Exercise 10:

Creating a JavaServer™ Page

Case Study: Creating a JavaServer Page

Estimated completion time: 1 hour 30 minutes.

In this exercise, you create a JavaServer™ Page to display a customer's account information after a successful customer login. This work was performed by a method in the servlet for the previous exercise. In this new design, the `Controller` servlet handles the processing required by a customer login request, invoking the necessary business methods on the `BankMgr` session EJB to validate the login and create the data model bean used by the JSP when creating the response. The servlet forwards the request to the JSP when it has finished executing the application business logic. The `ShowCustomerAccts` JSP generates the response using the data contained in the data model bean to build a table displaying the customer's account information on the client browser.

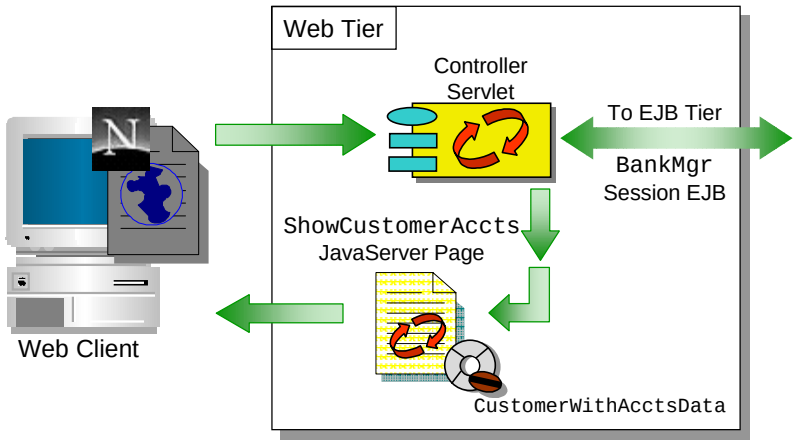

Figure 1 - The `ShowCustomerAccts` JSP

There are two sections to this lab exercise. In the first section, you code selected portions of the `ShowCustomerAccts` JavaServer Page and `Controller` servlet. In Section 2, you deploy and test the new web components.

After completing this lab, you will be able to:

- Describe the function of a `useBean` statement
- Use a `page` directive to import Java class files required by scripting elements used in a JSP
- Explain how to forward a request from a servlet to a JavaServer Page
- Describe two methods that can be used to pass data objects between web components when forwarding a request
- Create a JSP that generates a dynamic response using data model beans and JSP scripting elements

Discussion Topics:

- Discuss alternative ways in which data objects can be passed between web components when a request is forwarded.
- What is the function of a JSP `page` directive?
- What JSP mechanism is used to gain access to a data model bean stored to the session object?

Important Terms and Concepts

Refer to the JDK Javadoc or the lab Appendix for information on the important classes and methods used in this exercise including parameters and return values.

javax.servlet

```
public interface ServletConfig
   public ServletContext getServletContext()
public interface ServletContext
   public RequestDispatcher getRequestDispatcher(java.lang.String
                                 path)
public interface RequestDispatcher
   public void forward(ServletRequest request, ServletResponse
                                 response)
                throws ServletException, java.io.IOException
```

javax.servlet.http

```
public interface HttpServletRequest extends ServletRequest
   public HttpSession getSession(boolean create)
public interface HttpSession
   public java.lang.Object getAttribute(java.lang.String name)
   public void setAttribute(java.lang.String name, java.lang.Object
                                 value)
```

Setup Instructions

The files containing the skeleton code for the JavaServer Page and servlet controller created in this exercise are located in the course resources directory. A web browser serves as the application client in this lab. Use the accompanying course material as a resource when completing this exercise. You can view an example of the completed exercise in the “Solution Code” section of this lab module.

Section 1

Complete the `ShowCustomerAccts` JavaServer Page

Use the accompanying course material as a resource when completing this exercise. A partially completed JSP is furnished with the lab resources for this course as described in the setup section of this lab exercise. Follow the steps listed below to finish implementing this JSP.

