Beans in JSP

http://courses.coreservlets.com/Course-Materials/pdf/csajsp2/13-JavaBeans.pdf

Java Beans

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

Bean Properties

- **Usual rule to turn method name into property name**
  - Drop the word "get" or "set" and change the next letter to lowercase. Again, instance var name is irrelevant.
    - Method name: getUserFirstName
    - Property name: userFirstName
- **Exception 1: boolean properties**
  - If getter returns boolean or Boolean
    - Method name: getPrime or isPrime
    - Property name: prime
- **Exception 2: consecutive uppercase letters**
  - If two uppercase letters in a row after "get" or "set"
    - Method name: getURL
    - Property name: URL (not uRL)

Examples

Method Names	Property Name	Example JSP Usage
getFirstName setFirstName	firstName	<jsp:getProperty … property="firstName"/> <jsp:setProperty … property="firstName"/> ${customer.firstName}
isExecutive setExecutive (boolean property)	executive	<jsp:getProperty … property="executive"/> <jsp:setProperty … property="executive"/> ${customer.executive}
getExecutive setExecutive (boolean property)	executive	<jsp:getProperty … property="executive"/> <jsp:setProperty … property="executive"/> ${customer.executive}
getZIP setZIP	ZIP	<jsp:getProperty … property="ZIP"/> <jsp:setProperty … property="ZIP"/> ${address.ZIP}

Why Getters and setters?

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

```
public double speed;
```

- **with**

```
private double speed;
```

Note: in Eclipse, after you create instance variable, if you R-click and choose "Source", it gives you option to generate getters and setters for you.

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

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

Reason 1

1) You can put constraints on values

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

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

Reason 2

2) You can change your internal representation without changing interface

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

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

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

Reason 3

3) You can perform arbitrary side effects

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

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

Basic bean tasks

- **jsp:useBean**
  - In the simplest case, this element builds a new bean. It is normally used as follows:
    - <jsp:useBean id="*beanName*" class="*package.Class*" />
- **jsp:setProperty**
  - This element modifies a bean property (i.e., calls a set*Blah* method). It is normally used as follows:
    - <jsp:setProperty name="*beanName*"
property="*propertyName*"
value="*propertyValue*" />
- **jsp:getProperty**
  - This element reads and outputs the value of a bean property. It is used as follows:
    - <jsp:getProperty name="*beanName*"
property="*propertyName*" />

Where's the advantage?

- Simple interpretation:
  <jsp:useBean id="book1" class="coreservlets.Book" />
  can be thought of as equivalent to the scriptlet
  <% coreservlets.Book book1 = new coreservlets.Book(); %>
- But jsp:useBean has two additional advantages:
  - It is easier to derive object values from request parameters
  - It is easier to share objects among pages or servlets

Where do I put beans?

- **Beans installed in normal Java directory**
  - MyEclipse: src/*folderMatchingPackage*
  - Deployment: WEB-INF/classes/*folderMatchingPackage*
- **Beans must *always* be in packages!**

Explicit type convertion

```
<%
double discountCode = 1.0;
try {
  String discountString =
    request.getParameter("discountCode");
  discountCode =
    Double.parseDouble(discountString);
} catch(NumberFormatException nfe) {}
%>

<%-- setDiscountCode expects a double --%>

<jsp:setProperty
    name="entry"
    property="discountCode"
    value="<%= discountCode %>" />
```

Implicit type conversion

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

Simple automatic conversion

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

Select all

- **Use "*" for the value of the property attribute of jsp:setProperty to indicate that**
  - Value should come from request parameter whose name matches property name
  - Simple automatic type conversion should be performed
- **This is extremely convenient for making "form beans" -- objects whose properties are filled in from a form submission.**
  - You can even divide the process up across multiple forms, where each submission fills in part of the object.

Setting the scope

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

Page, application

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

Session, request

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

Accessing and setting existing beans

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

Example

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

JSP & Beans summary

- **Benefits of jsp:useBean**
  - Hides the Java syntax
  - Makes it easier to associate request parameters with Java objects (bean properties)
  - Simplifies sharing objects among multiple requests or servlets/JSPs

JSP Expression Language

http://courses.coreservlets.com/Course-Materials/pdf/csajsp2/15-Expression-Language.pdf

JSP Expression Language

The JSP 2 EL provides concise, easy-to-read access to

- Scoped variables
- Bean properties
- Collection elements
- Standard HTTP elements such as request parameters, request headers, and cookies

