Lab 7: Spring and JDBC

In this lab, you will write an application that uses Spring's SimpleJdbcTemplate support for writing data access objects (DAO).

Objectives:

- To access relational data using Spring SimpleJdbcTemplate

Part 1: Setting Up the Environment

In this part, you will configure an H2 database that your Spring program will work with.

Steps:

_1. If you did NOT do Lab06 Spring MVC, then please execute the steps in Lab06 Part 1 to configure Tomcat.

_2. Next, you need to set up Tomcat so it can access the database you will later create:

   a. Stop the Tomcat server if it's running.

      Using Windows Explorer or My Computer, copy the **{Lab Installation Directory}/lib/h2/h2-1.3.161.jar** file to the **{Tomcat Installation Directory}/lib** folder.

_3. Next, you need to start the H2 database server and its adminstrative GUI, which is Web-based. Here are the instructions for Windows Vista or later. **SKIP THIS IF YOUR COMPUTER HAS WINDOWS XP**.

   a. The database server runs best in an Adminstrative command prompt. To open the command prompt, click the Windows Start menu and enter **cmd** into the Search box, to find cmd.exe, but don't press Enter.

      Right-click on cmd.exe and choose *Run As Adminstrator* to start the privileged command prompt.

   b. In the command prompt, change to **{Lab Installation Directory}/lib/h2**. If your computer has the standard lab setup, enter something like:

      ```
      cd \Users\username\springclass\lib\h2
      ```

      Note: You will need to substitute your actual user name.

   c. To start the H2 database server and admin GUI, enter:

      ```
      java -jar h2-1.3.161.jar
      ```

      The server runs invisibly in the command prompt, and launches the Web-based GUI into your browser.

_4. Here are the instructions to start the H2 database if your computer has Windows XP. **SKIP THIS STEP IF YOUR COMPUTER HAS WINDOWS VISTA OR LATER AND YOU DID THE PREVIOUS STEP**.

   Using Windows Explorer or My Computer, navigate to **{Lab Installation Directory}/lib/h2/h2-1.3.161.jar** and double-click on the JAR - this starts the database running and opens a Web-based administration GUI.

_5. In the H2 GUI, set the *JDBC URL* to **jdbc:h2:tcp://localhost/~/test**, typing this VERY carefully.

   Set both the *User Name* and *Password* to **sa**, then press Connect.

_6. Enter the following into the large entry box and then press the Run (Ctrl+Enter) button to create and populate your database (you can copy and paste from a starters file in **{Lab Installation Directory}/starters/spring/lab11}**):

```
CREATE TABLE Segment
(
   SegmentNumber   INTEGER NOT NULL IDENTITY,
   SegmentDate     DATE,
   FlightNumber    INTEGER,
   OrigCity        VARCHAR(10),
   Miles           INTEGER,
   CONSTRAINT  PK_Item  PRIMARY KEY(SegmentNumber)
);


INSERT INTO Segment VALUES (NULL, '2009-10-18', 333, 'SFO', 367);
INSERT INTO Segment VALUES (NULL, '2009-10-18', 745, 'LAX', 1900);
INSERT INTO Segment VALUES (NULL, '2009-10-22', 453, 'BOS', 1900);
INSERT INTO Segment VALUES (NULL, '2009-10-18', 112, 'LAX', 367);
```

Press the Clear button then enter and run:

```
SELECT * FROM segment
```

You should see four segment rows.

Part 2: Setting Up the Web Project

In this part, you will create a servlet-based Web application that will act as the user interface for a Spring application that queries relational data using Spring's SimpleJdbcTemplate.

Steps:

_1. This lab depends on successfully completing the basic parts of *lab01* - it doesn't depend on the "experiments". If you did not finish the basic part of lab01, you should either finish it or ask the instructor to help you get that lab's solution.

_2. If necessary, switch to the JEE perspective by choosing *Window - Open Perspective - Other - Java EE (default)*.

_3. Eclipse uses *Dynamic Web Applications* for projects that use servlets and JSPs, so you will start by creating such a project:

a. Choose *File - New - Dynamic Web Project* to start the wizard.

b. On the first wizard page, for the *Project name*, enter **springlab07Web**, then press Next.

c. On the next wizard page, press Next.

d. On the last wizard page, put a checkmark in the *Generate web.xml deployment descriptor* box, then press Finish.

e. In the Project Explorer, expand the new project and note the following:

- There is a *Java Resources/src* folder in which you will put your Java code including servlets, Spring bean classes and so forth.

- There is a *WebContent* folder in which you will put any HTML files, JSPs and so forth. There is a subfolder of WebContent named *WEB-INF* that contains the standard JEE Web deployment descriptor (web.xml) and other files.

_4. Next, copy files from the *lab01* project to the new project:

a. In the Project Explorer pane, expand the *springlab01* project's *src* folder, then right-click on the *com.oaktreeair.ffprogram* package and choose *Copy*.

b. In the Project Explorer, expand the *springLab07Web* folder, then right-click on the *src* folder and choose *Paste* to copy the Spring bean classes to the new project.