1. Locate the file `ShowCustomerAccts.jsp` in the course resource directory and copy it to the `J2EEApp` package in the IDE workspace. Review the code. Notice that several sections of the JSP are incomplete. The JSP code requiring completion has been delimited with the comments "begin: insert new code" and "end: insert new code". You can locate these sections of code using the editor's search tools.
2. Finish writing the code for the `ShowCustomerAccts` JSP as follows:
   - Insert a `page` directive to:
     - Import the required Java class files
     - Set the page's `isErrorPage` attribute to `false`
   - Code a `useBean` command to retrieve the `CustomerWithAcctsData` object stored to the session by the servlet controller
     - Locate the JSP scripting elements, such as expressions and Java code blocks, within the page and delimit them appropriately

Update the `Controller` Servlet

The `Controller` servlet must be modified to invoke the functionality provided by the `ShowCustomerAccts` JSP. You may choose to modify the servlet created in the previous lab exercise or start with the supplied servlet from the lab resources directory as described below.

3. [Optional] Locate the file `Controller.java` in the course resource directory for this lab exercise, and copy it to the `J2EEApp` package in the IDE workspace. Review the code. Notice that several sections of the servlet need modification. The servlet code requiring modification has been delimited with the comments "begin: insert new code" and "end: insert new code". You can locate these sections of code using the editor's search tools.
4. Modify the `Controller` servlet as follows:
   - Update the URL for the static `CUSTOMER_SCREEN` field to point to the `ShowCustomerAccts` JSP file.
   - Modify the section of code in the `processRequest` method that contains the conditional branch for the `CUSTOMER_SCREEN` url processing as follows:
     - Remove the call to `displayCustomerWithAccts.`
     - Store the `CustomerWithAcctsData` object to the session.
     - Forward the request to the `ShowCustomerAccts` JSP for response generation.

Verify Your Work

5. Compile the `ShowCustomerAccts` JSP and the `Controller` servlet. Build the files contained in the `J2EEApp` package. The entire package should compile without errors. Fix any errors before continuing with this exercise. Verify your work against the solution code provided for this module if necessary.

Discussion Topics:

- Discuss ways of reducing the amount of JSP scripting code required in a JSP page to iterate through and display the fields of a complex object.
- What must be done to HTML commands embedded within a JSP for them to be properly executed by the JSP container?

Section 2

Deploy and Test

In this section, you deploy and test the `ShowCustomerAccts` JSP. Consult the "Assembly and Deployment Guidelines" for information on deploying web components into the J2EE container used for this course.

1. Using the J2EE assembly/deployment tool provided for this course, assemble and deploy the `ShowCustomerAccts` JSP into the J2EE runtime environment. To assemble and deploy the application components using the J2EE Reference Implementation deployment tool, you must:
    - Start the J2EE runtime, cloudscape database, and deployment tool.
    - Use the deployment tool to:
        - Undeploy the `BankApp` application.
        - Update the application files for `BankApp`.
        - Choose File > New Web Component to add the `ShowCustomerAccts` JSP to the application:
            - **New Web Component Wizard - `WAR` File General Properties**
                - Web Component will Go In: `BankAppWAR`
                - Contents - Add:
                    - Add Files to `.WAR` - Add Content Files:
                        - Choose the system workspace as the root directory. For example, `C:\<IDEHome>\Development`.
                        - Open the `J2EEApp` package.
                        - Select and add `ShowCustomerAccts.jsp` from the `J2EEApp` package.
                        - Add Files to `.WAR` - Add Class Files: None
            - **New Web Component Wizard - Choose Component Type**
                - JSP: Checked
            - **New Web Component Wizard - Component General Properties**
                - JSP Filename: `J2EEApp\ShowCustomerAccts.jsp`
                - Web Component Display Name: `CustomerAccts`
        - Deploy the application:
            - Return Client Jar: Unchecked
            - Verify the listing of JNDI names.
            - Verify that `BankContextRoot` is listed as the context root for `BankAppWAR`.
2. Start a browser and enter `http://localhost:8000/BankContextRoot/J2EE/` to invoke the servlet controller and test the application. Use the web server port number as configured when the server was installed.

Null values in the display are generally due to a method transaction setting of not supported on the EJB used to generate the page data.

