Exercise 9:

Creating a Servlet

Case Study: Creating a Servlet

Estimated completion time: 1 hour 30 minutes.

In this exercise, you create two servlets that handle HTTP requests from browser-based clients requiring access to the banking application's functions. In this exercise you create two servlets each of which interact with the business services provided by the EJB™ tier. The first servlet displays some information from the bank application's customer table based on the `customerID` passed as a parameter when invoking the servlet. Figure 1 shows the results of invoking the `CustomerDetailsServlet` with the `customerID` parameter set to 33.

As illustrated in figure 2, the `CustomerDetailsServlet` servlet interacts with the `Customer` entity EJB using the business methods on the `BankMgr` session EJB. The servlet uses the `BankMgr`'s `getCustomerData` method to retrieve information for the specified customer.

As shown in Figure 1, the servlet processes a customer login request and presents a valid customer with a listing of their bank accounts.

Invalid logins result in a response that displays the error screen illustrated in Figure 2.

As illustrated in figure 3, the `Controller` servlet interacts with the application entity components using the business methods on the `BankMgr` session EJB. The servlet uses the Bank Manager's `loginUser` method to validate a user login request and then invokes the Bank Manager's `getCustomerWithAcctsData` method to retrieve a listing of customer accounts for the current customer. For this exercise, all of the HTML commands required for response generation are contained within private servlet methods.

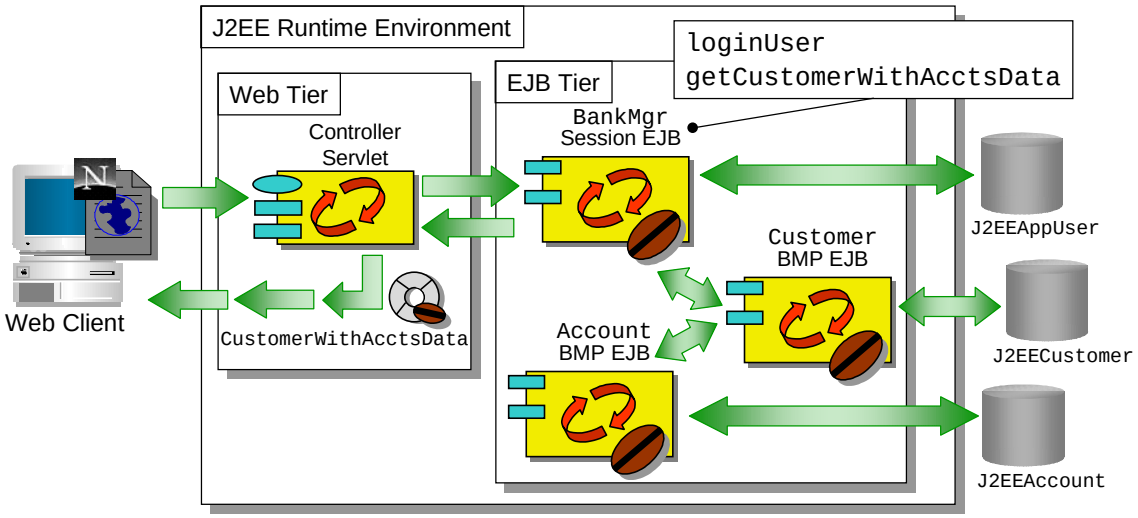

Figure 3 - The Controller Servlet

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

After completing this lab, you will be able to:

- List the steps necessary to deploy a web component into the J2EE runtime environment
- Describe how to access a session EJB from a servlet
- Describe how to store to and retrieve data objects from an `HttpSession` object
- Write code to generate a simple response using HTML commands inside a servlet
- Explain how to use `HttpRequest` and `HttpResponse` objects to interact with browser-based clients

Discussion Topics:

- Explain the difference between storing an object in a private servlet field and storing an object to an `HttpSession`.
- Describe how you could use the `HttpSession` object to control access to an application using a system login mechanism.
- Explain the difference between the `ServletRequest` methods `getParameter` and `getAttribute`.

