Lab 3: Stateful Session Beans

In this lab, you will create a stateful session bean that lets a teacher enter test scores into an application. The teacher will enter scores in batches, and is interested knowing how many scores were in the batch and the average score. The batch information is not terribly important, however, so the application does not save the data persistently. Therefore, you will use a stateful session bean instead of an entity bean to keep track of the batch information.

In a real application, this stateful bean would interface with entities that would save the actual test score data persistently. However, in this lab, for simplicity, the test score records will not be saved in a database.

Here is a UML diagram:

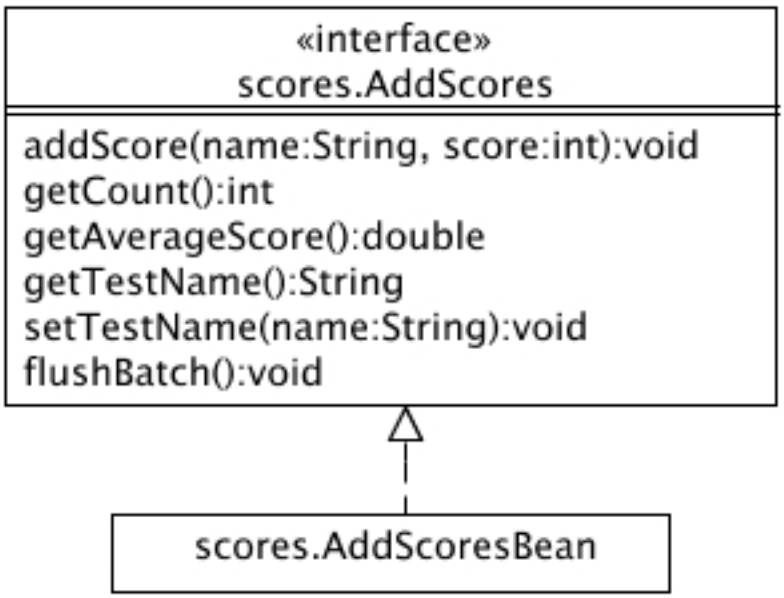

Figure 1: UML Class Diagram

The *AddScores* interface represents the client's view of the stateful session bean. It defines methods to let the user add a score to the batch, retrieve the count of batched scores, retrieve the average score for the batch and set and retrieve the test name.

The data for each test score consists of the student's name and the test score.

Objectives:

- To write stateful a session EJB
- To investigate the stateful session bean lifecycle

Part 1: Stateful Session Bean

Steps:

_1. You need to create an EJB project for your EJB. Follow these steps:

   a. Start Eclipse if it's not running.

   b. Choose **File - New EJB Project**

   c. On the first wizard page, for the *Name*, enter **lab03EJB**, then at the bottom of the page, click the *Add project to an EAR* checkbox and enter **lab03EAR**. Press Next.

   d. On the next wizard page, uncheck the *Create an EJB client JAR...* checkbox, then press Finish.

   e. You should see the new EJB and EAR projects in the Project Explorer.

_2. In the Project Explorer pane, expand the *lab03EJB/Deployment Descriptor: lab03EJB* -- you should see an empty folder for *Session Beans*.

_3. Right-click on the *Session Beans* folder and choose **New - Session Bean**. Complete the wizard:

   a. On the first wizard page, for the *EJB project*, select *lab03EJB*.

      For the *Java package*, enter **scores**.

      For the *Classname*, enter **AddScoresBean**.

      For the *State type*, select *Stateful*.

      In the *Create business interface* section, note that the local interface is selected. Then enter **scores.AddScores** for the interface name. Press Next.

   b. On the next wizard page, uncheck the *Inherited abstract method* and *Constructors from superclass* checkboxes, then press Finish.

_4. In the Project Explorer, expand *Session Beans/AddScoresBean* -- you should see two entries:

- **AddScores** represents the bean's local business interface (note the "L" in the icon)
- **AddScoresBean** represents the bean's implementation class

_5. Double-click *AddScores* to open the business interface into the Java editor. Note how the wizard inserted the *@Local* annotation, then within the interface, write the business method definitions as described in the UML diagam in Figure 1.

   Press Ctrl+S to save.

_6. Edit *AddScoresBean*. From the Eclipse menu, choose *Source - Override/Implement Methods* and select all of the methods from the *AddScores* interface.

