Introduction to Web Engineering SENG2050/6050

Lecture 3b
Java Beans

Review

- Good Design - Web Engineering
- Java Server Pages (JSP)
- Scripting Elements
  - ✓ Expressions
  - ✓ Scriptlets
  - ✓ Declarations

Lecture 3a: Java Beans

- Java Beans
  - Conformance
  - Good Usage
- JSP and Java Beans

Good Design - Web Engineering

- Separate the user interface from the "business logic"
  - ✓How?
- Implement the interface in HTML+CSS and the logic in Java
  - ✓Okay, but *how*?

JSP lets you mix "business logic"-> Java and "interface"-> HTML+CSS

Good Design - Web Engineering

E.g., the “feedback” form is submitted to a JSP…

```
<%
  String name =
  request.getParameter("name");
%>
…
<p>
 <%= name %>.
 Thank you for your feedback,
 Please visit us again.
</p>
…
```

Good Design - Web Engineering

E.g., the “temperature” form is submitted…

```
<%!
  double kelvinToCelsius(double k) {…}
  double kelvinToFarenheit(double k) {…}
%>
<%
  double kelvin =
     request.getParameter("temperature");
%>
…
  <td><%= kelvinToCelsius() %></td>
  <td><%= kelvinToFarenheit() %></td>
…
```

Good Design - Web Engineering

- The idea was to separate the Java from the HTML; however, in these examples, the Java and HTML are all mixed together!
  - ✓Exactly! JSP itself is not enough to provide real separation of interface and business logic.
- A solution:
  - ✓Create a new Java class that does all calculations.
  - ✓Create an instance of the class in your JSP, then access its methods to get the required dynamic values.

Java Beans

- JavaBeans are classes written in Java programming language.
- A Java Bean follows a set of rules for accessing and changing its values
  - ✓They are used to encapsulate many objects into a single object (the bean), so that the bean can be passed around rather than the individual objects.
  - ✓JSP has built-in support for accessing the methods of a bean without having to explicitly write any Java code in the JSP.

Java Beans

The Java Bean Rules:

- ✓A bean class must have a zero-argument constructor.
- ✓A bean class should have no public attributes (all private).
- ✓Persistent values should be accessed through methods called `get`*Xxx* (for non-Booleans) or `is`*Xxx* (for Booleans).
- ✓Persistent values should be mutated (changed) through methods called `set`*Xxx.*
- ✓The class should be `serializable` (able to persistently save and restore its state) .
- ✓It should not contain any required event-handling methods.

Java Beans

```
public class PersonBean implements java.io.Serializable{
  private String name;
  private boolean deceased;

  public PersonBean() { }

  public String getName() {
    return this.name; }
  public void setName(String name) {
    this.name = name; }
  public boolean isDeceased() {
    return this.deceased; }
  public void setDeceased(boolean deceased) {
    this.deceased = deceased; }
}
```

Java Beans

- By convention…
  - ✓The attribute name should start with a lowercase letter
  - ✓The corresponding methods should be `set`*Xxx*, `is`*Xxx* and `get`*Xxx*, where the first *X* is capitalized.
- However, an `is`*Xxx* or `get`*Xxx* method might not match any of the attributes explicitly.
  - ✓It could calculate a value dynamically – e.g., `getDate()`.
- Also, a `set`*Xxx* method might not match any of the attributes explicitly (this is less common).
  - ✓Setting a "value" could actually set several attributes.

Java Beans – Why Bother?

- With a zero-argument constructor you can create an instance of a bean without knowing anything about it
  - ✓This is good coding practice.
  - ✓Allows interfaces to remain fixed while implementation details inside a bean change (or a new bean is installed).

Java Beans – Why Bother?

- Without `get`*Xxx*, `set`*Xxx* and `is`*Xxx*, introspection doesn't work
  - Introspection is used to discover what properties a bean has – their names, if they are simple or indexed, their type and the getters and setters methods.
  - Without introspection, the simplicity of Java Bean-based component use starts to fall apart.

