SENG2050 - Lab 05

Beans and MVC

Please finish Lab 04 before you start this work!

This tutorial is designed to help you to extend your Java Server Pages (JSP) and Java knowledge by creating more complex JSPs. We will also introduce sessions and run through an example of how to use them. We will learn how to link our Sevlets, JSPs and Beans together to create MVC Applications.

The tasks in this tutorial will focus on creating a basic online shopping cart. This will utilise sessions to allow each user to have their own list of items they wish to purchase. We will use controllers to manage user interactions, beans to implement domain data and business logic, and JSPs to generate pages.

Create a webapps/lab05/ directory to complete these tasks.

1 Part A - Sessions in an Online Store

A session is a place to store data unique to a single user of the application. Each user can have their own session. In this tutorial we will use the session to create a cart bean object.

1.1 Task 1 - A Cart Bean

Below is the code of **CartBean.java**. This is how the web application stores all the information about the user's shopping (the products the user wants to buy, and their quantity).

```
1  package pkg;
2
3  import java.io.Serializable;
4  import java.util.*;
5
6  public class CartBean implements Serializable {
7
8      private Map<String, Integer> itemMap;
9
10     public CartBean() {
11         this.itemMap = new HashMap<>();
12     }
13
14     public synchronized void addItem(String name, Integer qty) {
15         Integer existingQty = this.itemMap.getOrDefault(name, 0);
```

```
16          this.itemMap.put(name, qty + existingQty);
17      }
18
19      public Set<String> getItemNames() {
20          return this.itemMap.keySet();
21      }
22
23      public int getQuantity(String name) {
24          return this.itemMap.getOrDefault(name, 0);
25      }
26
27      public boolean isEmpty() {
28          return this.itemMap.isEmpty();
29      }
30  }
```

Save this class into /classes/pkg/CartBean.java, and then compile this java class. We will use Cart as a Java Bean.

1.2 Task 2 - A Cart Page

Every online store needs to let users view the items in their cart. Start by creating a new file cart.jsp. We can display the currently stored items as so:

```
1  <jsp:useBean id="cart" class="pkg.CartBean" scope="session" />
2
3  <!DOCTYPE html>
4  <html>
5      <head>
6          <title>My Cart</title>
7      </head>
8      <body>
9          <h1>My Cart</h1>
10     <a href="/lab05/store.jsp">Back to Store</a>
11         <% if (cart.isEmpty()) { %>
12             <p>There are no items in the cart. Start shopping!</p>
13         <% } else { %>
14             <ul>
15             <% for (String item : cart.getItemNames()) { %>
16                 <li><%= item %> - <%= cart.getQuantity(item) %></li>
17             <% } %>
18             </ul>
19         <% } %>
20     </body>
21 </html>
```

The most important part of this page is the jsp:useBean tag. This directive tells the page we have access to a variable named 'cart' of type 'pkg.CartBean'. If the JSP page finds that this

variable has not been created, it will automatically create a new instance of CartBean.

Start by navigating to this page in the browser (http://localhost:8080/lab05/cart.jsp). On first access, we will have no items in our cart. To add items, we need to create a new store page!

1.3 Task 3 - A Store Page

We can create a very simple store with a HTML form. The form will accept two inputs:

1. The name of an item (presented with a SELECT dropdown).
2. The quantity of said item the user wants to buy.

Create a new file store.jsp. Add the following contents:

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>The Goode Store</title>
5      </head>
6      <body>
7          <h1>The Goode Store</h1>
8          <a href="/lab05/cart.jsp">View Cart</a>
9          <form action="/lab05/additem" method="POST">
10             <table>
11                 <tr>
12                     <td>Item:</td>
13                     <td>
14                         <select name="item" required>
15                             <option disabled>Please select an item ...</option>
16                             <option>Apples</option>
17                             <option>Bananas</option>
18                             <option>Coffee</option>
19                             <option>Biscuits</option>
20                             <option>Milk</option>
21                         </select>
22                     </td>
23                 </tr>
24                 <tr>
25                     <td>Quantity:</td>
26                     <td>
27                         <input type="number" name="qty" required />
28                     </td>
29                 </tr>
30                 <tr>
31                     <td colspan="2"><input type="submit" value="Add to Cart" /></td>
32                 </tr>
33             </table>
34         </form>
```

