out value="${sg.team1}"/></td>
            <td><c:out value="${sg.team2}"/></td>
            <td><c:out value="${sg.score1}"/></td>
            <td><c:out value="${sg.score2}"/></td>
          </tr>
        </c:forEach>
    </table>
  </body>
</html>
```

Change definition of SoccerSearchBean class

- Will need to iterate through collection of results using JSTL iterator constructs
  these work best if given a java.util.Collection object; so add a method that returns the collection from the SoccerSearchBean

```
package soccer;
import java.sql.*;
import java.util.*;
public class SoccerSearchBean {
   …
   private Vector results;
   public Collection getCollection()
       { return results; }
```

The two JSP pages

- NoResults.jsp
  - Got passed a String as a request attribute named Message
  - Printed this string as part of a response
- MatchReport.jsp
  - Got passed the “SoccerSearchBean” as a request attribute named theLeague
  - Has to get collection of results from this bean and print in a table generating loop

Web-xml

- Will need extra taglib entries that specify the location of the tld files;

NoResults.jsp

```
<%@ taglib uri="http://java.sun.com/jstl/core"
   prefix="c" %>
<html><head><title>Soccer League Resutls</title></head>
<body bgcolor="white">
<h1 align="center"><font color="red">No Results</font></h1>
<p><c:out value="${requestScope.Message}" />
</body>
</html>
```

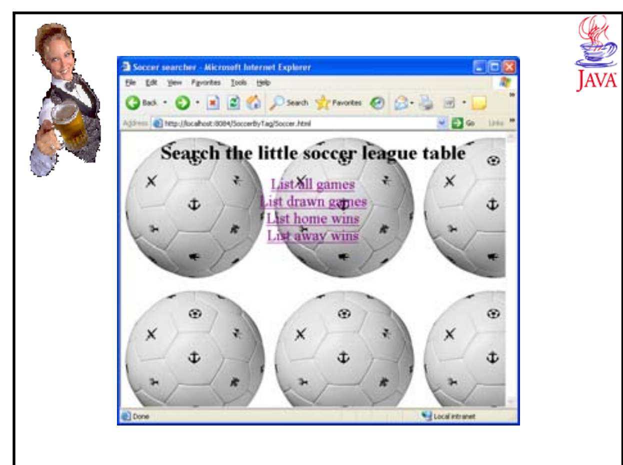
- Tag library identified.
- <c:out – print a value
- EL ${…}
  - requestScope
  - Attribute name

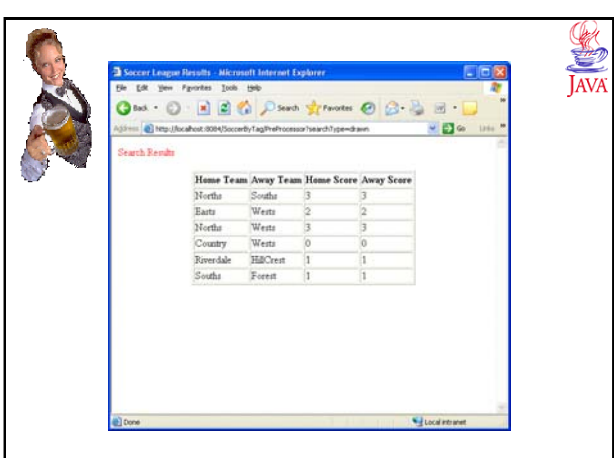
MatchReport.jsp

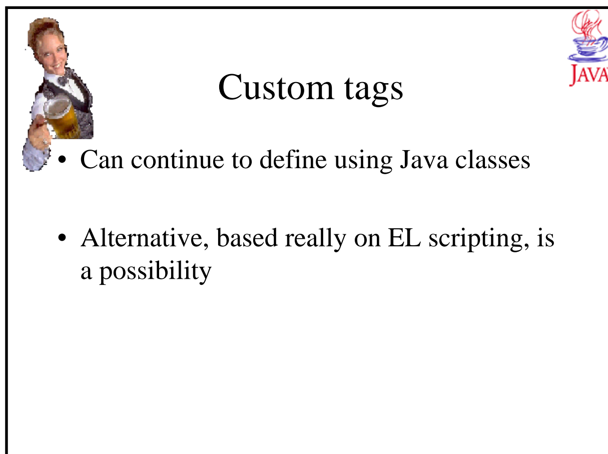
```
<%@ taglib uri="http://java.sun.com/jstl/core"
   prefix="c" %>
<html><head><title>Soccer League Results</title></head>
<body bgcolor=white>
<h1 align=center><font color=red>Search Results</font></h1>
<p>
<table align=center border=2>
<caption>Results</caption>
<tr>
<th align=center>Home Team</th>
<th align=center>Away Team</th>
<th align=right>Home Team Score</th>
<th align=right>Away Team Score</th>
</tr>
…
```

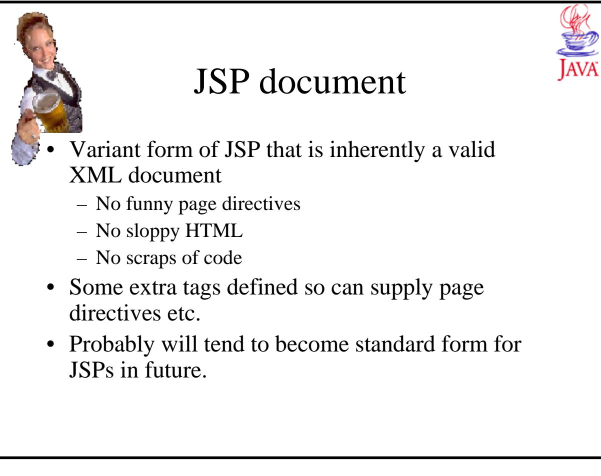
MatchReport.jsp

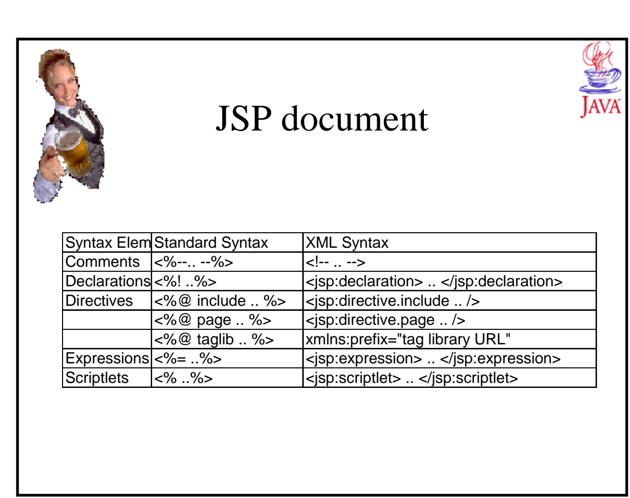
```
<%@ taglib uri="http://java.sun.com/jstl/core"
