**ISCG7425 – Enterprise Java Programming**

**Lab Session 3**

Struts Framework + Hibernate

1. Download and setup the Struts + Hibernate example project provided on Moodle. Alter the session factory settings in the hibernate.cfg.xml file located in the resources directory to reflect your database and authentication settings.

If you are using hsqldb then hibernate.cfg.xml will already be setup for you.

Note: you will need to drop your existing database table as the configuration will differ from what Hibernate creates automatically.

|  |
| --- |
| <?xml version=*'1.0'* encoding=*'utf-8'*?>  <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">  <hibernate-configuration>  <session-factory>  <!-- Database connection settings -->  <property name=*"connection.driver\_class"*>org.hsqldb.jdbcDriver</property>  <property name=*"connection.url"*>jdbc:hsqldb:hsql://localhost</property>  <property name=*"connection.username"*>sa</property>  <property name=*"connection.password"*></property>    <!-- JDBC connection pool (use the built-in) -->  <property name=*"connection.pool\_size"*>10</property>    <!-- SQL dialect -->  <property name=*"dialect"*>org.hibernate.dialect.HSQLDialect</property>    <!-- Enable Hibernate's automatic session context management -->  <property name=*"current\_session\_context\_class"*>thread</property>    <!-- Disable the second-level cache -->  <property name=*"cache.provider\_class"*>org.hibernate.cache.NoCacheProvider</property>    <!-- Echo all executed SQL to stdout -->  <property name=*"show\_sql"*>true</property>    <!-- Drop and re-create the database schema on startup -->  <property name=*"hbm2ddl.auto"*>update</property>  <mapping class=*"com.jcasey.model.Book"* />  </session-factory>  </hibernate-configuration> |

1. Have a look at the model object class Book located in the com.jcasey.model package. Note the usage of annotations to setup the Hibernate database bindings ie:

* @Entity
* @Table(name="Book")
* @Id
* @GeneratedValue
* @Column(name="book\_id")

1. Have a look at the model object class Book located in the com.jcasey.model package. Note the use of annotations to setup the Hibernate database bindings ie:

* @Entity
* @Table(name="Book")
* @Id
* @GeneratedValue
* @Column(name="book\_id")

1. Develop the BookManager classes list method. The list method operates at the persistence layer, and will talk to the database via Hibernate. The list method will take three parameters: genre, title and author and programmatically add Restrictions for different criteria to Hibernate Criteria object. If a user does not fill in any fields then the list method should return all results.

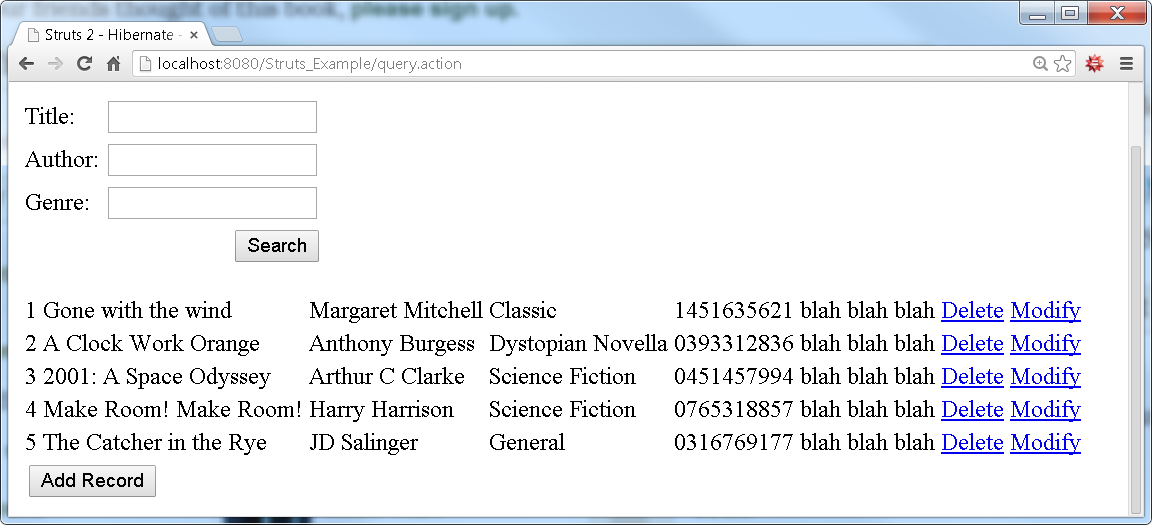
At a higher level the list(genre, title, author) method will be called by the middle tier: action object SearchBook and the method query(). Use the links below to get more information on creating Hibernate Criteria objects.