Code semplification

- **When in JSP 2.x-compliant server with current web.xml version, change:**

```
<jsp:useBean id="someName"
             type="somePackage.someClass"
             scope="request, session, or application"/>
<jsp:getProperty name="someName"
                 property="someProperty"/>
```

- **To:**

```
${someName.someProperty}
```

Invoking EL

- **Basic form: ${expression}**
  - These EL elements can appear in ordinary text or in JSP tag attributes, provided that those attributes permit regular JSP expressions. For example:
    -
    - Name: ${expression1}
    - Address: ${expression2}
    -
    - <jsp:include page="${expression3}"/>
- **The EL in tag attributes**
  - You can use multiple expressions (possibly intermixed with static text) and the results are coerced to strings and concatenated. For example:
    - <jsp:include page="${expr1}blah${expr2}"/>

Scope?

- **${varName}**
  - Searches the PageContext, the HttpServletRequest, the HttpSession, and the ServletContext, *in that order*, and output the object with that attribute name.

Dot and Array notation

- **Equivalent forms**
  - ${name.property}
  - ${name["property"]}
- **Reasons for using array notation**
  - To access arrays, lists, and other collections
    - See upcoming slides
  - To calculate the property name at request time.
    - {name1[name2]} (no quotes around name2)
  - To use names that are illegal as Java variable names
    - {foo["bar-baz"]}
    - {foo["bar.baz"]}

Accessing collections

- **${attributeName[entryName]}**
- **Works for**
  - Array. Equivalent to
    - theArray[index]
  - List. Equivalent to
    - theList.get(index)
  - Map. Equivalent to
    - theMap.get(keyName)
- **Equivalent forms (for HashMap)**
  - ${stateCapitals["maryland"]}
  - ${stateCapitals.maryland}
  - But the following is illegal since 2 is not a legal var name
    - ${listVar.2}

Implicit objects

- **pageContext. The PageContext object.**
  - E.g. ${pageContext.session.id}
- **param and paramValues. Request params.**
  - E.g. ${param.custID}
- **header and headerValues. Request headers.**
  - E.g. ${header.Accept} or ${header["Accept"]}
  - ${header["Accept-Encoding"]}
- **cookie. Cookie object (not cookie value).**
  - E.g. ${cookie.userCookie.value} or ${cookie["userCookie"].value}
- **initParam. Context initialization param.**
- **pageScope, requestScope, sessionScope, applicationScope.**
  - Instead of searching scopes.

EL Operators

- **Arithmetic**
  - + - * / div % mod
- **Relational**
  - == eq != ne < lt > gt <= le >= ge
- **Logical**
  - && and || or ! Not
- **Empty**
  - Empty
  - True for null, empty string, empty array, empty list, empty map. False otherwise.
- **CAUTION**
  - Use extremely sparingly to preserve MVC model

Activating EL

- **Available only in servers that support JSP 2.0 or 2.1 (servlets 2.4 or 2.5)**
  - E.g., Tomcat 5 or later, WebLogic 9 or later, WS 6+,
    - Not Tomcat 4 or WebLogic 8 or WebSphere 5
  - For a full list of compliant servers, see http://theserverside.com/reviews/matrix.tss
- **You must use the JSP 2.x web.xml file**
  - Download from coreservlets.com, use one from Tomcat 5 or 6, or Eclipse/MyEclipse will build one for you