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 ServletRequest
   public java.lang.String getParameter(java.lang.String name)
public interface ServletResponse
   public java.io.PrintWriter getWriter() throws java.io.IOException
   public void setContentType(java.lang.String type)
public interface Servlet
   public java.lang.String getServletInfo()
   public void init(ServletConfig config) throws ServletException
```

`javax.servlet.http`

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

Setup Instructions

The files containing the skeleton code for the `Controller` servlet 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 `Controller` Servlet

Use the accompanying course material as a resource when completing this exercise. A partially completed servlet 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 servlet.

1. Locate the file `Controller.java` in the course resource directory, and copy it to the `J2EEApp` package in the IDE workspace. Review the code. Notice that several of the servlet's methods are incomplete. The servlet 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. Complete each of the servlet's methods listed below using the comments provided as a guideline for writing the required code segments. Notice that the servlet's `doGet` and `doPost` methods are wrappers for the `processRequest` method, which contains the bulk of the servlet's processing logic.
    - `loginUser`
        - Store the user's ID code to the session object.
    - `displayCustomerWithAccts`
        - Set the response content type.
        - Create a `PrintWriter` output object.
        - Code the `while` loop to display the account data elements.
        - Close the output object.
    - `processRequest`
        - Retrieve (or create if necessary) an `HttpSession` object.
        - Retrieve and store the `action` parameter from the `request` object.
        - Complete the code for the conditional branch that processes the `validateLogin` action.
        - Code the conditional branch that processes the `CUSTOMER_SCREEN` URL.

Verify Your Work

3. Compile 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:

- In what instances would you use the `ServletResponse` method `getOutputStream` to obtain an output object for generating a response?
- When might you use the `ServletRequest` method `setAttribute`?
- What method can be invoked on a session object to release its resources and sever its association with a client?
- Discuss the implications of using the string concatenation operator "+" when generating a response using the `PrintWriter` object's `println` method. For example,

