Struts Lab 3: Creating the View

In this lab, you will create a Web application that lets a company's fleet manager track fuel purchases for the company's vehicles. You will concentrate on creating the presentation layer (view) using Struts, JSTL and HTML tags.

Objectives:

- To work with the View portion of a Struts MVC application
- To use dynamically generated ActionForm classes

Steps:

_1. Ask your instructor for the following information:

Lab Installation directory: _____

If your computer has the standard lab setup, the directories will be something like:

```
Lab Installation Directory:       c:\strutsclass
```

_2. Start WSAD if it's not running and ensure that you are in the Web perspective.

_3. Your first job is to create a *project* for your lab.

a. Choose *File - New - Dynamic Web Project*. On the first page of the wizard, enter:

```
Project name:                 strutslab03Web
Project location:             accept the default
Configure advanced options:   checked
```

Then press Next.

b. On the *J2EE Settings Page*, enter:

```
EAR Project:  strutslab03EAR
```

Then press Next.

c. On the *Features* page, enter:

```
Struts support:           checked
```

```
JSP Standard Tag Library: checked
```

Then press Finish.

_4. Next you will import some Java files to get you started:

a. Click the *JavaSource* folder to highlight it, then right-click on *JavaSource* and choose **Import...**

Note: In some versions of WSAD, the folder is named *JavaResources* instead of *JavaSource*.

b. Choose **File System**.

c. For the *From directory*, click the **Browse** button and navigate to the **{Lab Installation directory}/starters/strutslab03** directory.

d. Expand the *strutslab03* folder in the left window pane, then highlight it.

e. Place a checkmark next to the **purchase** directory, then press Finish.

f. Back in the project, examine the *JavaSource* folder to see the newly imported files.

_5. Import an HTML file to help you get started:

a. Highlight the *WebContent* folder, right-click on it, then choose **Import...**.

b. Choose **File System**.

c. Pull down the drop-down list to get back to the **{Lab Installation directory}/starters/strutslab03** directory.

d. Expand and highlight the *strutslab03* folder in the left window pane.

e. Place a checkmark next to the **purchases.html** file, then press Finish.

f. Back in the project, examine the newly imported file and ignore any *Broken link* warnings for now.

_6. The imported *purchase* package contains two pre-written classes:

- **GasPurchase:** A JavaBean class that represents a fuel purchase, with appropriate properties
- **PurchaseList:** A *Singleton* class that implements a list of GasPurchases

Please carefully examine these classes and ensure that you understand them, especially the properties in the GasPurchase class.

_7. The imported input form is **purchases.html**. You should edit this file and preview it and ensure that you understand it.

_8. Rename the file to **purchases.jsp** so it can work with the Struts framework.

_9. Next you must convert the input form so it works with Struts and JSTL. Edit **purchases.jsp** and update it:

- Write the appropriate *taglib* directive so you can use the Struts HTML custom actions
- Write the appropriate *taglib* directive so you can use the JSTL core custom actions
- At the top, under the *taglib* directives, add the following so your JSP can access the *PurchaseList* object. You cannot use the standard *jsp:useBean* action since it's a Singleton object:

```
<%
    purchase.PurchaseList purchases =
            purchase.PurchaseList.getInstance();
    pageContext.setAttribute ( "purchases", purchases );
%>
```

Classes that implement the *Singleton* design pattern do not expose a public no-argument constructor, and thus cannot be instantiated by the standard JSP actions. It would probably be a good idea to write a new custom action so that we could avoid the scripting element, but that's left as an exercise for the student.

- In the HTML table that displays the current purchases, use the JSTL core *forEach* action to iterate through the list of purchases, outputting a row for each purchase, with a column for each property in the *GasPurchase* bean. Here is some code to get you started:

```
<c:forEach var="purchase" items="${purchases.purchaseList}">
  <tr>
    <td><c:out value="${purchase.purchaseDate}"/></td>
    <td><c:out value="${purchase.purchasePrice}"/></td>
  . . .
  </tr>
</c:forEach>
```

  You should insert this code immediately after the </tr> tag that closes the table header.

- Convert the HTML form so it's a Struts form that references an *action* named **AddPurchase.action**. You will still have an "unresolved" warning since you have not yet defined the action.
- Convert each *input*, *select*, *option* and *submit* HTML elements. to their Struts equivalents. You should choose property names that match properties in the *GasPurchase* JavaBean. **Warning:** Be sure to properly close each tag, either explicitly or by using the XML empty-tag syntax!

_10. Next you can create the *Action* class that will respond to the request:

a. Create a new package named **actions** in the *JavaSource* folder.

b. In the new package, create a class named **AddPurchase** that is a subclass of **org.apache.struts.action.Action**.

c. In the new class, override the *execute* method from the superclass.