   prefix="c" %>
…
<c:forEach var="soccergame"
   items="${requestScope.theLeague.collection}">
<tr>
<td><c:out value="${soccergame.team1}" /></td>
<td><c:out value="${soccergame.team2}" /></td>
<td><c:out value="${soccergame.score1}" /></td>
<td><c:out value="${soccergame.score2}" /></td>
</tr>
</c:forEach>
</table></body></html>
```



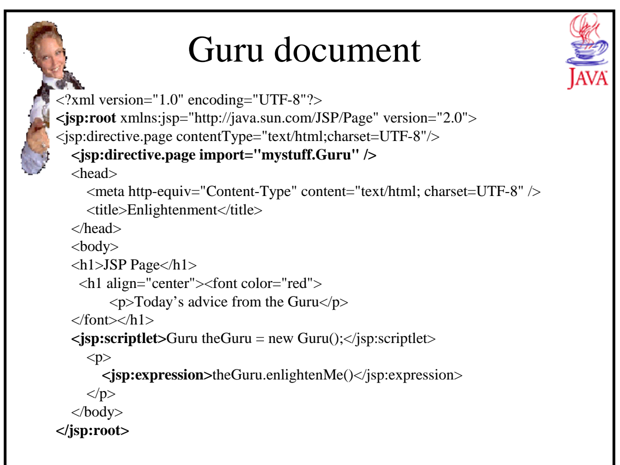


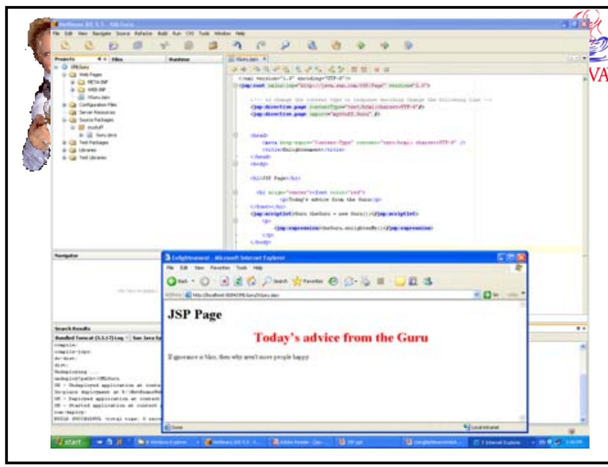




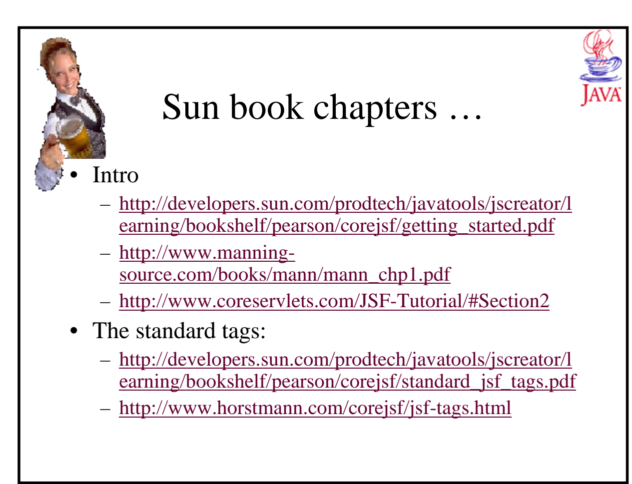


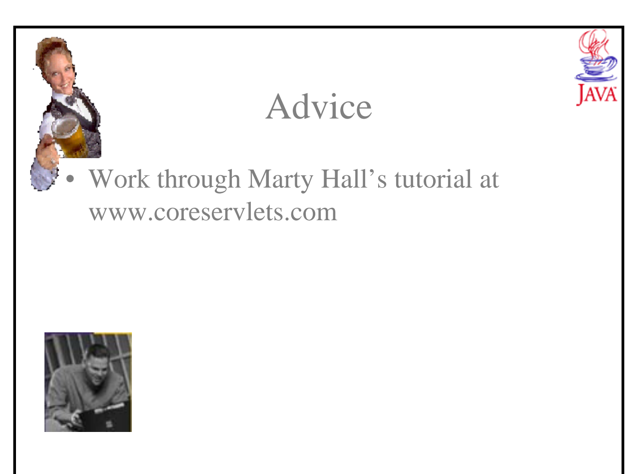
Syntax Elem	Standard Syntax	XML Syntax
Comments	<%--.. --%>	<!-- .. -->
Declarations	<%! ..%>	<jsp:declaration> .. </jsp:declaration>
Directives	<%@ include .. %>	<jsp:directive.include .. />
	<%@ page .. %>	<jsp:directive.page .. />
	<%@ taglib .. %>	xmlns:prefix="tag library URL"
Expressions	<%= ..%>	<jsp:expression> .. </jsp:expression>
Scriptlets	<% ..%>	<jsp:scriptlet> .. </jsp:scriptlet>

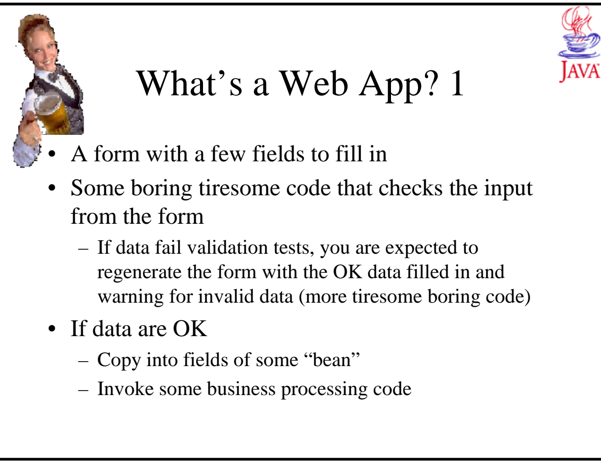

Guru document
<?xml version="1.0" encoding="UTF-8"?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0">
<jsp:directive.page contentType="text/html;charset=UTF-8"/>
<jsp:directive.page import="mystuff.Guru" />
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Enlightenment</title>
</head>
<body>
<h1>JSP Page</h1>
<h1 align="center">
<p>Today’s advice from the Guru</p>
</h1>
<jsp:scriptlet>Guru theGuru = new Guru();</jsp:scriptlet>
<p>
<jsp:expression>theGuru.enlightenMe()</jsp:expression>
</p>
</body>
</jsp:root>
JAVA


JSP Page
Today’s advice from the Guru

Java Server Faces
Look Ma! No hands.
No servlet either, in fact not much code at all
JAVA


Sun book chapters …
• Intro
– http://developers.sun.com/prodtech/javatools/jscreator/learning/bookshelf/pearson/corejsf/getting_started.pdf
– http://www.manning-source.com/books/mann/mann_chp1.pdf
– http://www.coreservlets.com/JSF-Tutorial/#Section2
• The standard tags:
– http://developers.sun.com/prodtech/javatools/jscreator/learning/bookshelf/pearson/corejsf/standard_jsf_tags.pdf
– http://www.horstmann.com/corejsf/jsf-tags.html
JAVA


Advice
• Work through Marty Hall’s tutorial at www.coreservlets.com
JAVA


What’s a Web App? 1
• A form with a few fields to fill in