Solution Code: Section 1

```
// ShowCustomerAccts.jsp

<%@ page import="J2EEApp.CustomerWithAcctsData, J2EEApp.AccountData,
                 J2EEApp.CustomerData, java.util.*"
         isErrorPage="false" %>

<html>
<head><title> Customer Data </title></head>
<body>

<form method="get" >

<jsp:useBean id="cwad" scope="session"
class="J2EEApp.CustomerWithAcctsData" />

<h1>Accounts for:<br>
    <%= ((cwad.getCustomerData()).getFirstName() +
        " " +
        (cwad.getCustomerData()).getLastName()) %>
</h1>
<%
if ((cwad.getAccounts()).isEmpty()) {
%>
    <p><h3><strong>No accounts found!</strong></h3><p>

<%
}
else {
%>

    <%-- start table --%>
    <p><table border="1">

        <%-- output row headers --%>
        <tr><th><strong>Account Number</strong>
        <th><strong>Balance</strong>
        <th><strong>Description</strong></tr>

    <%-- output row data --%>
    <%
    Iterator i = (cwad.getAccounts()).iterator();
    AccountData acct = null;
    while (i.hasNext()) {
        acct = (AccountData)i.next();
    %>
        <tr>
            <td> <%= acct.getAcctNumber() %> </td>
            <td> <%= acct.getBalance() %> </td>
            <td> <%= acct.getDescription() %> </td>
        </tr>
    <%
    }
    %>

    </table>
```

```
<%
}
%>

    <h3>
        <p>
        <input type="submit" value="Quit" name="QUIT">
        <input type="hidden" name="action" value="quit">
    </h3>

</form>

</body></html>

************************************************************

// Controller.java

package J2EEApp;

import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.io.*;
import javax.naming.*;

/**
 * MVC controller...web tier entry point to banking application.
 * Provides processing logic for screen navigation and bean management.
**/

public class Controller extends HttpServlet {

   private BankMgrHome myBankMgrHome;

   private static String LOGIN_SCREEN = "/J2EEApp/Login";
   private static String QUIT_SCREEN = "/J2EEApp/QuitMessage";
   private static String LOGIN_ERROR_SCREEN = "/J2EEApp/LoginError";
   private static String CUSTOMER_SCREEN =
      "/J2EEApp/ShowCustomerAccts.jsp";
   private static String DEFAULT_SCREEN = LOGIN_SCREEN;

   public void init(ServletConfig config) throws ServletException {...}

   public void destroy() { }

   protected void doGet(HttpServletRequest request,
                        HttpServletResponse response)
      throws ServletException, java.io.IOException {...}

   protected void doPost(HttpServletRequest request,
                        HttpServletResponse response)
      throws ServletException, java.io.IOException {...}

   public String getServletInfo() {...}

   protected void processRequest(HttpServletRequest request,
                                 HttpServletResponse response)
```

```
throws ServletException, java.io.IOException {

System.out.println("Entering controller.processRequest()");
String url = null;

// get session...create if first time in
HttpSession session = request.getSession(true);

// retrieve/initialize action and
// display results in output screen
String currAction =
   (request.getParameter("action") != null) ?
      request.getParameter("action") : null;
System.out.println("The current action is: " +
                  ((currAction != null) ? currAction : "null"));

if (currAction == null) {
   // display first screen...login screen
   System.out.println("currAction is null");
   url = LOGIN_SCREEN;
}
else {

   //*************************************************
   // start validateLogin
   //*************************************************
   if (currAction.equals("validateLogin")) {
      System.out.println("In action validateLogin");
      String un = request.getParameter("username");
      String pw = request.getParameter("password");
      if ((un == null) || !(loginUser(un, pw, session))) {
         System.out.println
            ("Error: invalid login attempt for userName: " +
               un + " and password: " + pw);
         url = LOGIN_ERROR_SCREEN;
      }
      else {
         System.out.println
            ("In action validateLogin for userName: "
               + un + " and password: " + pw);
         url = CUSTOMER_SCREEN;
      }
   }
   //*************************************************
   // end validateLogin
   //*************************************************

   //*************************************************
   // start loginErrorConfirmed
   //*************************************************
   else if (currAction.equals("loginErrorConfirmed")) {
      System.out.println("In action loginErrorConfirmed");
      url = LOGIN_SCREEN;
   }
   //*************************************************
   // end loginErrorConfirmed
   //*************************************************
```

```
         //*************************************************
         // start quit
         //*************************************************
         else if (currAction.equals("quit")) {
            System.out.println("In action quit");
            session.invalidate();
            url = QUIT_SCREEN;
         }
         //*************************************************
         // end quit
         //*************************************************
         else {
            System.out.println("Encountered unhandled action: " +
                               currAction + " in controller");
            url = DEFAULT_SCREEN;
         }
      }
      System.out.println("Set url to: " + url);

      if (url == CUSTOMER_SCREEN) {
         // display customer data
         try {
            BankMgr bm = myBankMgrHome.create();
            session.setAttribute("cwad",
               bm.getCustomerWithAcctsData
                  ((String)session.getAttribute("userID")));
            bm.remove();
            ServletContext sc = getServletContext();
            RequestDispatcher rd = sc.getRequestDispatcher(url);
            rd.forward(request,response);
         } catch (Exception e) {
            System.out.println
               ("Exception retrieving CustomerWithAcctsData");
            System.out.println("Error: " + e);
         }
      }
      else {
         if (url == LOGIN_ERROR_SCREEN) {
            displayLoginErrorScreen(response);
         }
         else if (url == QUIT_SCREEN) {
            displayQuitScreen(response);
         }
         else if ((url == DEFAULT_SCREEN) || (url == LOGIN_SCREEN)) {
            displayLoginScreen(response);
         }
         else {
            System.out.println
               ("Unhandled URL...reverting to login screen");
            displayLoginScreen(response);
         }
      }

   } // end of processRequest()


    private void displayLoginScreen(HttpServletResponse response)
       throws java.io.IOException {...}
```

```
   private void displayLoginErrorScreen(HttpServletResponse response)
      throws java.io.IOException {...}

   private void displayLoginDataFields(java.io.PrintWriter out) {...}

   private void displayQuitScreen(HttpServletResponse response)
      throws java.io.IOException {...}

   private void initMyBankMgrHome() {...}

   private boolean loginUser(java.lang.String username,
         java.lang.String password, HttpSession session) {...}

} // end of controller

*************************************************************
```