You will have compile errors that you will fix in a moment.

c. Repeat the above step to copy the *spring.xml* file from the *springlab01* project, copying into the *Lab07Web* project's *WebContent/WEB-INF* folder.

_5. Next, configure the project for Spring:

a. Open Windows Explorer or My Computer, then resize your windows so you can see both Eclipse and Windows Explorer.

In Windows Explorer, navigate to **{Lab Installation Directory}/lib/spring3.1.4**. If your computer has the standard lab setup on Windows XP, this directory is:

```
C:\springclass\lib\spring3.1.4
```

On Windows Vista or later:

```
C:\Users\username\springclass\lib\spring3.1.4
```

b. In Windows Explorer, highlight all of the JARs then drag and drop them to the Eclipse Project Explorer onto the *WebContent/WEB-INF/lib* folder.

c. In the Project Explorer, double-click on the *WEB-INF/web.xml* file to open it into the Web deployment descriptor editor.

Click the *Source* tab at the bottom of the editor window, then copy and paste the following text from the **{Lab Installation Directory}/starters/lab06/listener.txt** above the *welcome-file-list* start tag:

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring.xml</param-value>
</context-param>
<listener>
    <display-name>ContextLoaderListener</display-name>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
```

This configures the Web application so that it can access a Spring application context.

Save and close the deployment descriptor.

_6. Next, you will configure a data source that represents the database. Follow these steps:

a. Create a file named **context.xml** in the *WebContent/META-INF* folder.

b. Complete the file, copying from **{Lab Installation Directory/spring/starters/lab07/context.txt** so it looks like:

```
<Context path="/springlab07Web" docBase="springlab07Web" debug="0">
  <Resource name="jdbc/flier"
            auth="Container"
            type="javax.sql.DataSource"
            username="sa" password="sa"
            driverClassName="org.h2.Driver"
            url="jdbc:h2:tcp://localhost/~/test"/>
</Context>
```

This defines a data source with JNDI name *jdbc/flier*.

_7. Next, create a *resource reference* for the data source in the Web project:

a. In the Project Explorer, double-click on the *WebContent/WEB-INF/web.xml* file to open it into the deployment descriptor editor.

b. After the *listener* end-tag, define the resource reference, copying from **{Lab Installation Directory}/starters/lab07/resource.txt**:

```
<resource-ref>
    <res-ref-name>jdbc/flier</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

c. Save and close the Web deployment descriptor.

Part 3: Spring JdbcTemplate DAO

In this part, you will create a data access object (DAO) class using the Spring JdbcTemplate type.

Steps:

_1. Create an interface for your DAO:

a. In the Project Explorer, right-click on the *Java Resources: src* folder and choose *New - Interface* and create an interface named **SegmentDao** in a package named **com.oaktreeair.ffprogram.dao**.

b. Add the following methods to the interface:

```
public int getSegmentCount();
public Collection<Segment> findAllSegments();
public int insertSegment(Segment s);
```

You will need to import *java.util.Collection* and *com.oaktreeair.ffprogram.Segment*.

_2. Create the DAO implementation class:

a. In the Project Explorer, right-click on the *JavaResources: src/com.oaktreeair.ffprogram.dao* package and choose *New - Class* to start the wizard.

b. On the first wizard page, ensure that the *Package* is **com.oaktreeair.ffprogram.dao**.

For the *Name*, enter **SegmentDaoImpl**.

For the *Interfaces*, press the Add button, and in the *Choose interfaces* box, start typing **Seg**, then select *SegmentDao*.

Press OK followed by Finish to complete the wizard. Eclipse opens the new class into the editor.

_3. Annotate the DAO as a repository component and give it a name:

```
@Repository("segmentDao")
```

Import the correct type. Note the Spring ID of the DAO is *segmentDao*. You will need that ID when you later write servlets.

_4. Let's start by using dependency injection to obtain the resources the DAO needs:

a. Define a field to hold the JDBC template:

```
private JdbcTemplate template;
```

Import the correct type.

b. Inject a data-source reference and create the template:

```
@Autowired
public void setDataSource(DataSource ds)
{
    template = new JdbcTemplate(ds);
}
```

Import the javax.sql.DataSource type.

_5. Let's start by implementing only the *getSegmentCount* method:

a. Execute a SQL command via the template to retrieve the count of rows in the Segment table:

```
int count = template.queryForInt(
        "SELECT COUNT(*) FROM Segment");
```

b. Modify the *return* statement to return the count.

Part 4: Web Front End for the DAO

In this part, you will configure the DAO with Spring and write a simple servlet to invoke its *getSegmentCount* method.

Steps:

_1. Configure the data source in the Spring configuration file:

a. Open the *WebContent/WEB-INF/spring.xml* Spring configuration file into the editor. Press the *Source* tab at the bottom of the editor window.

b. If necessary, enter the following text to ensure that Spring scans to find your classes configured with @Component and/or @Repository annotations:

```
<context:component-scan
  base-package="com.oaktreeair.ffprogram"/>
```

c. Enter the following text to configure the data source and the DAO itself:

```
<jee:jndi-lookup id="flierDataSource"
   jndi-name="jdbc/flier" resource-ref="true"/>
```

Note how the JNDI name matches the context.xml configuration file you created earlier in the lab.