* <http://docs.jboss.org/hibernate/orm/3.5/api/>
* <http://docs.jboss.org/hibernate/orm/3.5/api/org/hibernate/Criteria.html>
* <http://docs.jboss.org/hibernate/orm/3.5/api/org/hibernate/Query.html>
* <http://docs.jboss.org/hibernate/orm/3.5/api/org/hibernate/criterion/Restrictions.html>

1. Update the list() method code so that your Restrictions criteria are fuzzy and make use of the like() method.
2. Wire the BookManager up to your SearchBook Struts Action POJO and use the list() method to query the book table based on the genre, title and author criteria.

Hold onto a reference of the returned LinkedList of books and use this list to drive the s:iterator table as shown below. As the Hibernate Book class is simply just a POJO class with annotations the Struts framework will be able to automatically work with this new class.

Note: in the sample code provided this should already be done. Make sure understand how Hibernate returns results, and how the results are bubbled up to the user interface by the middle tier, the presentation layer and how everything is configured in the struts.xml file.



1. Create a new action named add in your struts.xml file.

|  |
| --- |
| <action name=*"add"* class=*"com.jcasey.action.ModifyBook"* method=*"execute"*>  <result name=*"add"*>add.jsp</result>  </action> |

The add action will reference the pre-existing execute method on the pre-defined class ModifyBook. This method checks if a book exists already. If a book already exists then the method will return a result of “update”. The update function will be implemented next week. If the book does not exist the function will return “add” which will cause Struts to display the “add.jsp” page which allows a user to enter a new book.

To allow a user to add a new book add a new Struts s:form to the main content page “query.jsp” use a code block similar to the following and note how the submit button is linked to the add action.

|  |
| --- |
| <s:form action=*"add.action"* method=*"post"*>  <s:iterator value=*"books"*>  <tr>  <td>  <s:property value=*"bookId"*/>  </td>  <td>  <s:property value=*"title"*/>  </td>  <td>  <s:property value=*"author"*/>  </td>  <td>  <s:property value=*"genre"*/>  </td>  <td>  <s:property value=*"isbn"*/>  </td>  <td>  <s:property value=*"blurb"*/>  </td>  </tr>  </s:iterator>  <s:submit name=*"add"* method=*"execute"* label=*"Add Record"*  align=*"left"* type=*"button"*/>  </s:form> |

Update the file “add.jsp” to allow a user to enter values for all the fields required by a book record using the skeleton code outlined below:

|  |
| --- |
| <%@ page contentType=*"text/html; charset=UTF-8"*%>  <%@ taglib prefix=*"s"* uri=*"/struts-tags"*%>  <html>  <head>  <title>Struts 2 - Book Database</title>  </head>  <body>  <h2>Struts 2 - Book Database</h2>  <s:actionerror />  <s:form action=*"save.action"* method=*"post"*>  <s:textfield name=*"bookId"* label=*"Id"* size=*"20"* readonly=*"true"* />  <s:submit name=*"Add"* label=*"Add"* align=*"right"* type=*"button"*/>  </s:form>  </body>  </html> |

Finally, create a new struts action named “save.action” to save user data to the database using the method add() on the class ModifyBook. The action will load the query.action page once the system saves the record to the database.

Use the following state transition diagram to understand how everything fits together.