  ```
  out.println("<tr><td>" + acct.getAcctNumber() + "</td></tr>");
  ```

Section 2

Deploy and Test

In this section, you deploy and test the `Controller` servlet. 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 `Controller` servlet 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`.
        - Verify that the method transaction attributes for all of the EJB methods are set correctly.
        - Choose File > New Web Component to add the `Controller` servlet to the application:
            - **New Web Component Wizard - `WAR` File General Properties**
                - WAR Display Name: `BankAppWAR`
                - Contents - Add:
                    - Add Files to `.WAR` - Add Content Files: None
                    - Add Files to `.WAR` - Add Class Files:
                        - Choose the system workspace as the root directory. For example, `C:\<IDEHome>\Development`.
                        - Open the `J2EEApp` package.
                        - Select and add `Controller.class` from the `J2EEApp` package.
            - **New Web Component Wizard - Choose Component Type**
                - Servlet: Checked
            - **New Web Component Wizard - Component General Properties**
                - Servlet Class: `J2EEApp.Controller`
                - Web Component Display Name: `TheController`
            - **New Web Component Wizard - Component Aliases**
                - Add: `/J2EE/*`
        - Choose the Web Context tab for the `BankApp`, and set the Context Root for `BankAppWAR` to `BankContextRoot`
        - 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.

Discussion Topics:

- Explain the purpose of a web component alias.
- What is the function of a `.war` file?
- What is the significance of the Context Root to the application web components?

Solution Code: Section 1

// 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";
   private static String DEFAULT_SCREEN = LOGIN_SCREEN;

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

   public void destroy() { }

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

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

   public String getServletInfo() {
      return "This servlet is the main controller for BankApp";
   }

   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();
            CustomerWithAcctsData cwad =
               bm.getCustomerWithAcctsData
                  ((String)session.getAttribute("userID")));
            bm.remove();
            displayCustomerWithAccts(response, cwad);
         } 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 {
      System.out.println("Entering controller.displayLoginScreen()");
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html><title>User Login</title>");
      out.println("<form method=\"get\">");
      displayLoginDataFields(out);
```

```
      out.println("</form></html>");
      if (out != null) out.close();
      System.out.println("Leaving controller.displayLoginScreen()");
      return;
   }

   private void displayLoginErrorScreen(HttpServletResponse response)
      throws java.io.IOException {
      System.out.println
         ("Entering controller.displayLoginErrorScreen()");
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html><title>Login Error</title>");
      out.println("<form method=\"get\">");
      out.println("<h3>Error validating user login request...<p>");
      displayLoginDataFields(out);
      out.println("</form></html>");
      if (out != null) out.close();
      System.out.println
         ("Leaving controller.displayLoginErrorScreen()");
      return;
   }

   private void displayLoginDataFields(java.io.PrintWriter out) {
      out.println("<h3>Please enter your username and password:");
      out.println("<table><tr><td>Username:<td><input name=\"username\"
                     type=\"text\">");
      out.println("<tr><td>Password:<td><input name=\"password\"
                     type=\"password\">");
      out.println("</table>");
      out.println("<input type=\"submit\" value=\"Login\"
                     name=\"LOGIN\">");
      out.println("<input type=\"reset\" name=\"resetButton\"
                     value=\"Reset\">");
      out.println("<input type=\"hidden\" name=\"action\"
                     value=\"validateLogin\">");
      return;
   }

   private void displayCustomerWithAccts(HttpServletResponse response ,
      CustomerWithAcctsData pCWAD) throws java.io.IOException {
      System.out.println
         ("Entering controller.displayCustomerWithAccts()");

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

      out.println("<html><title>Customer Data</title>");
      out.println("<form method=\"GET\">");
      out.println("<h1>Accounts for:<br>");
      out.println((pCWAD.getCustomerData()).getFirstName());
      out.println(" ");
      out.println((pCWAD.getCustomerData()).getLastName());
      out.println("</h1>");
      out.flush();
      if ((pCWAD.getAccounts()).isEmpty()) {
         out.println("<p><h3><strong>No accounts found!</strong>");
         out.println("</h3><p>");
```

```
      }
      else {
         //start table
         out.println("<p><table border=\"1\">");
         //output row headers
         out.println("<tr><th><strong>Account Number</strong>");
         out.println("<th><strong>Balance</strong>");
         out.println("<th><strong>Description</strong></tr>");
         //output row data
         Iterator i = (pCWAD.getAccounts()).iterator();
         AccountData acct = null;
         while (i.hasNext()) {
            acct = (AccountData)i.next();
            out.println("<tr><td>");
            out.println(acct.getAcctNumber());
            out.println("</td><td>");
            out.println(acct.getBalance());
            out.println("</td><td>");
            out.println(acct.getDescription());
            out.println("</td></tr>");
         }
         //end table
         out.println("</table><p>");
      }
      //output logout button
      out.println("<p><h3><input type=\"submit\" value=\"Quit\"");
      out.println(" name=\"QUIT\"></h3>");
      out.println("<input type=\"hidden\" name=\"action\"
                     value=\"quit\">");
      out.println("</body></html>");

      if (out != null) out.close();

      System.out.println
         ("Leaving controller.displayCustomerWithAccts()");
      return;
   }

   private void displayQuitScreen(HttpServletResponse response)
      throws java.io.IOException {
      System.out.println("Entering controller.displayQuitScreen ()");
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html><title>QUIT</title>");
      out.println("<body><h1>Logout successful...");
      out.println("<br>Session cancelled...");
      out.println("<br>Goodbye</h1></body></html>");
      if (out != null) out.close();
      System.out.println("Leaving controller.displayQuitScreen ()");
   }

   private void initMyBankMgrHome() {
      System.out.println("Entering Controller.initMyBankMgrHome()");
      try {
         Context c = new InitialContext();
         Object result = c.lookup("bankmgrServer");
         myBankMgrHome =
            (BankMgrHome)javax.rmi.PortableRemoteObject.
```

```
                narrow(result,BankMgrHome.class);
         }
         catch (Exception e) { System.out.println(e); }
         System.out.println("Leaving Controller.initMyBankMgrHome()");
      }

      private boolean loginUser(java.lang.String username,
            java.lang.String password, HttpSession session) {
         System.out.println("Entering controller.loginUser()");
         String uid = null;
         try {
            BankMgr bm = myBankMgrHome.create();
            uid = bm.loginUser(username, password);
            bm.remove();
            System.out.println("UID for username: " + username +
               " with password: " + password + " is: " + uid);
            session.setAttribute("userID", uid);
         } catch (Exception e) {
            System.out.println("Exception in controller.loginUser()");
         }
         System.out.println("Leaving controller.loginUser()");
         return (uid != null);
      }

} // end of controller

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

Answers for Discussion Topics:

- Explain the difference between storing an object in a private servlet field and storing an object to an `HttpSession`.

  Answer: Storing an object to a private servlet field makes the object visible to all servlet instances. This is an ideal way to share common data elements used by all servlet instances. For example, the home interface to a session bean or a resource factory reference. Storing an object to a session makes that object available only to the session owner. Data objects owned by or associated with a particular client that must be maintained between client requests can be stored to the session object.

- Describe how you could use the `HttpSession` object to control access to an application using a system login mechanism.

  Answer: The absence of a valid session could be used to trigger the user login process, which would create a valid session, possibly with some type of client specific identifier, after validating the login request.

- Explain the difference between the `ServletRequest` methods `getParameter` and `getAttribute`.

- **Answer: Paraphrased from JavaDoc – The method `getParameter` returns the value of a request parameter as a `String`, whereas `getAttribute` returns the value of the named attribute as an `Object`. Both methods return `null` if the parameter/object does not exist.**

- In what instances would you use the `ServletResponse` method `getOutputStream` to obtain an output object for generating a response?

- **Answer: Paraphrased from JavaDoc – The `ServletOutputStream` returned by `getOutputStream` is used to send binary data in a MIME body response. To send character data, use the `PrintWriter` object returned by `getWriter`. To mix binary and text data, for example, to create a multipart response, use a `ServletOutputStream` and manage the character sections manually.**

- When might you use the `ServletRequest` method `setAttribute`?

- **Answer: Paraphrased from JavaDoc – The method `setAttribute` stores an attribute in the current request. Attributes are reset between requests. This method is most often used with `RequestDispatcher`. For example, for setting a request attribute before forwarding the request to another web component (servlet or JSP).**

- What method is invoked on a session object to release its resources and sever its association with a client?

  Answer: The `invalidate` method invalidates the session and unbinds any objects bound to it.

- Discuss the implications of using the `String` concatenation operator "+" when generating a response using the `PrintWriter` object's `println` method. For example,