   Eclipse writes method stub implementations. For methods that return a value, the stubs return a "placeholder" value of either zero or *null*.

_7. Now it's time to complete the EJB implementation. Follow these steps:

   a. To save the test scores, you should define a HashMap as a private field:

```
private HashMap<String, Integer> scores =
    new HashMap<String, Integer>();
```

      Use *Source - Organize Imports* to import *java.util.HashMap*.

   b. In the *addScore* method, to add a student name (key) and score, use the HashMaps's *put* method, for example:

```
scores.put(name, score);
```

c. In the *getAverageScore* method, to calculate the average score, use the HashMap's *keys* method to retrieve a list of student names and then use a *for* loop. Your code might look something like:

```
double total = 0;
Set<String> keys = scores.keySet();
for(String key : keys)
{
    int score = scores.get(key);
    total += score;
}
return total/scores.size();
```

d. In the *getCount* method, to determine the count of scores, use the HashMap's *size* method.

e. For the test name, define a private String field. Your *getTestName* method can simply return the String and the *setTestName* can accept a String and assign it to the field.

f. For the *flushBatch* method, first annotate this method with **@Remove** so that the EJB container knows it's OK to end the session and garbage collect the stateful session EJB.

If you were writing a real-world program, at this point you would write the accumulated scores to a database, but for now, complete the method by simply calling the HashMap's *clear* method.

Use *Source - Organize Imports* so that Eclipse imports the required type for the @Remove annotation.

Part 2: Using a Web Client

In this part, you will write a servlet that acts as a client to the AddScores stateful session EJB.

Steps:

_1. Choose **File - New - Dynamic Web Project**. Complete the wizard:

a. Enter **lab03Web** for the *Project name*.

At the bottom of the wizard page, put a checkmark next to the *Add project to EAR* box, then ensure that the *EAR Project Name* is **lab03EAR** (the same EAR that contains the EJB project). Press Finish.

b. In the Project Explorer, right-click on the *lab03Web* project and choose *Build Path - Configure Build Path* to start a wizard so you can set up a reference from the Web project to the EJB project.

c. On the *Java Build Path* page, click the *Projects* tab, then press the Add button and select *lab03EJB*, then press OK followed by OK to return to Eclipse.

_2. Import two fully completed JSPs via the *File System* from the **{Lab Installation directory}/starters/ejblab03**:

a. In the Project Explorer, right-click on the new project's *WebContent* folder and choose *Import...* to start the import wizard.

b. On the first wizard page, choose *General - File system*, then press Next.

c. On the next wizard page, for the *From directory*, press the Browse button and navigate to the **{Lab Installation Directory}/starters/ejblab03** folder, then press OK. If your computer has the standard lab setup, the proper directory is:

```
c:\j2eeclass\starters\ejblab03
```

In the left pane, click on the *ejblab03* to highlight it, then put a checkmark next to *index.jsp* and *addScore.jsp* entries in the right pane as shown in Figure 2:

