**Advanced Java**

**Final Assignment**

**Web Applications**

**Name:**  **120 Exam Points**

Due: Thursday December 9th at 10:00 AM. This assignment will be considered as part of your final exam score. **The in-class part of your final will be worth 55 points.** You will have two hours to complete the in-class portion of the final exam.

1. You will take the remaining part of your final exam on Thursday December 9th from 10:15 AM to 12:15 PM.
2. This is part of your final exam, so you **are not allowed to speak with anyone about this assignment.** You may not ask for any help from the tutors or classmates. Questions on this project will need to be asked in class. **This document needs to be placed in your drop box with the NetBeans project.**
3. The code must run correctly without any intervention by me. All files must be included in the project, including any property files or binary files that your program needs. You will run the code with me before you leave the final exam. You are welcome to run it with me before your final.
4. You may use your implementation of my Database interface and the related classes from assignment 5 if you don’t like the code provided. Grades will be based mostly on functionality. How web pages look will not add or subtract greatly from the point total. Just make it reasonable. I will take off some points if it is too ugly.
5. Write the code for this final exam in the order the requirements are specified. Once a major requirement is not finished or does not work, **the remaining requirements will** **not** earn any points.

**Requirement 1**: Copy my web project called *WebBanking* to your P drive. Modify it as follows. **(10 points)**

1. Don’t compile or run the program. First, reconfigure it to use your database table and login information. You need to insert code to create tables into the CreateTables class and provide the SQL scripts to create tables in the SQL scripts folder. Ensure the tables you create work with the database code you are using. **(5 points)**
2. Run the code to create tables in your account. (**2 points**)
3. Verify the tables are correctly created.
4. Review the File called index.html
5. Review the servlet called **BankingControl** in a package called **servlets.** This servlet will be the controller for the project. That is all user input will be sent to this servlet and BankingControl will determine which file or JSP page will formulate the response. BankingControl may have helper files and classes to help it do its job. All the forms we have will direct the action to the servlet BankingControl.
6. Create four JSP pages called Menu, NewCustomer, NewAccount, and Transaction. **(3 points)**

**Requirement 2:** Code the following files: *index.html, Menu.jsp, BankingControl.java*, as specified below. (**25 points – rough outline of points given below**)

**index.html** will have look as shown below. Use the appropriate header size tag. The Login button should contain the word Login. This page needs a hidden input field named **action** and having the value **Menu**. BankingControl will transfer control to the JSP page Menu.jsp when it gets the parameter action equal to ”Menu”. You need to create a User object and place it in the Session object before sending control to the Menu.jsp file. Index.html will have input fields for the user’s first and last name. **(5 points)**

Welcome to Banking by <Your first and last name>

Please Enter your First and Last Name

First Name: <input text area>

Last Name: <input text area>

<**Login** Button>

**BankingControl**: Put all your request processing code into the *processRequest* method. **(10 points)**

1. Add the following code. Fill in missing information and complete this concept.

String action = request.getParameter("action");

if (action == null) {

//write code to forward to index.html

return;

}

action = action.trim();

if(action.equalsIgnoreCase("Menu")){

// Obtain the user’s first and last name from the parameters and create a

// User bean that will pass this information to the JSP page. You

// need to write the User class as a JavaBean.

// Place the User class in the common package.

// Place an instance of the User class in the session object.

// Call this session object User

// Forward to the Menu.jsp file and return.

//

}

else{

// Log an invalid action command message

// Output the message: **Invalid Action -- index.html Returned**

// return the file index.html to the client.

}

**Menu.jsp:** Menu.jsp will display page radio buttons to allow the user to select exactly one option from the list of choices. The **Reset** button will reset any input items and the **Submit** button will post the action to the server. BankControl.java will receive this information. Note that Bank control is not codded for these actions yet. You are just creating eh JSP in this step. **(10 points)**

Banking by <Your First and last Name>

Welcomes

<User’s First Name> <User’s Last Name>

Please Make a Selection from the Following Choices.