Java Beans – Why Bother?

- If you obey the Java Bean rules…
  - ✓Other programmers, and even automatic programming tools, will be able to use your classes more easily.
  - ✓You will be able to reuse others' beans.
- If you do not…
  - ✓You have to explicitly write interfaces between JSP and your classes.
  - ✓It will be harder for others to use your classes.

JSP and Java Beans

default scope

- `<jsp:useBean id="beanInstanceName" scope="page|request|session|application" class="Classname" />`
  - ✓A JSP "action"
  - ✓Creates an instance of *Classname* and binds it to the variable *beanInstanceName*

```
<jsp:useBean id="pb" class="PersonBean"/>
<% pb.setName("John"); %>
<% pb.setDeceased(False); %>
Name: <%= pb.getName() %> is <%=
  pb.isDeceased() %>
```

JSP and Java Beans

`scope` controls the visibility of the bean

- ✓`scope="page"` – the bean can be used within this page only. A new bean is created for each request.
- ✓`scope="request"` – the bean can be used in any JSP that is processing the same request.
- ✓`scope="session"` – the bean can be used in any page that the user accesses during this session.
- ✓`scope="application"` – the bean can be used in any page in the current application.

JSP and Java Beans

- If an instance exists with the same `id` and the same `scope`, then it is reused, otherwise a new instance is created
  - ✓A bean instance is stored in the `page`, `request`, `session` or `application` implicit object, that is, `pageContext`, `HttpServletRequest`, `HttpSession` or `ServletContext` respectively.
  - ✓Each of these objects has methods `setAttribute(`*`name`*`,` *`object`*`)` and *`getAttribute(name)`* providing you with access to the bean.

JSP and Java Beans

Once you have a Java Bean instance, you can use special JSP actions to set and get its values…

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

- ✓Uses the bean's accessor method.
- ✓Calls *beanInstanceName*`.get`*PropertyName*`()`.

`<jsp:getProperty name="pb" property="name" />` is equivalent to `<%= pb.getName() %>`

- ✓Also works for `is`*PropertyName*`()` methods

JSP and Java Beans

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

- ✓Uses the bean's mutator method, *beanInstanceName*`.set`*PropertyName*`(`*newValue)*

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

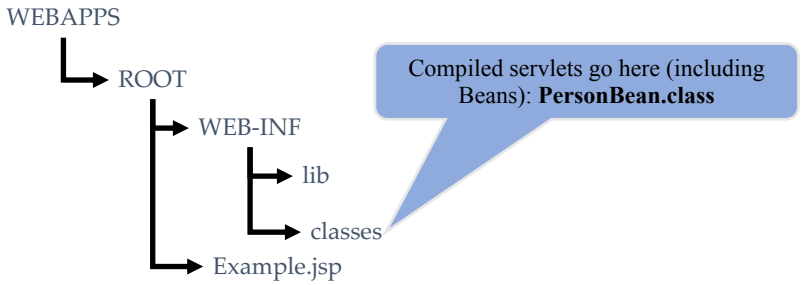
is equivalent to

```
<% pb.setName("Cesar"); %>
```

JSP and Java Beans

- ➢Note the capitalisation of `property="data"`, `getData()` and `setData()`
  - ✓This relationship **must** hold for `jsp:setProperty` and `jsp:getProperty` to work!
  - ✓This is independent of the private attribute name within the Java Bean class.

Where Do the Beans Go?