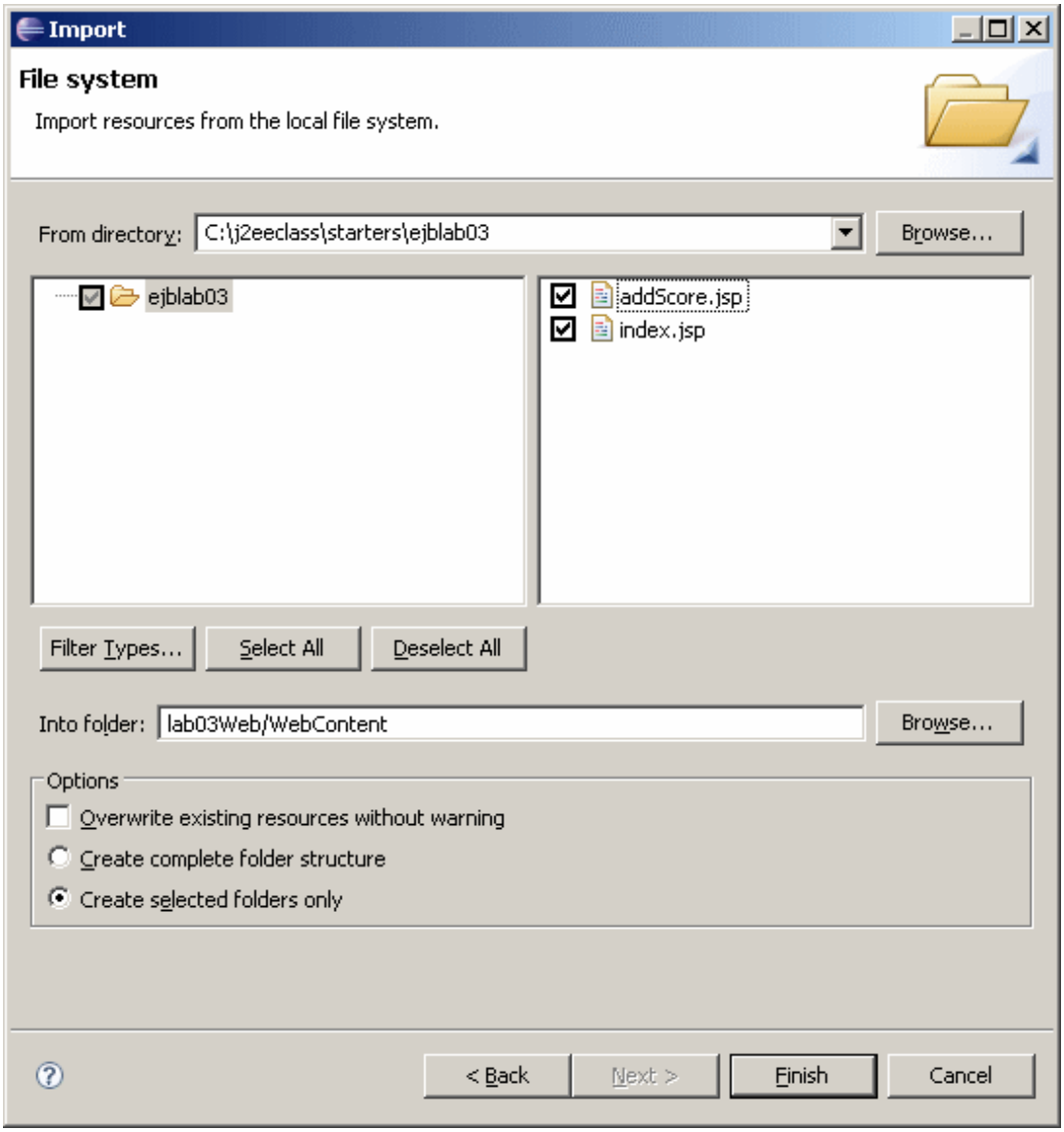

Figure 2: Import Wizard

Press Finish to complete the import. You should see the imported files in the project's WebContent folder.

d. Open *index.jsp* into the editor and note how it defines an HTML form into which the user can enter

a "test name". When submitted, the form invokes a servlet named *CreateBatchServlet* that you will write in a moment.

e. Open *addScore.jsp* into the editor and note how it defines an HTML form into which the user can enter a "student name" and a "test score". This form invokes a servlet named *AddScoreServlet* that you will write in a moment.

Note also how the form has three "submit" buttons:

- An "Add Score" button, that when pressed, adds the current test score to the batch.
- A "New Batch" button, that when pressed, finishes the current batch and starts a new one.
- A "Done" button, that when pressed, terminates the current batch and sends the user back to the home page.

Note: You can ignore any errors in Eclipse in addScore.jsp complaining about a "tag library descriptor".

_3. Next, create the Java class for the CreateBatchServlet. Follow these steps:

a. In the Project Explorer, in the *lab03Web* project, right-click on *Java Resources: src* and choose *New - Servlet* to start the wizard.

b. On the first wizard page, for the *Java package* enter **servlets**.

For the *Class name*, enter **CreateBatchServlet**, then press Next.

c. On the next page, note the *URL Mapping*, which assigns a URL to the servlet, then press Next.

d. On the next wizard page, uncheck the *doPost* checbox, then press Finish. Eclipse opens the servlet class into the editor.

_4. Your next job is to complete the CreateBatchServlet.

This servlet will create an instance of the AddScores stateful session EJB you wrote in the last part. Unfortunately, we cannot use dependency injection to retrieve the reference, since the application needs a new AddScores instance every time the user navigates to the index.jsp home page.