• Some boring tiresome code that checks the input from the form
– If data fail validation tests, you are expected to regenerate the form with the OK data filled in and warning for invalid data (more tiresome boring code)
• If data are OK
– Copy into fields of some “bean”
– Invoke some business processing code
JAVA

What's a Web App? 2

- Then you check the results of the business code
  - Different results => forwarding to different response page generators (so need some navigation rules built into your code)
- Finally you layout response pages including some generated response data.

Can't we automate the coding more?

- Servlets –
  - No automation of coding, write it all yourself.
- JSPs
  - Some automation
    - jsp:setproperty property="*" name=…
    - JSTL tags reduce detail of coding for things like output of a table of response data c:forEach etc
- We want more!

More!

- Declarative style not coding style
  - Map data fields to bean fields
  - Map requests to bean operations
- Declare the navigation rules
  - *"If business logic says "success" – go to SuccessReply.jsp,*
    but if business logic says "fail" go to … "
- And make allowance for more interactivity as well!

You want more?
You got JSF.

- Compose a form page with HTML and special JSF tags
  - Input field
    - Data must be supplied
    - String length at least 6
    - Corresponds to "xxx" field in bean record structure (and hence in data base)
  - Action button
    - Invoke specified processing function in bean – function returns string value indicating outcome
- Implement the bean
- Fill in an XML tag defining "navigation rules"
  - if bean's function returns "---" goto page1.jsp elsif bean's function returns ":::" goto page2.jsp etc

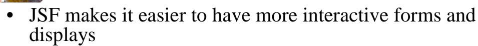
Increasing move to declarative style

- You don't code validations
- You declare
  - I want a validity check done on length of string
  - I want a validity check done on numeric value
  - …
- You don't code navigation
  - This outcome goto this page
  - …

Point & click

- Depends on your GUI based development tool
  - Generally, not as sophisticated as Microsoft Visual Studio
- Pick a "tag" from palette, drop onto form, fill in dialog or complete auto-generated text template