- If a Bean is in a "package", then must use directory names the same as the package names

Example


```
public class PersonBean implements java.io.Serializable{
 private String name;
 private boolean deceased;
 public PersonBean() { }
 public String getName() {
   return this.name; }
 public void setName(String name) {
   this.name = name; }
 public boolean isDeceased() {
   return this.deceased; }
 public void setDeceased(boolean deceased) {
   this.deceased = deceased; }
}
```

Example

```
<jsp:useBean id="PersonBean" class="package.PersonBean" />

  <li>Initial value (getProperty):
      <i><jsp:getProperty name="PersonBean" property="name" /></i>

  <li>Initial value (JSP expression):
      <i><%=PersonBean.getName() %></i>

  <li><jsp:setProperty name="PersonBean"
      property="name" value="Cesar" />
      after setting property with setProperty:
      <i><jsp:getProperty name="PersonBean" property="name" /></i>

  <li><% PersonBean.setName("Michael"); %>
      after setting property with scriptlet:
          <i><%= PersonBean.getName() %></i>
```

JSP and Java Beans

- Why use `jsp:getProperty` and `jsp:setProperty`?
  - ✓Enforces use of the Java Bean interface – you can't bypass it, thus can't break all the nice "separation" features.
  - ✓You can reuse other people's beans without writing any Java code.
  - ✓Can even be used by non-programmers – e.g., Web page designers only need `jsp:useBean`, `jsp:getProperty` and `jsp:setProperty` to have full access to a beans dynamic behaviuor.

JSP and Java Beans

- You can use a JSP expression to set the value in `jsp:setProperty`

```
<jsp:setProperty name="PersonBean" param="name"
value='<%= request.getParameter("name") %>' />
```

note: single quote

- What if no name parameter was given?
  - ✓ Be careful not to access null properties.
  - ✓ Your Java Bean should account for this.

JSP and Java Beans

- Setting a bean from a parameter is so common that JSP has a special syntax for it

```
<jsp:setProperty
  name="beanInstanceName"
  property="propertyName"
  param="parameterName" />
```

- ✓ Bonus! This tag automatically converts the parameter `String` into any of the Java built-in types (`byte`, `short`, `int`, `long`, `float`, `double`, `boolean`, `char`) and equivalent objects (`Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `Boolean`, `Character`) – **but may throw exceptions**

JSP and Java Beans

- `jsp:setProperty` can also set **all** properties from their corresponding parameters in one go!

```
<jsp:setProperty name="beanInstanceName"
property="*" />
```

  - ✓Sets every property for which a parameter is passed.
  - ✓Property names and parameter names must match exactly – including capitalization.
  - ✓If a parameter is not passed, then the property is not set – missing parameters are not set to null.

Creating Beans Conditionally

```
<jsp:useBean … scope="scope">
    Java statements
 </jsp:useBean>
```

  - ✓If a new instance is created, then execute the contained Java statements.
  - ✓Used to initialise a bean when it is unknown which of several pages will use it first – initialisation can be page-specific .

Thinking Java Beans

- Using Java Beans successfully requires a particular way of thinking
  - ✓To implement `int s = sum(21, 15)` as a bean…

```
private int arg1 = 0;
public void setArg1(arg1)
  { this.arg1 = arg1; }
public int getArg1() { return arg1; }
// similarly arg2
public int getSum()
  { return getArg1() + getArg2() };
```

Thinking Java Beans

- then…

```
<jsp:useBean id="sum" class="package.Sum" />
<jsp:setProperty name="sum"
  property="arg1" value="21" />
<jsp:setProperty name="sum"
  property="arg2" value="15" />
<jsp:getProperty name="sum"
  property="sum"/>
```

  - ✓if `arg1` and `arg2` had been submitted by an appropriate form, then all `jsp:setProperty` tags could be replaced by one with `property="*"`

Thinking Java Beans

- Note that I used `getArg1()` and `getArg2()` inside the `getSum()` method
  - ✓This is more good coding practice.
  - ✓If I decide to change the way `arg1` is stored, then I only have to change `getArg1()` and `setArg1()` – other methods which need to access or mutate the value of `arg1` won't need to be changed.
  - ✓As a general rule, only the corresponding set and get methods should directly access a private attribute – all other methods (even within the same class) should go through these methods.

