JSP Lab 3: JSPs, MVC and JavaBeans

This lab uses a similar HTML input form as the last lab - it lets the user register for a conference. In the last lab, you coded to an architecture known as *Model 1*, in which JSPs respond directly to forms and do all of the work. The downside of Model 1 is that the JSPs tend to be full of scriptlet code, which obscures the HTML and makes them difficult for non-programmers to work with.

In this lab, you will convert the conference-registration application to use *Model 2*, also known as *model-view-controller* (MVC). This lab will use a servlet that responds to the form and then creates and initializes a JavaBean that represents a registration record. The servlet will store a reference to the bean at session scope, then redirect the request to the JSP for display. This will reduce the number of lines of scriptlet code in the JSP and make it easier to work with.

Because we have not yet covered servlets or MVC in great detail, we will provide you with a fully completed servlet. Your job is to modify your JSP from the last lab so it works in this new architecture and uses the JavaBean.

The following figure illustrates the lab's components:

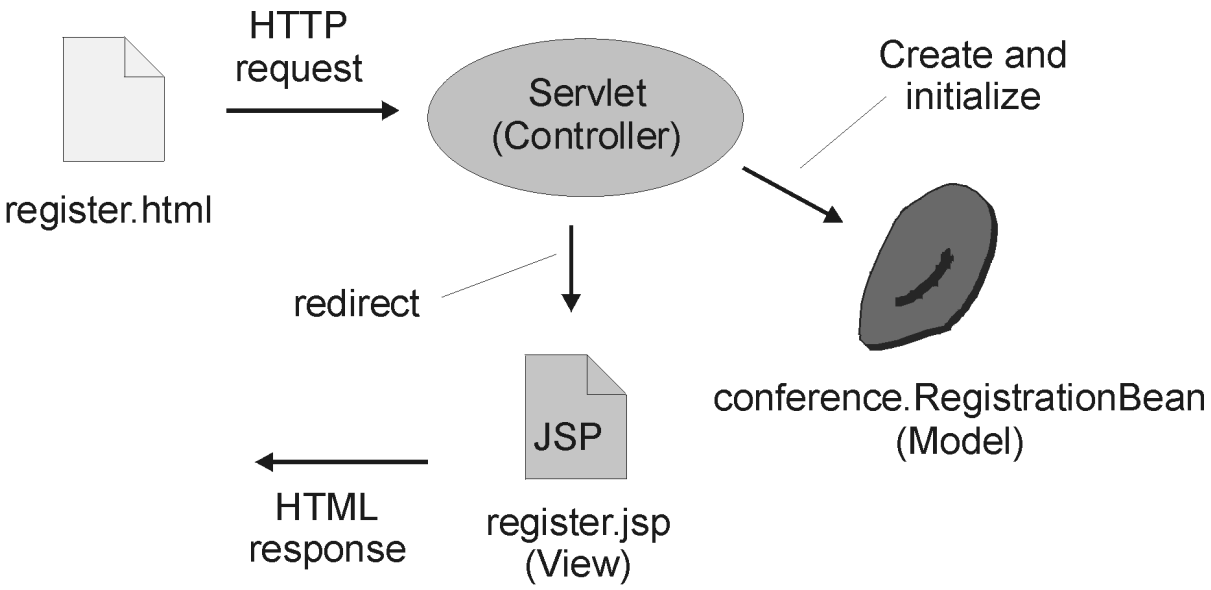

Figure 1: Model 2 Architecture

Objectives:

- To take a look at the MVC design pattern for web applications
- To write a JSP that uses a JavaBean

Steps:

_1. Ask your instructor for the following information:

BEA Installation directory: _____

HTML Deployment directory: _____

Lab Installation directory: _____

BEA Username and password: _____

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

```
BEA Installation directory:    c:\java\bea
HTML Deployment directory:     c:\j2eeclass\DefaultWebApp
Lab Installation directory:    c:\j2eeclass
BEA Username and password:     administrator for both
```

_2. Use Windows Explorer to copy **register.jsp** from the last lab to the **{Lab Installation directory}/jsplab03** directory. Do not copy the other files - we've provided you with alternate versions that work with MVC.

Note: if you did not finish the last lab, you can copy this file from the last lab's **solutions** directory.

_3. **Important:** Please ensure that you performed the steps in the last lab to prevent your browser from caching files. Also be sure that you have closed the browser after performing that task.