- *Who needs programmers? Business studies graduates can code the forms. Yuk.*

More interactive



- JSF makes it easier to have more interactive forms and displays
  - You
    - Tag a button with a “listener” attribute that identifies function in “backing bean”
    - Mark button as “immediate”
  - Generated code
    - Includes some Javascript that causes a postback of incomplete form and a request that function be run
- Can be used to make options change, e.g.
  - Select country from list (with autopostback)
  - Form “immediately” changes to display states in that country (so instead of Alaska, Arizona, you get NSW, SA, …)

JSF Examples

- Soccer (again)
- Membership (again)

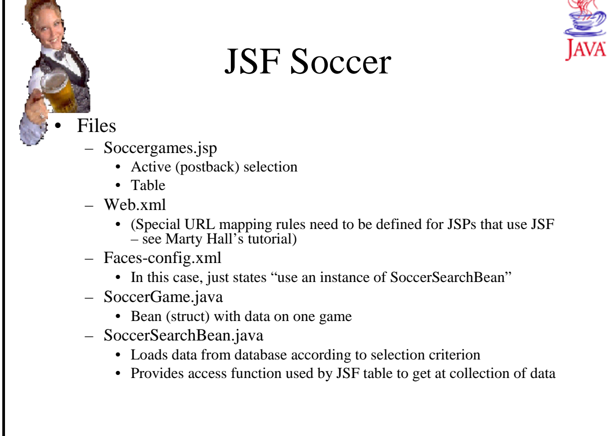
- *(Both built using JSF from within NetBeans – this stuff is too tiresome without a sophisticated IDE)*

JSF Soccer

JSF Soccer

- Selection field with auto-postback to allow user to pick games of interest
- Table that displays collection of rows

- Also use of css style sheet
  - *(I don’t usually bother with such pretties, but if one isn’t writing any code one does have more time for the aesthetics)*



JSF Soccer

- Files
  - Soccergames.jsp
    - Active (postback) selection
    - Table
  - Web.xml
    - (Special URL mapping rules need to be defined for JSPs that use JSF – see Marty Hall’s tutorial)
  - Faces-config.xml
    - In this case, just states “use an instance of SoccerSearchBean”
  - SoccerGame.java
    - Bean (struct) with data on one game
  - SoccerSearchBean.java
    - Loads data from database according to selection criterion
    - Provides access function used by JSF table to get at collection of data

JSF Soccer : JSP page

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Soccer Game 1</TITLE>
<LINK REL="STYLESHEET"
      HREF="./styles/styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">Soccer games</TH></TR>
</TABLE>
<h:form>
    <p class="HEADING">Select games for display</p>
    <div align="center">
        <h:selectOneListbox>...</h:selectOneListbox>
    </div>
</h:form>
<br/>
        <div align="center">
            <h:dataTable>...</h:dataTable>

        </div>
</BODY></HTML>
</f:view>
```