Answers for Discussion Topics:

- Discuss alternative ways in which data objects can be passed between web components when a request is forwarded.

  Answer: Objects can be stored to the session object or as an attribute of the current request using the `ServletRequest` method `setAttribute`.

- What is the function of a JSP `page` directive?

  Answer: A JSP directive contains information to help a JSP container configure and run a JSP page. Directives do not produce any output into the current *out* stream. The `page` directive defines page-dependent attributes, including `language`, `session`, `import`, `isErrorPage`, `errorPage`, `isThreadSafe`, and so on.

- What JSP mechanism is used to gain access to a data model bean stored to the session object?

  Answer: You use the `<jsp:useBean>` action to create, if necessary, and use a bean instance and associate it with a scope and ID, such that the instance is accessible by the JSP scripting elements within the specified scope using the associated ID.

- Discuss ways of reducing the amount of JSP scripting code required in a JSP page to iterate through and display the fields of a complex object.

  Answer: A programmer can create a custom tag library that JSP developers use when they need this functionality.

- What must be done to HTML commands embedded within a JSP for them to be properly executed by the JSP container?

  Answer: Unlike servlets, where HTML commands must be embedded in `println` statements, HTML commands require no special treatment or encapsulation when used inside a JSP.

Filename: lab10.doc
Directory: \\~home\nclarke\FJ310_revC_0301\BOOK\EXERCISE\Exercise_Originals
Template: C:\Program Files\Microsoft Office\Templates\Normal.dot
Title: Case Study: The New User Interface
Subject:
Author: jdibble
Keywords:
Comments:
Creation Date: 10/17/00 10:42 AM
Change Number: 12
Last Saved On: 04/24/01 12:40 PM
Last Saved By: Nancy Clarke
Total Editing Time: 10 Minutes
Last Printed On: 04/24/01 12:40 PM
As of Last Complete Printing
Number of Pages: 12
Number of Words: 2,354 (approx.)
Number of Characters: 13,419 (approx.)