_4. Edit the provided **register-m2.html** file in the **jsplab03** directory and note how the form invokes the Controller servlet instead of the JSP as it did in the last lab.

_5. In the **jsplab03** directory, you should see a subdirectory, **conference**, that contains the **RegistrationBean.java** file. This is mostly completed (to save you typing) and contains a JavaBean that represents a conference registration. Please edit this file and examine it.

_6. In RegistrationBean.java, note that the Bean defines several properties and a single logic method, *calculateAmountDue*. Please write down the names and types of each of the properties that this Bean exposes - recall that you can infer a property by looking for "getters" and "setters".

_7. Then complete the calculateAmountDue method so that it returns the amount due, based on the number of attendees, the number of CDs and whether or not a discount should be applied. Note that there are symbolic constants defined near the top of the class that you can use. You can copy logic from your last lab's JSP, if you wish.

_8. In the **jsplab03** directory, you should see a file named **Controller.java** - this is the servlet that acts as a controller for this MVC application. The file is already completed, but you should open this file in your editor and note that the servlet:

- Creates an instance of a RegistrationBean and initializes its properties from the client's request
- Stores a reference to the Bean into the *session* object, thus making it available for a JSP under the name "regBean" at session scope
- Redirects the request to the **register.jsp** JSP

Don't worry too much about the actual code -- we will cover it later in the class. At this point in the course, it's only important that you understand the concepts.

_9. The time has come to update the JSP so it works as the view in our MVC application. This JSP should produce output similar to the JSP in the last lab, but will not retrieve information from the form. Instead, this JSP will obtain a reference to the JavaBean and then retrieve Bean properties and call Bean methods. Edit **register.jsp** and modify it:

- Delete all of the scriptlets
- Near the top, insert a *useBean* element that retrieves a reference using the *id* "regBean" at *session* scope (recall that the servlet stored the Bean reference using that name into the session)
- Display the same information as in the last lab, but here the JSP should retrieve the information from the Bean using *getProperty* elements
- To display the amount due, call the Bean's *calculateAmountDue* method from a scriptlet or call it from within a JSP *expression*

_10. Now it's time to compile and deploy your application. As in the last lab, you can use Ant to do all of this.

Open the **build.xml** file from the **jsplab03** directory into your editor and see if you can make sense of it. Note that the *deploy* target depends on the *build* target. That means that Ant will first run the build, which compiles the servlet and Bean, followed by the deploy, which copies the files to the deployment directories.

To build and deploy using Ant, open a command prompt and change to the **jsplab03** directory and then run Ant. For example, assuming that the labs are installed in the **jspclass** directory on drive C:

```
c:
cd \j2eeclass\jsplab03
ant deploy
```

Be sure to examine the results of running Ant for error messages and fix any compile errors before continuing.

_11. Next you need to configure the WebLogic container so it knows about your servlet. Use your editor to open the **{Lab Installation directory}/DefaultWebApp/WEB-INF/web.xml** file.

In this XML deployment descriptor, add this servlet definition immediately following any existing "servlet" element (if there are no "servlet" elements, put it before the "welcome-file-list" element:

```
<servlet>
  <servlet-name>Controller</servlet-name>
  <servlet-class>Controller</servlet-class>
</servlet>
```

Then add this servlet-mapping definition immediately following any existing "servlet-mapping" element (if there are no "servlet-mapping" elements, put it after the "servlet" element you just added):

```
<servlet-mapping>
  <servlet-name>Controller</servlet-name>
  <url-pattern>Controller</url-pattern>
</servlet-mapping>
```

Warning: XML is case-sensitive! Make sure you enter those lines exactly as shown here. Be sure to save the file before continuing.

_12. Stop and restart the container. See Lab One if you cannot remember how.

_13. Test your application by starting your browser and entering this URL:

```
http://localhost:7001/register-m2.html
```

Enter information into the form, then press the submit button to invoke the JSP. Do you see the proper HTML page? It should be similar to the last lab. Is the amount due correct? Does the discount checkbox work properly? If not, correct any problems in the JSP source file, re-deploy, stop and restart the container and then refresh the browser.

_14. To help you understand the benefits of MVC, you should now carefully compare the **register.jsp** JSP from Lab 2 and the JSP in this lab. Which is easier to read? Which is more maintainable? Which would be easier for a non-programmer to work with?

_15. To prove the benefits, modify your application so that it displays the amount due in red (use the HTML <font color="red"> tag), then redeploy. How easy would that have been for a non-programmer?