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation=
           "http://java.sun.com/xml/ns/j2ee web-app_2_4.xsd"
         version="2.4">
  ...
</web-app>
```

(Selectively) Deactivate EL

- **What if JSP page contains ${ ?**
  - Perhaps by accident, perhaps if you make a custom tag library that also uses ${...} notation and evaluates it directly (as with first release of JSTL).
- **Deactivating the EL in an entire Web application.**
  - Use a web.xml file that refers to servlets 2.3 (JSP 1.2) or earlier.
- **Deactivating the expression language in multiple JSP pages.**
  - Use the jsp-property-group web.xml element
- **Deactivating the expression language in individual JSP pages.**
  - Use <%@ page isELIgnored="true" %>
- **Deactivating individual EL statements.**
  - In JSP 1.2 pages that need to be ported unmodified across multiple JSP versions (with no web.xml changes), you can replace $ with $, the HTML character entity for $.
  - In JSP 2.0 pages that contain both EL statements and literal ${ strings, you can use \${ when you want ${ in the output

Communication Servlet-JSP

Bean

- public String getFirstName(…) { … }

Servlet

- Customer someCust = lookupService.findCustomer(…);
- request.setAttribute("customer", someCust);
- (Use RequestDispatcher.forward to go to JSP page)

JSP

- <h1>First name is ${customer.firstName}</h1>

MVC with Servlets and JSP

http://courses.coreservlets.com/Course-Materials/pdf/csajsp2/14-MVC.pdf

Servlets or JSPs?

- **Servlet only. Works well when:**
  - Output is a binary type. E.g.: an image
  - There is *no* output. E.g.: you are doing forwarding or redirection as in Search Engine example.
  - Format/layout of page is highly variable. E.g.: portal.
- **JSP only. Works well when:**
  - Output is mostly character data. E.g.: HTML
  - Format/layout mostly fixed.
- **Combination (MVC architecture). Needed when:**
  - A single request will result in multiple substantially different-looking results.
  - You have a large development team with different team members doing the Web development and the business logic.
  - You perform complicated data processing, but have a relatively fixed layout.

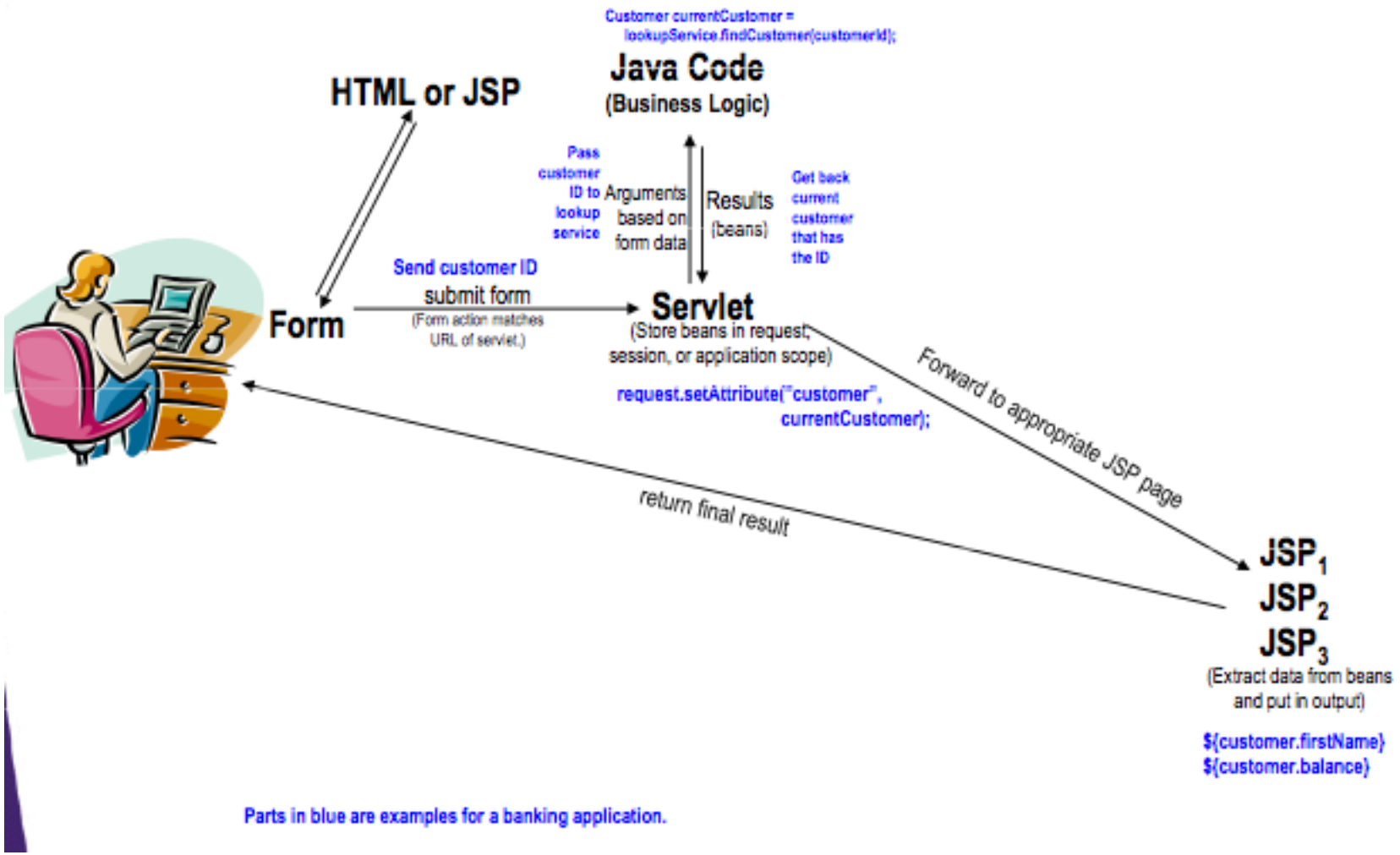
Servlets or JSPs?

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

MVC

- **Use MVC (Model 2) approach when:**
  - One submission will result in more than one basic look
  - Several pages have substantial common processing
  - Your application is moderately complex
- **Approach**
  - A servlet answers the original request
  - Servlet calls business logic and stores results in beans
    - Beans stored in HttpServletRequest, HttpSession, or ServletContext
  - Servlet invokes JSP page via RequestDispatcher.forward
  - JSP page reads data from beans
    - Most modern servers: ${beanName.propertyName}
    - JSP 1.2 servers: jsp:useBean with appropriate scope (request, session, or application) plus jsp:getProperty

MVC


Customer currentCustomer =
lookupService.findCustomer(customerId);
Java Code
(Business Logic)
HTML or JSP
Pass customer ID to lookup service
Arguments based on form data
Results (beans)
Get back current customer that has the ID
Send customer ID
submit form
(Form action matches URL of servlet.)
Form
Servlet