Thinking Java Beans

The general pattern for using a Java Bean is:

1. Store all "inputs" in the bean using `jsp:setProperty`
   - ✓ If these properties are derived from form parameters, then use property="*".
2. Get the "results" from the bean using `jsp:getProperty`
   - ✓ The get method will calculate the results from the current set inputs.

Thinking Java Beans

What if I want to set multiple values?

- ✓Use multiple `jsp:setProperty` tags.
- ✓Be sure to check that all properties have been set at the start of any get method that uses them.
- ✓If the set properties are not valid, then throw an exception (JSP has a mechanism for handling these exceptions – more on this later).
- ✓An `isInputsValid()` method is also useful.

Thinking Java Beans

What if I want to return multiple values?

- ✓Use multiple `jsp:getProperty` tags.
- ✓Each get method should check if the result has already been calculated: if it has, then simply return the required result, otherwise calculate and store all results, then return the part requested.
- ✓If a set method changes one of the inputs (making the result invalid with respect to the current input set), then it should change all parts of stored result to be invalid.
- ✓An `isResultsValid()` method is also useful (or individual `isResultValid()` methods).

Thinking Java Beans

What if I want to call a method with no arguments?

E.g., resetAllToDefaults() to be called before setProperty "*"

- There are two ideas to solve this…

1. Get status of result
   - isResetSuccessful() or
   - getResetStatus(), returning whether the method was successful or not.
2. Set with “dummy” argument
   - setAllToDefaults(true), where true is a dummy boolean argument

More Examples

➢JavaBeans Tutorial

✓https://docs.oracle.com/javase/tutorial/javabeans/TOC.html

What if?

- The JSP + Java Beans architecture described so far works when we want a single response
  - ✓ But what if we want a different response (completely different HTML page, not just different dynamic values) depending on the user's inputs?

1. Java embedded in HTML – yuck!

```
<% if (bean.getType() == 1) { %>
  <!-- HTML template 1 -->
<% } else if (…) { %>
```

2. HTML output by Java Bean – Double yuck!

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

The Model View Controller (MVC) Architecture

3. Use one JSP to store all parameters in a Bean, then dynamically "forward" responsibility for outputting the response to another JSP
   - ✓ The programmer writes the first JSP – it only contains Java and JSP actions, no HTML.
   - ✓ The page designer writes the other JSPs – they only contain HTML and `jsp:getProperty` tags.
   - ✓ This is the essence of the "Model View Controller" – MVC architecture (more on this later).

MVC Architecture

- Forwarding to another page is done with the JSP action `<jsp:forward page="url" />`
  - ✓Passes responsibility for the request to the relative *url* (**must be on the same server**).
  - ✓Preserves the `request` implicit object.
  - ✓Shares beans unless `scope="page"` (Any other is ok).
  - ✓The `page` attribute can be a JSP expression

```
<jsp:forward page="<%=
 bean.getNextPage() %>" />
```

MVC Architecture

- `jsp:forward` is handled inside the server
  - ✓The server treats it as a single request
  - ✓The client sees it as the original request's URL
- `response.sendRedirect(`*url*`)` is handled by the client (Web browser)
  - ✓The client sees the new URL
  - ✓The server treats it as two requests – the request object is not preserved, and beans instances must have `session` or `application` scope to be shared

Java Bean Resources

- JavaBeans Tutorial
  - ✓https://docs.oracle.com/javase/tutorial/javabeans/TOC.html
- ✓Lots of Java Bean Resources, including links to free beans and books
  - ✓https://www.tutorialspoint.com/jsp/jsp_java_beans.htm

JSP Resources

- Java Server Pages (JSP)
  - ✓https://www.tutorialspoint.com/jsp/jsp_java_beans.htm
- Training Materials from the textbook
  - ✓http://courses.coreservlets.com/Course-Materials/
- Web
  - ✓http://www.jsptut.com/

THE END

QUESTIONS??

THANKS!!