So instead of using dependency injection, the servlet will perform a JNDI lookup. The servlet will then store the EJB's reference in the HTTP session, and transfer control to the "Add Scores" page. Follow these steps to complete the servlet:

a. In the servlet's *doGet* method, write code to create an initial JNDI context and retrieve a reference to the stateful EJB, thus creating an instance:

```
Context ctx = new InitialContext();
AddScores batch = (AddScores)ctx.lookup(
    "lab03EAR/AddScoresBean/local");
```

Use *Source - Organize Imports* to import the proper types.

Note: Ignore errors regarding unhandled exceptions for the time being.

b. Retrieve the test name that the user entered into the HTML form and set it into the EJB:

```
String testName = request.getParameter("testName");
batch.setTestName(testName);
```

c. Create an HTTP session, store the EJB reference in it, then transfer to the addScores.jsp:

```
HttpSession session = request.getSession();
session.setAttribute("batch", batch);

response.sendRedirect("addScore.jsp");
```

Use *Source* - *Organize Imports* to import the proper types.

d. Highlight all of the lines within the doGet method and use *Source* - *Surround With* - *Try/catch Block* so that Eclipse writes simple exception handling.

Save to compile - you should have no errors.

_5. In the same fashion as in an earlier step, create a new servlet named **AddScoresServlet** in the *servlets* package. The new servlet should override only *doGet*.

_6. This new servlet has several jobs. To get started, write code to respond to the "Add Score" button:

a. In the servlet's *doGet* method, first attempt to retrieve the EJB reference from the HTTP session. If it's not there, perhaps because the user skipped the home page, send them back to the home page:

```
HttpSession session = request.getSession();
AddScores batch = (AddScores)session.getAttribute("batch");
if (batch != null)
{

}
else
    response.sendRedirect("index.jsp");
```

b. Inside the "if" statement, write a nested "if" to determine which button the user pressed:

```
String action = request.getParameter("action");
if (action.equals("Add Score"))
{

}
```

c. Within the "if", retrieve the data that the user entered and set it into the stateful EJB. Then send the user back to the HTML form so they can enter another score:

```
String studentName = request.getParameter("studentName");
String score = request.getParameter("score");
batch.addScore(studentName, Integer.parseInt(score));
response.sendRedirect("addScore.jsp");
```

d. After the closing brace of the nested "if", write an "else if" to determine if the user pressed the "New Batch" button:

```
else if (action.equals("New Batch"))
{

}
```

e. Within the "else if", retrieve the current EJB's name, then call its *flushBatch* method. Recall that you annotated flushBatch with @Remove -- when you call this method, the container gets rid of the EJB.

```
String testName = batch.getTestName();
batch.flushBatch();
```

Then use JNDI to retrieve another EJB reference, which creates a new stateful EJB. Note that you can copy and paste from the CreateBatchServlet to save typing:

```
Context ctx = new InitialContext();
batch = (AddScores)ctx.lookup("lab03EAR/AddScoresBean/local");
```

Set the new EJB's name to the test name, then store its reference in the HTTP session and send the user back to the addScores.jsp:

```
batch.setTestName(testName);
session.setAttribute("batch", batch);
response.sendRedirect("addScore.jsp");
```

Highlight all of the lines within the "else if" block and use *Source - Surround With - Try/catch Block*.

f. After the closing brace of the "else if", determine if the user pressed the "Done" button. If so, flush the batch and send the user back to the home page:

```
else if (action.equals("Done"))
{
    batch.flushBatch();
    response.sendRedirect("index.jsp");
}
```

g. Use *Source - Organize Imports* as necessary and save to compile. Ensure that you have no errors.

_7. To test your program, right-click on index.jsp and choose *Run As - Run on Server*. You should be able to create a batch, add scores, flush the batch and return back to the home page.

Optional Part 3: Standalone Java Client

In this part, you will write a standalone Java client program for the AddScores EJB.

Steps:

_1. Using the same techniques as you did in the last lab, create a new Java project named **lab03Client**.

_2. As in the last lab, create (or copy) a *jndi.properties* file in the project's root directory. The properties file should contain connection parameters for the server.

_3. Create a Java class named **Client** in the *client* package with a *main* method.

Complete *main* so it:

- Creates an initial JNDI context
- Retrieves the business interface reference via JNDI
- Sets the test name
- Adds a few student-name / score pairs
- Displays the count and average score to the console
- Displays the test name by calling the *getTestName* method
- Flushes the batch

_4. Test your client as before.