Includes for JSF tag libraries

Link to css styles

HTML markup

“Form” with active selection – see later

Table mapped to backing bean

JSF Soccer : JSP page

```
<h:form>
   <p class="HEADING">Select games for display</p>
   <div align="center">
      <h:selectOneListbox  styleClass="ROW1" immediate="true“
           onclick="submit()" valueChangeListener=
                    "#{soccerSearchBean.changeGameSelection}">
         <f:selectItem itemValue="all" itemLabel="All results" />
         <f:selectItem itemValue="drawn" itemLabel="Drawn games" />
         <f:selectItem itemValue="away" itemLabel="Away wins" />
         <f:selectItem itemValue="home" itemLabel="Home wins" />
      </h:selectOneListbox>
   </div>
</h:form>
```

JSF Soccer : JSP page

```
<h:dataTable   value="#{soccerSearchBean.results} "
     var="game" border="1"  rowClasses="ROW1,ROW2">
   <h:column>
        <f:facet name="header">
             <f:verbatim>Team1</f:verbatim>
        </f:facet>
           <h:outputText value="#{game.team1}"/>
    </h:column>
    <h:column>
         <f:facet name="header">
              <f:verbatim>Score1</f:verbatim>
         </f:facet>
         <h:outputText value="#{game.score1}"/>
     </h:column>
      …
</h:dataTable>
```