_2. Next, create a simple servlet that will access the segment count via the DAO:

a. In the Project Explorer, right-click on the *springlab07Web* project and choose *New* - *Servlet* to start the wizard.

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

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

c. On the next page, examine the defaults, then press Next.

d. On the final page, in the "Which method stubs" section, uncheck the *doPost* box, then press Finish.

Eclipse opens the new servlet into the Java editor.

_3. Complete the new servlet's *doGet* method:

a. Retrieve a reference to the Spring application context:

```
ServletContext servletContext = getServletContext();
WebApplicationContext ctx =
    WebApplicationContextUtils.getRequiredWebApplicationContext(
        servletContext);
```

Choose *Source* - *Organize Imports* so that Eclipse imports the required types.

b. Retrieve a reference to the DAO from the application context in the normal fashion:

```
SegmentDao dao = . . .;
```

c. Initialize the servlet's HTML output stream:

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
```

d. Output simple HTML content containing the segment count:

```
out.println("<html><body>");
out.println("<p>Segment count: " +
   dao.getSegmentCount() + "</p>");
out.println("</body></html>");
```

e. Import types as necessary, then save the servlet.

_4. To test, in the Project Explorer, right-click on *DisplaySegmentCount.java* and choose *Run As - Run on Server*, select Tomcat, then press Finish. Wait for the server to start - you should see a Web page with the segment count (4).

Part 5: Completing the Application

In this part, you will complete the DAO and write servlets as the user interface.

Steps:

_1. You will start by implementing the DAO's *findAllSegments* method.

First, create a *nested inner class* that knows how to create Segment objects from rows in the Segment database table:

a. In the Project Explorer, right-click on *JavaResources: src/com.oaktreeair.ffprogram.dao/ SegmentDaoImpl* and choose *New - Class* to start the wizard.

b. On the first wizard page, ensure that the *Package* is *com.oaktreeair.ffprogram.dao*.

Put a checkmark in the *Enclosing type* checkbox - this causes the wizard to generate an inner class.

For the *Name*, enter **SegmentRowMapper**.

For *Interfaces*, press the Add button and start typing **ParameterizedR** and then select *ParameterizedRowMapper* and press OK followed by Finish to complete the wizard.

_2. The wizard gave you a head start, but you need to tweak the inner class a bit:

a. Edit the inner class definition so it looks like:

```
public class SegmentRowMapper
  implements ParameterizedRowMapper<Segment>
```

b. Inside the nested class, add the *mapRows* method definition:

```
public Segment mapRow(ResultSet rs, int rowNum)
   throws SQLException
{
}
```

_3. Complete the SegmentRowMapper inner class's *mapRow* method:

a. Create a new Segment object using its zero-argument constructor.

b. Call each of the "set" methods on the Segment object, initializing with data from the result set. For example, to set the Segment's *flightNumber* property:

```
seg.setFlightNumber(rs.getInt("FlightNumber"));
```

Be sure to call ALL of the "set" methods. For your convenience, here is the Segment table schema:

```
CREATE TABLE Segment
(
   SegmentNumber   INTEGER,
   SegmentDate     DATE,
   FlightNumber    INTEGER,
   OrigCity        VARCHAR(10),
   Miles           INTEGER
)
```

Note that you will need to map SQL types to their equivalent Java types by calling the appropriate getXXXX() method on the result set.

c. Return the fully initialized Segment object.

_4. Update the DAO class to create a row mapper object:

a. In the SegmentDaoImpl class (NOT the SegmentRowMapper inner class), define a field to hold an instance of the SegmentRowMapper class:

```
private SegmentRowMapper mapper;
```

b. Modify the *setDataSource()* method so that it creates an instance of the SegmentRowMapper class using the zero-argument constructor, storing the reference in the field you just defined.

_5. Now complete the *findAllSegments* method:

a. Call the template's *query* method to return a list of Segments, using the mapper to convert from result set to Segment object:

```
List<Segment> segments = template.query(
        "SELECT * FROM Segment", mapper);
```

b. Modify the *return* statement to return the list.

_6. In the same fashion as earlier, create a new servlet named **DisplayAllSegments** whose doGet() method uses the DAO to retrieve the Segment list and then displays them. For extra credit (!), display the list in an HTML table.

_7. Run and test.

_8. If you have time, implement the DAO's *insertSegment* method. Then write an HTML input form to let the user enter all of the Segment data except the segmentNumber (that's an auto-generated column in the database). The HTML form's Submit button should invoke a new servlet that retrieve the HTML form data and calls the DAO's *insertSegment* method to insert the new segment into the database.

_9. In the Servers tab, right-click on the Tomcat server and remove the springlab07Web project from the server, then right-click again and stop the server.