<choices go here >

<**Submit** Button><**Reset** Button>

The choices are:

1. New Customer –> action = “New Customer”
2. New Account –> action = “New Account”
3. Account Transaction –> action = “Account Transaction”
4. Logout –> action = “Logout”

**Requirement 3:** Add code to the following files as specified below. **(35 points)**

**BankingControl**: Now needs to handle the actions specified by Menu.jsp. In each case, get the user information and forward that to the correct JSP using a User object. Output the appropriate message and close the database connection before forwarding to the appropriate JSP for each new action. If the user logs out, then index html is returned to the client and the User object is removed from the session. **(10 points)**

|  |  |
| --- | --- |
| **Action** | **Forward to which JSP or html file** |
| New Customer | NewCustomer.jsp |
| New Account | NewAccount.jsp |
| Account Transaction | AccountTransaction.jsp |
| Logout | index.html -- Remove User object from session first |

**NewCustomer.jsp** – This JSP will display the fields to allow the user to enter a new customer into our system. The action of this page is either **Add Customer** or **Menu**. The information you need to obtain is the customer’s first name, last name, and address. The **Menu** action will be returned if the user clicks on a button with the name: **Return to Main Menu**. The **Add Customer** action will be returned if the user enters all the customer information and clicks on a button labeled **Add Customer.** Provide a button to clear the data and allow the user to start entering information again. The **Customer Since** value should be set to the current date. You don’t need to collect the birth date for the customer. If you do obtain the customer’s birth date, then you will earn 2 bonus points. The rest of the page is worth 6 points. **(9 points)**

**NewAccount.jsp --** This JSP will allow the user to enter new account information and add it to the system. The user will enter the customer number and initial balance to create the account. The action of this page will be either **Add Account** or **Menu.** The **Menu** action will be returned if the user clicks on a button with the name: **Return to Main Menu**. The **Add Account** action will be returned if the user enters the customer id and account amount and then clicks on a button labeled **Add Account.** Provide a button to clear the data and allow the user to start entering information again. **(7 points)**

**AccountTransaction.jsp –** This JSPwill allow the user to perform different actions. The actions possible on this html page will be **Debit**, **Credit, Get Balance**, **List All Accounts** or **Menu.**  The **Menu** action will be returned if the user clicks on a button named: **Return to Main Menu**. The form on this page will contain input fields for the account number and the amount of the transaction. Radio buttons will be given for the user to select Get Balance, Credit or Debit. Besides the button to return to the main menu, buttons should be given to submit the form and to reset the form. **(9 points)**

**Index.html 🡪** Remains the same**.**

**Requirement 4:** Write the JSP GetBalance.jspand Modify **BankingControl** to handle all the new options specifiedby NewCustomer.jsp, NewAccount.jsp, and AccountTransaction.jsp. **(50 points)**

**BankingControl (30 points)**

|  |  |
| --- | --- |
| **Action** | **Code** |
| Add Customer | Add this new customer to the database, **log** the message **New Customer Added with name <first name> <last name>** and forward processing to Menu.jsp |
| Add Account | Add this new account to the database, **log** the message **New Account Added with amount <money amount>** and forward processing to Menu.jsp |
| Debit | Debit this account in the database by the amount given, **log** the message  **Account <account number> Debited $<amount>** and forward processing to **GetBalance.jsp**. Your code must check to ensure the amount of money is non-negative and that you have enough money to keep the balance of the account >= 0. |
| Credit | Credit this account in the database by the amount given, log the message  **Account <account number> Debited $<amount>** and forward processing to **GetBalance.jsp** |
| Get Balance | Obtain the balance for this account, **log** the message **Balance of Account <account number> is $<amount>.** Forward control to a JSP page called **GetBalance.jsp** to display the balance of the account. |
| List All Accounts | Forward to a JSP called **ListAllAccounts**. |

**GetBalance.jsp** Displays the balance of the account along with other account information including the account number and customer’s name. Provide a button to return to the main menu. (**10 points**)

**ListAllAccounts** Displays all accounts in a table with account information, customer’s name and account balance. Include identification numbers like customer’s ID, account ID, but don’t include address information or birth date or customer since information. (**10 points**)