JSF : SoccerGame.java

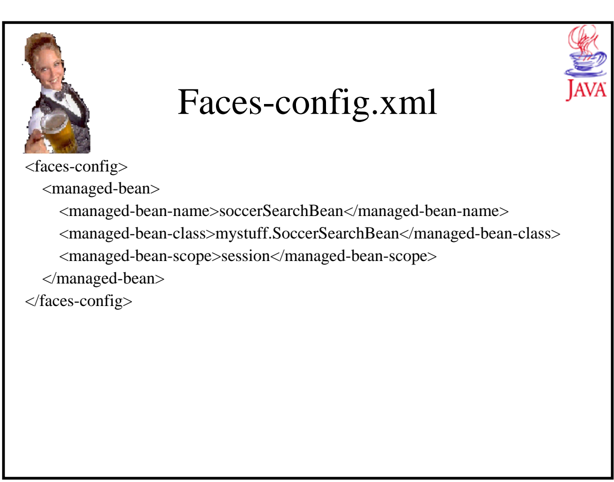
```
public class SoccerGame {
   private int score1;
   private int score2;
   private String team1;
   private String team2;
   public SoccerGame(String ateam1, int ascore1, String ateam2, int ascore2)
   {
      team1 = ateam1; team2 = ateam2; score1 = ascore1; score2 = ascore2;
   }
   public void setScore1(int newscore) { score1 = newscore; }
   public void setScore2(int newscore) { score2 = newscore; }
   …
   public int getScore1() { return score1; }
   public int getScore2() { return score2; }
   …
}
```

JSF : SoccerSearchBean.java

```
public class SoccerSearchBean {
   private static final String allstr =
        "select * from TEAMS";
   …
   private Vector results;
   /** Creates a new instance of SoccerSearchBean */
   public SoccerSearchBean() {
      doSearch("all");
   }

   public void changeGameSelection(ValueChangeEvent event)
   {
      String selection = (String) event.getNewValue();
      doSearch(selection);
   }
```

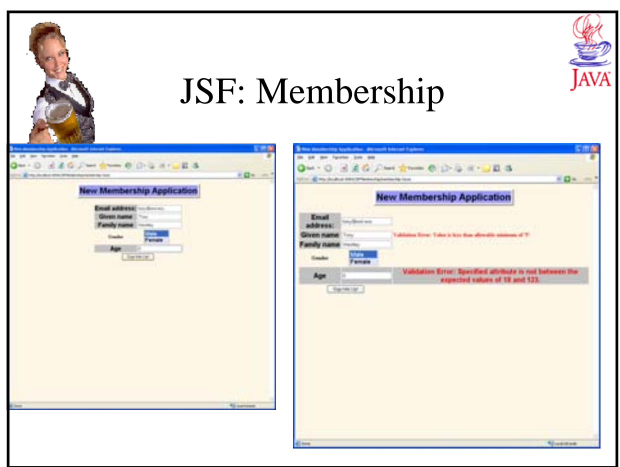
JSF : SoccerSearchBean.java

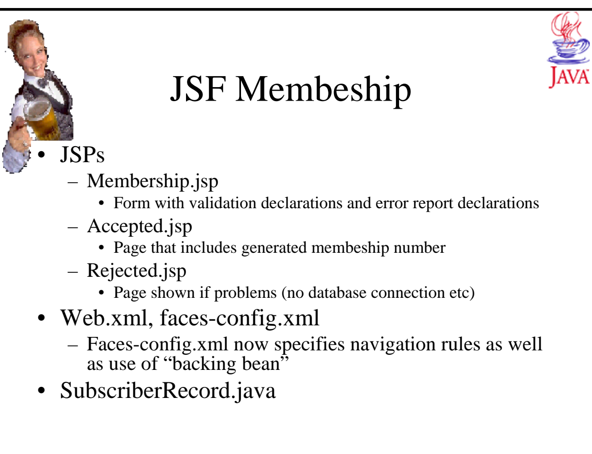
```
   public void doSearch(String searchType) {
      results = new Vector();
      try {
         …
      }
   }
   public Vector getResults() { return results; }
}
```