```
35          <% if (request.getParameter("success") != null) { %>
36              <p><em>Item added to cart</em></p>
37          <% } else if (request.getParameter("error") != null) { %>
38              <p>Error, <em><%= request.getParameter("error") %></em></p>
39          <% } %>
40      </body>
41  </html>
```

1.4 Task 4 - Adding Items to the Cart

To add items to the cart, we need to create a simple Servlet. A Servlet is the best place for this as Servlets are designed for processing requests, and changing state in the application. In this case, it will be retreiving the inputs from the form, and modifying our CartBean object.

Our CartBean is a session-scoped bean. This means it 'lives' in a HttpSession object, that is bound to a single user. Create a new Servlet 'AddItemServlet.java', map it to the url 'additem', and add the following doPost method:

```
1  @Override
2  protected void doPost(HttpServletRequest request, HttpServletResponse response)
3    throws IOException {
4    HttpSession session = request.getSession();
5
6    // Get the cart bean from the session
7    CartBean cart = (CartBean) session.getAttribute("cart");
8    if (cart == null) {
9      cart = new CartBean();
10     session.setAttribute("cart", cart);
11   }
12
13   // Retrieve + validate request inputs
14   String item = request.getParameter("item");
15   String qtyStr = request.getParameter("qty");
16   int qty = 0;
17
18   String errors = "";
19
20   if (item == null || qtyStr == null) {
21     errors += "item or qty are null\n";
22   }
23
24   try {
25     qty = Integer.parseInt(qtyStr);
26   } catch (NumberFormatException ex) {
27     errors += "qty is not a number\n";
28   }
29
```

```
30    // Handle error
31    if (errors.length() > 0) {
32      response.sendRedirect("/lab05/store.jsp?error=" + errors);
33      return;
34    }
35
36    // Add item to cart
37    cart.addItem(item, qty);
38    response.sendRedirect("/lab05/store.jsp?success=true");
39  }
```

Compile this servlet and run the application. Visit store.jsp, and add some items to the cart. Navigate to the cart.jsp page. You will be presented with the items currently stored in the users cart.

1.5 Task 5 - Sessions

A session is bound to an individual user. Technically, the user is identified by the browser. This means, for every browser session, the human user can have their own list of items in the cart.

- Try opening a new private/igcognito browser window. Navigate to cart.jsp. Are your items still there?
- Add some items in the private broweser window. Switch back to cart.jsp in your original browser window. Are the new items there?
- The two browser windows will have a different list of items. This is because they are different sessions!

Question: Which activities are considered to be part of one session? Which activities are not? Make sure you understand the differences.

2 Part B - A Simple Quiz Game

To continue our use of sessions, we will create a simple quiz game.

2.1 Task 6 - A Quiz

Create a file called **quiz.jsp**. On this .jsp create a form which should include:

- Four questions and four answers. These input answers should be named 'answer1', 'answer2', 'answer3' and 'answer4'.
- A submit button which submits to **quizResults.jsp**

Create a new class to capture this form data called QuizBean.java. Save it into WEB-INF/classes/pkg. Add four fields (corresponding to answer 1 to 4), with appropriate getters and setters.

Create a fifth method **getScore()**. Your **getScore()** method should check the answers the user submits with the correct answers and return a score out of 4 (as an integer). Note: this method does not need a backing field. It will simply compare the provided answers with the correct answers (your set the correct answers) and return the users number of correct answers.

On your quizResults.jsp page add the following bean declaration and setters the top of your JSP file:

```
<jsp:useBean id="quiz" class="pkg.QuizBean" />
<jsp:setProperty name="quiz" property="answer1" param="answer1" />
<jsp:setProperty name="quiz" property="answer2" param="answer2" />
<jsp:setProperty name="quiz" property="answer3" param="answer3" />
<jsp:setProperty name="quiz" property="answer4" param="answer4" />
```

Within the html body tag include the following:

```
<p>
Your score is: <jsp:getProperty name="quiz" property="score" />
</p>
```

Now test your mini quiz page. Navigate to localhost:8080/lab05/quiz.jsp. Answer the questions and view your score.

2.2 Task 7 - Re-doing the quiz

Now we are going to use the quiz bean on both pages.

1. Copy over the **jsp:useBean** tag to quiz.jsp.
2. To make the same Bean accessible to both pages, specify **scope=”session”** in both **jsp:useBean** tags.
3. In the text fields of quiz.jsp, print out the current value of each answer. This will provide the effect of 'pre-filing' the answers. e.g.

   ```
   <input type="text" name="answer1" value="<jsp:getProperty name="quiz" property="answer1" />" />
   ```

   or

   ```
   <input type="text" name="answer1" value="<%= quiz.getAnswer1() %>" />
   ```

4. Create a link on your results page that allows the user to take the quiz again.

When the user goes to the quiz page they will see the answers they previously entered by setting the default values of the inputs from the current values stored in the Bean.

You will have to give each property of your Bean a default value (e.g. an empty string), which will become the initial default value for the corresponding input.

Check that you understand why this is happening. Confirm with your demonstrator.

3 Full MVC Designs

The two applications we created here are examples of MVC designs. However, both examples largely follow simpler 'controller-less' MVC designs. The AddItem Servlet from Task 4 is the closest action to a full MVC design pattern. However, it itself is simple, and does not have enough complexity to demonstrate full MVC.

When we have a full MVC designed application, we never directly access JSPs. We always access our servlets, who will process the request, setup the Java beans, and forward generating the page to a JSP. However, this requires more complex interactions than we have seen here. For full MVC interactions, we need applications that need to manage state and implement validation.

This section will demonstrate how to setup for a full-MVC user interaction. We will have a Servlet that processes requests and populates beans, that subsequently forwards to a JSP to display a HTML page. To forward a request to a page, we use the "RequestDispatcher" object (found in the "javax.servlet" package) like so:

```
RequestDispatcher dispatcher = getServletContext()
        .getRequestDispatcher("/WEB-INF/presenting.jsp");
dispatcher.forward(request,response);
```

Where "WEB-INF/presenting.jsp" is a JSP that will read the processed data, present it, and send it to the client. The important bit here is the location of the JSP. It is in the WEB-INF directory, which cannot be externally accessed. In a full MVC application, we assume a JSP needs some setup before display. Hence, we need to hide them to make a user cannot directly access them. Note: Sometimes we store JSPs in a sub-directory named 'jsps' (i.e. /WEB-INF/jsps). This is simply for organisation.

3.1 Task 8 - MVC Applications

To demonstrate MVC, we are going to re-implement our store using full MVC. This will require creating a new Servlet, Bean and JSP file.

1. Start by creating a new Servlet named 'StoreController'. Map it to the URI '/store'.
2. Create a new Java bean called 'StoreBean'. This bean will store all of the types of items sold.
3. Create a new JSP file called 'storepage.jsp', and save it in the WEB-INF directory.

The doGet method of StoreController will display the front page of the store. It will present the form used to add items, an error/success message when adding items to the cart, as well as the users current cart (for simplicity). Create the doGet as followed:

```
1 @Override
2 protected void doGet(HttpServletRequest request, HttpServletResponse response)
3 throws IOException, ServletException {
4   // Process inputs
5   String success = request.getParameter("success");
6   String error = request.getParameter("error");
7
8   // Setup + populate beans (if required)
```

```
 9     HttpSession session = request.getSession();
10     ServletContext application = getServletContext();
11
12     CartBean cart = (CartBean) session.getAttribute("cart");
13     if (cart == null) {
14       cart = new CartBean();
15       session.setAttribute("cart", cart);
16     }
17
18     StoreBean store = (StoreBean) application.getAttribute("store");
19     if (store == null) {
20       store = StoreBean.buildStoreBean();
21       application.setAttribute("store", store);
22     }
23
24     // Pass data to the JSPs
25     request.setAttribute("cart", cart);
26     request.setAttribute("items", store.getStoreItems());
27     request.setAttribute("success", success != null);
28     request.setAttribute("error", error);
29
30     // Forward to the appropriate JSP
31     RequestDispatcher dispatcher = getServletContext()
32         .getRequestDispatcher("/WEB-INF/storepage.jsp");
33     dispatcher.forward(request, response);
34   }
```

The StoreBean.java file will simply store a list of items sold by the store:

```
1  public class StoreBean {
2      private List<String> storeItems;
3
4      public StoreBean() {
5          this.storeItems = new ArrayList<>();
6      }
7
8      public List<String> getStoreItems() {
9          return this.storeItems;
10     }
11
12     public static StoreBean buildStoreBean() {
13         StoreBean bean = new StoreBean();
14         bean.getStoreItems().add("Apples");
15         bean.getStoreItems().add("Bananas");
16         bean.getStoreItems().add("Coffee");
17         bean.getStoreItems().add("Milk");
18         bean.getStoreItems().add("Biscuits");
```

```
        return bean;
    }
}
```

The storepage.jsp file will simply display a form for the items, the error/success message, and the users current items.

```
<%@page import="java.util.*" %>
<%@page import="pkg.CartBean" %>
<%
    // Retreive the data
    List<String> items = (List<String>) request.getAttribute("items");
    CartBean cart = (CartBean) request.getAttribute("cart");
    Boolean success = (Boolean) request.getAttribute("success");
    String error = (String) request.getAttribute("error");
%>
<!DOCTYPE html>
<html>
    <head>
        <title>The MVC Store</title>
    </head>
    <body>
        <h1>The MVC Store</h1>
        <h2>Add Items</h2>
        <form action="/lab05/store" method="POST">
            <table>
                <tr>
                    <td>Item:</td>
                    <td>
                        <select name="item" required>
                            <option disabled>Please select an item ...</option>
                            <% for (String item : items) { %>
                            <option><%= item %></option>
                            <% } %>
                        </select>
                    </td>
                </tr>
                <tr>
                    <td>Quantity:</td>
                    <td>
                        <input type="number" name="qty" required />
                    </td>
                </tr>
                <tr>
                    <td colspan="2"><input type="submit" value="Add to Cart" /></td>
                </tr>
            </table>
```

```
41          </form>
42          <% if (success) { %>
43              <p><em>Item added to cart</em></p>
44          <% } %>
45          <% if (error != null) { %>
46              <p>Error, <em><%= error %></em></p>
47          <% } %>
48          <h2>My Cart</h2>
49          <% if (cart.isEmpty()) { %>
50              <p>There are no items in the cart. Start shopping!</p>
51          <% } else { %>
52              <ul>
53              <% for (String item : cart.getItemNames()) { %>
54                  <li><%= item %> - <%= cart.getQuantity(item) %></li>
55              <% } %>
56              </ul>
57          <% } %>
58      </body>
59  </html>
```

Note: In this example, we have a little bit of Java to retreive the passed data and generate content. When we start using JSTL and JSP EL, this will be removed from our JSPs.

And finally, the StoreController doPost method will implement the add items functionality (similarly to our AddItemServlet class):

```
1   @Override
2   protected void doPost(HttpServletRequest request, HttpServletResponse response)
3     throws IOException, ServletException {
4     HttpSession session = request.getSession();
5
6     // Get the cart bean from the session
7     CartBean cart = (CartBean) session.getAttribute("cart");
8     if (cart == null) {
9       cart = new CartBean();
10      session.setAttribute("cart", cart);
11    }
12
13    // Retreive + validate request inputs
14    String item = request.getParameter("item");
15    String qtyStr = request.getParameter("qty");
16    int qty = 0;
17
18    String errors = "";
19    if (item == null || qtyStr == null) {
20      errors += "item or qty are null\n";
21    }
22
```

```
23      try {
24        qty = Integer.parseInt(qtyStr);
25      } catch (NumberFormatException ex) {
26        errors += "qty is not a number\n";
27      }
28
29      // Handle error
30      if (errors.length() > 0) {
31        response.sendRedirect("/lab05/store?error=" + errors);
32        return;
33      }
34
35      // Add item to cart
36      cart.addItem(item, qty);
37      response.sendRedirect("/lab05/store?success=true");
38  }
```

Compile and run the application to make sure everything works. The functionality of the store will be identical, however the application has be refactored to follow a MVC design pattern.