  ```
  out.println("<tr><td>" + acct.getAcctNumber() + "</td></tr>");
  ```

  Answer: Using the `String` concatenation operator can increase the garbage collection required when exiting a servlet due to the increased number of `String` buffers employed.

- Explain the purpose of a web component alias.

  Answer: A web component alias is used to map URLs to servlets and JSPs. For example, if you enter `/J2EEApp/*` in the Aliases dialog box, your web component handles all requests coming to the set of URLs in `J2EEApp`. Using the web component alias allows you to configure web tier application entry points based on URLs set in servlets and JSPs using href.

- What is the function of a `.war` file?

  Answer: The `.war` file contains the application web-tier components including `.class`, `.jsp`, `.html`, and deployment descriptor files. A `.war` file is included in the application `.ear` file along with any client and EJB `.jar` files used by the application.

- What is the significance of the Context Root to the application web components?

- **Answer: Paraphrased from the J2EE RI Deployment Tool Help system – The context root is the base directory of the `.war` file. The Servlet Context uses the context root to resolve the path to the `.war` file. The URL to a JSP or servlet takes this form:**

  ```
  hostname:port/ContextRoot/<relative URLs in the .war file>
  ```

Filename:	lab09.doc
Directory:	H:\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:	04/24/01 10:18 AM
Change Number:	2
Last Saved On:	04/24/01 10:18 AM
Last Saved By:	Nancy Clarke
Total Editing Time:	0 Minutes
Last Printed On:	04/24/01 12:39 PM

As of Last Complete Printing

Number of Pages:	16
Number of Words:	3,264 (approx.)
Number of Characters:	18,605 (approx.)