Faces-config.xml

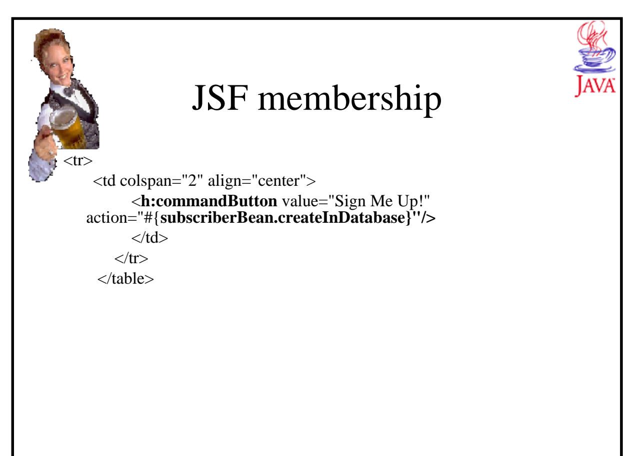
```
<faces-config>
   <managed-bean>
      <managed-bean-name>soccerSearchBean</managed-bean-name>
      <managed-bean-class>mystuff.SoccerSearchBean</managed-bean-class>
      <managed-bean-scope>session</managed-bean-scope>
   </managed-bean>
</faces-config>
```


JSF: Membership
JAVA
New Membership Application
New Membership Application


JSF Membeship
JAVA
• JSPs
– Membership.jsp
• Form with validation declarations and error report declarations
– Accepted.jsp
• Page that includes generated membeship number
– Rejected.jsp
• Page shown if problems (no database connection etc)
• Web.xml, faces-config.xml
– Faces-config.xml now specifies navigation rules as well as use of "backing bean"
• SubscriberRecord.java

JSF membership
JAVA
<h:form>
<TABLE>
<TR><TH CLASS="HEADING">
Email address: </TH>
<td>
<h:inputText id="email" required="true"
value="#{subscriberBean.email}">
<f:validateLength minimum="5" maximum="20"/>
</h:inputText>
</TD>
<TD>
<h:message for="email" styleClass="RED"/>
</TD>
</TR>

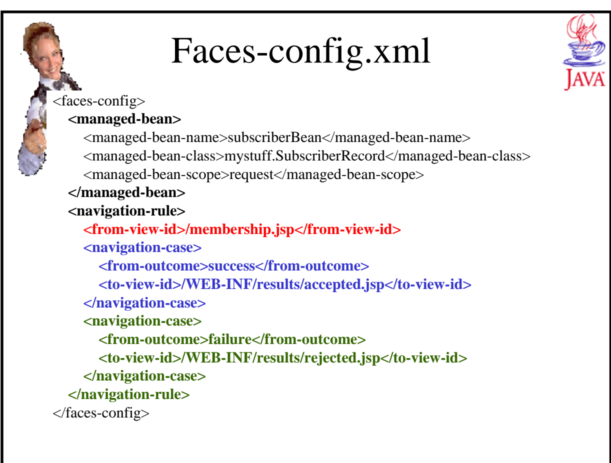
JSF membership
JAVA
<th>Gender</th>
<td>
<h:selectOneListbox required="true" styleClass="ROW1" >
<f:selectItem itemValue="Male" itemLabel="Male" />
<f:selectItem itemValue="Female" itemLabel="Female" />
</h:selectOneListbox>
</td>
</tr>
<tr CLASS="HEADING">
<th>Age</th>
<td>
<h:inputText id="age" value="#{subscriberBean.age}">
<f:validateLongRange maximum="123" minimum="18" />
</h:inputText>
</td>
<TD>
<h:message for="age" styleClass="RED"/>
</TD>
</tr>


JSF membership
JAVA
<tr>
<td colspan="2" align="center">
<h:commandButton value="Sign Me Up!"
action="#{subscriberBean.createInDatabase}"/>
</td>
</tr>
</table>