  Hint: To override a method in WSAD, right-click on the editing pane and choose **Source - Override/Implement Methods**. Note that if you do this, be sure to select the correct method from the superclass (you want the one with "Http" parameters). Also, after overriding, we recommend that you delete the generated *return* statement and change the parameter names to something a little less generic. See the course notes for example names.

  You will complete the new method in the next step.

_11. To complete the *execute* method, follow these steps:

a. In the *execute* method, cast the generic *ActionForm* parameter to a variable of type **DynaActionForm**.

b. Retrieve all of the user input from the DynaActionForm, storing each as a separate String (we will convert types in a moment). The fields to retrieve are:

  - gallons
  - purchaseDate
  - creditCard
  - discounted
  - milesDriven

- purchasePrice

Here is some code to get you started (assuming that you named the cast DynaActionForm *theForm*:

```
String sgallons = (String)theForm.get ( "gallons" );
```

c. Convert the non-String properties (**gallons**, **milesDriven** and **purchasePrice**) to *double* variables. Here is some code to get you started:

```
double gallons = Double.parseDouble ( sgallons );
```

d. Create an instance of the *GasPurchase* JavaBean.

e. Call the *setter* methods on the JavaBean to set its properties using data extracted from the form. **Note:** To set the boolean property, you can use code like:

```
if ( sdiscounted.equals ( "on" ) )
    gp.setDiscounted ( true );
else
    gp.setDiscounted ( false );
```

f. Add the GasPurchase bean instance to the PurchaseList:

```
PurchaseList purchaseList = PurchaseList.getInstance();
purchaseList.addPurchase ( gp );
```

g. Return a mapping ActionForward with a string: **success**.

_12. Now you must configure a Form bean for Struts. Follow these steps:

a. Ensure that you are in the Web perspective.

b. Click the *Struts Explorer* tab at the bottom of the upper-left window pane.

c. Expand the *strubslab03Web* and *default module* folders.

d. Right-click the *FormBeans* folder and choose **Add Form Bean**.

e. Then enter or select:

```
Form Bean Name:         AddPurchaseForm
Create:                 selected
Model:                  dynaform using Dynamic Action Form
```

Then press Finish.

_13. The dynamic form bean you just created is bereft of properties. To add the properties for the gas-purchase form, follow these steps:

a. In the Struts Explorer, double-click the newly added form bean to open the *Struts Configuration File Editor*.

b. Click on the *FormBeans* tab at the bottom of the editor pane.

c. At the top of the editor pane, click on the *FormProperties* tab.

d. In the *Form Properties* section, click the **Add** button and set the new property's name to **purchaseDate**.

e. Make sure that *purchaseDate* is selected, then in the *Form Beans Attributes* section, set the *Type* to **java.lang.String**.

f. Repeat the two steps above for the remaining properties, each of which should be Strings:

- purchasePrice
- gallons
- milesDriven
- discounted
- creditCard

_14. Next you must configure the Action class. Follow these steps:

a. In the Struts Explorer, right-click the *Actions* folder and choose **Add Action Mapping**.

b. Then enter or select:

```
Action Mapping Path:    /AddPurchase
Forwards:               Add a single forward:

    Name            Path
    ----            ----
  success          /purchases.jsp

Form Bean Name:            AddPurchaseForm
Form Bean Scope:           request
An Existing Action Class:  selected
Reference:                 actions.AddPurchase
```

Then press Finish.

_15. Double-click the newly added action to bring up the *Struts Configuration File Editor* and ensure that you are in the *ActionMappings* tab (see the bottom of the editor pane). In the *Action mapping attributes* section, enter or select:

```
Input:     /purchases.jsp
```

In the *Form Bean Specification* section, enter or select:

```
Validate:  No
```

Press Ctrl+S to save your changes.

_16. Press the *Source* tab at the bottom of the editor and ensure that you understand the Struts configuration file.

_17. Next, you need to edit the deployment descriptor. Go back to the *Project Navigator* instead of the *Struts Explorer*. Find the *Web Deployment Descriptor* entry in the *WebContent* folder and double-click it to open it in an editor.

At the bottom of the editor window, press the *Servlets* tab. Then find the *URL Mappings* section and change the **.do* entry to ***.action**.

You should have no broken link warnings after this completing this step.

_18. You can now test your Web application. Find the *purchases.jsp* file under *WebContent*, right-click it and choose **Run on server...** and press Finish.

You should see an empty Purchases table and a form to let you enter a new purchase. Fill in appropriate values in the form and press the **Add Purchase** button -- the action should update the list of purchases and send you back to the *purchases.jsp* page (please re-examine the *action* mapping in the struts-config.xml file to be sure you understand this).

Try adding a purchase with bad data (e.g. *price* set to "abcd") -- we will fix this issue in the next lab by adding validation to the Web application.