(Store beans in request, session, or application scope)
request.setAttribute("customer", currentCustomer);
Forward to appropriate JSP page
return final result
JSP1
JSP2
JSP3
(Extract data from beans and put in output)
${customer.firstName}
${customer.balance}
Parts in blue are examples for a banking application.

MVC misconception

- **An elaborate framework is necessary**
  - Frameworks are often useful
    - JSF (JavaServer Faces)
      - You should *strongly* consider JSF 2.0 for medium/large projects!
    - Struts
  - They are *not* required!
    - Implementing MVC with the builtin RequestDispatcher works very well for most simple and even moderately complex applications

MVC in 6 steps

1. **Define beans to represent result data**
   - Ordinary Java classes with at least one get*Blah* method
2. **Use a servlet to handle requests**
   - Servlet reads request parameters, checks for missing and malformed data, calls business logic, etc.
3. **Obtain bean instances**
   - The servlet invokes business logic (application-specific code) or data-access code to obtain the results.
4. **Store the bean in the request, session, or servlet context**
   - The servlet calls setAttribute on the request, session, or servlet context objects to store a reference to the beans that represent the results of the request.

MVC in 6 steps

5. **Forward the request to a JSP page.**
   - The servlet determines which JSP page is appropriate to the situation and uses the forward method of RequestDispatcher to transfer control to that page.
6. **Extract the data from the beans.**
   - JSP 1.2 (Old!)
     - The JSP page accesses beans with jsp:useBean and a scope matching the location of step 4. The page then uses jsp:getProperty to output the bean properties.
   - JSP 2.0 (Preferred!)
     - The JSP page uses ${nameFromServlet.property} to output bean properties
   - Either way, JSP page does not create or modify bean; it merely extracts and displays data that servlet created.

Example

- **Bean**
  - public String getFirstName(…) { … }
- **Servlet**
  - Customer someCust = lookupService.findCustomer(…);
  - request.setAttribute("customer", someCust);
  - (Use RequestDispatcher.forward to go to JSP page)
- **JSP**
  - <h1>First name is ${customer.firstName}</h1>

Example

- **Servlet**

```
ValueObject value = LookupService.findResult(...);
HttpSession session = request.getSession();
session.setAttribute("key", value);
RequestDispatcher dispatcher =
  request.getRequestDispatcher
                      ("/WEB-INF/SomePage.jsp");
dispatcher.forward(request, response);
```

- **JSP 1.2**

```
<jsp:useBean id="key" type="somePackage.ValueObject"
             scope="session" />
<jsp:getProperty name="key" property="someProperty" />
```

- **JSP 2.0**

```
${key.someProperty}
```

Beware of forward!

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

```
<img src="foo.gif" …>
<link rel="stylesheet"
      href="my-styles.css"
      type="text/css">
<a href="bar.jsp">…</a>
```

  - Browser treats addresses as relative to *servlet URL*

Forward or redirect?

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

Example

- See the Bank Balance lookup example in

http://courses.coreservlets.com/Course-Materials/pdf/csajsp2/14-MVC.pdf

(slide 30-42)