SubscriberRecord.java
JAVA
public class SubscriberRecord {
private String givenName;
…
private int id;
public String getGivenName() { return givenName; }
public void setGivenName(String aName) { givenName = aName; }
…
public int getAge() { return age;}
public void setAge(int aval) { age = aval; }
public int getId() { return id; }

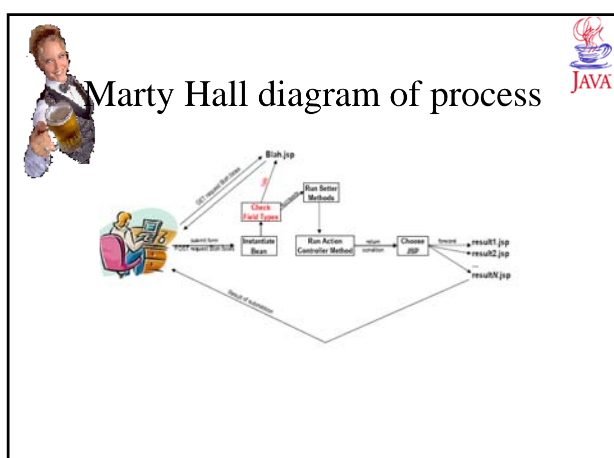
SubscriberRecord.java

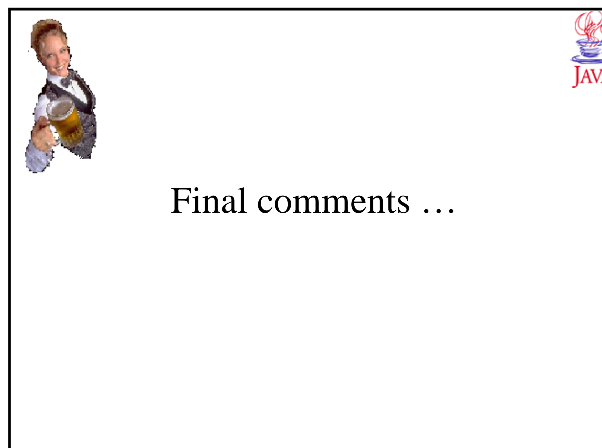
```
public String createInDatabase() {
    try {
        Connection db = null;
        …
        Statement stmt = db.createStatement();
    …
    id = value;
    } catch (Exception e) {
        System.out.println("Exception " + e);
        return "failure";
    }

    return "success";
  }
}
```



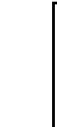
Faces-config.xml

```
<faces-config>
  <managed-bean>
    <managed-bean-name>subscriberBean</managed-bean-name>
    <managed-bean-class>mystuff.SubscriberRecord</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
  <navigation-rule>
    <from-view-id>/membership.jsp</from-view-id>
    <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/WEB-INF/results/accepted.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>failure</from-outcome>
      <to-view-id>/WEB-INF/results/rejected.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>
```





Model-2 JSP or MVC



- Servlet (Control)
  - Handles initial request
  - Creates beans with application data
  - Selects next processing step (another servlet or a JSP page)
- Beans (Model)
  - Hold application data
- JSPs (View)
  - Create HTML page with embedded data extracted from beans

Bean styles

- Obviously lots of variations, but some typical ones:
  - Package of data
    - Little more than a class with a set of public data elements
  - Bean-that-talks-to-database
    - Given its id, fills its data members from relational table(s)
    - Returns packages with selections of its data
    - Accepts update requests, modifies own data and does updates on DB table(s)
  - Organization beans
    - Program logic, creates and operates on "beans-that-talk-to database", prepares data packages, returns these to servlet for it to place in request/session/application attributes

A new division of labor …

- What do you need to make a good web-app?
  - **Creative graphic designer**
    - uses some program package to build pretty pages that will display dynamic content – may have sufficient computer training to insert xml-style tags – *get value, put it here*
  - **Deployment expert**
    - Knows how to arrange the libraries, write the web.xml and ".tld" deployment files
    - Defines security constraints, enters users into security system, defines their roles
  - **Application programmer**
    - Writes the beans and servlets needed for the web-app
  - **Framework programmer**
    - Writes things like specialized tag libraries