Java Beans

Dr. Michael Whitney

** Slides influenced by both textbooks

Uses of JSP Constructs

Simple Application

Complex Application

- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Servlet/JSP combo (MVC)
- MVC with JSP expression language
- Custom tags
- MVC with beans, custom tags, and a framework like Struts or JSF

Why Beans?

1. Encapsulation: hides the Java syntax:
    a. Separates content and presentation.

2. Makes it easier to associate request parameters with Java objects (bean properties).

3. Simplifies sharing objects among multiple requests or servlets/JSPs.

Background: What Are Beans?

Java classes that follow certain conventions:

1. **Must have a zero-argument (empty) constructor:**
   - You can satisfy this requirement by explicitly defining such a constructor.
2. **Should have no public instance variables (fields):**
   - You should already follow this practice and use accessor methods instead of allowing direct access to fields.
3. **Implements the Serializable interface - an object can be represented as a sequence of bytes:**
   - A Bean persists by having its properties, fields, and state information saved and restored to and from storage. The mechanism that makes persistence possible is called serialization.
   - All Beans must persist - Beans must implement the java.io.Serializable interface.

For more on beans, see http://java.sun.com/beans/docs/

Why You Should Use Accessors, Not Public Fields

To be a bean, you cannot have public fields.

So, you should replace:

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

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

Why You Should Use Accessors, Not Public Fields

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.

Why You Should Use Accessors, Not Public Fields

2) You can change your internal representation without changing interface

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

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

Why You Should Use Accessors, Not Public Fields

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.

Using Beans: Basic 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:getProperty

This element reads and outputs the value of a bean property. It is used as follows:

<jsp:getProperty name="*beanName*" property="*propertyName*" />

jsp:setProperty

This element modifies a bean property (i.e., calls a method of the form set*Xxx*).

<jsp:setProperty name="*beanName*"
property="*propertyName*"
value="*propertyValue*" />

Building Beans: jsp:useBean

Format

<jsp:useBean id="*name*" class="*package.Class*" />

Purpose

Allow instantiation of Java classes without explicit Java programming (XML-compatible syntax)

Notes

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.

Accessing Bean Properties: jsp:getProperty

Format

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

Purpose

Allow access to bean properties (i.e., calls to getXxx methods) without explicit Java programming

Notes

```
<jsp:getProperty name="book1" property="title" />
```

is equivalent to the following JSP expression

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

Setting Simple Bean Properties: jsp:setProperty

Format

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

Purpose

Allow setting of bean properties (i.e., calls to setXxx methods) without explicit Java programming

Notes

```
<jsp:setProperty name="book1" property="title" value="Core Servlets and JavaServer Pages" />
```

is equivalent to the following scriptlet

```
<% book1.setTitle("Core Servlets and JavaServer Pages"); %>
```

Example: StringBean

```
package coreservlets;
public class StringBean {
	private String message = "No message specified";
	public String getMessage() {
		return(message);}
	public void setMessage(String message) {
		this.message = message;}}
```

Stringbean has a string property called message.

Using JavaBeans with JSP

Using JavaBeans with JSP

```
<h1>using javabeans with jsp</h1>
<jsp:usebean id="stringBean" class="coreservlets.stringbean" />
<ol>
    <li>Initial value (from jsp:getproperty): <em><jsp:getProperty name="stringBean" property="message" /></em></li>
    <li>Initial value (from jsp expression): <em><%= stringbean.getMessage() %></em</li>>
    <li><jsp:setProperty name="stringBean" property="message" value="Best string bean: Fortex" />
        Value after setting property with jsp:setproperty:
        <em><jsp:getProperty name="stringBean" property="message" /></em></li>
    <li><% stringBean.setMessage("My favorite: Kentucky Wonder"); %></li>
        Value after setting property with scriptlet:
        <em><%= stringBean.getMessage() %></em>
    </li>
</ol>
```

Sharing Beans

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.

Values of the scope Attribute

Values of the scope Attribute

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. Allows methods in the same servlet access t the 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).

Values of the scope Attribute

Values of the scope Attribute

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.

Bean Example (from Murach's)

```
package murach.business;
import java.io.Serializable;    // Convention #3 - class contains get/set/is methods available to other classes to write to object

public class User implements Serializable {
        private String firstName; //Convention #2 - no public variables
        private String lastName;

        public User() {      // convention #1 - 0 argument constructor
                firstName = "";
                lastName = "";
        }

        public User(String firstName, String lastName) {
                this.firstName = firstName;
                this.lastName = lastName;
        }

public String getFirstName() {
        return firstName;
}
public void setFirstName(String firstName) {
        this.firstName = firstName;
}
public String getLastName() {
        return lastName;
}
 public void setLastName(String lastName) {
        this.lastName = lastName;
}
}
```

Code that uses JSP tags to access the User Bean

```
<%@ page import="murach.business.User" %>
<%
        User user = (User) request.getAttribute("user");
        if (user == null) {
                user = new User();
        }
%>

<label>First Name:</label>
<span><%= user.getFirstName() %></span><br>
<label>Last Name:</label>
<span><%= user.getLastName() %></span><br>
```

```
<jsp:useBean id="user" scope="request" class="murach.business.User"/>

<label>First Name:</label>
<span><jsp:getProperty name="user" property="firstName"/></span><br>
<label>Last Name:</label>
<span><jsp:getProperty name="user" property="lastName"/></span><br>
```