|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name** |  | **Section** |  |

**PROJECT** ( **Reading from a Sequential File** ) **Total points=10**

**Objective** To read information from a data file.

***PROJECT DESCRIPTION***

This series of projects will be a continuation from the last computer laboratory assignment, which had you use an application that wrote data to a sequential file.   
 This time we will now read that **payroll.txt** data file, which was created earlier, and display it to the program user.

The program code given in **Figure 1** , which follows, reads payroll information from a data file and displays the information to the user. Modify the code to use looping techniques to allow the user to enter five separate payroll reports into a text file.

***Information about This Project***

This program illustrates an example of sequential file processing.

***Steps To Complete This Project***

**STEP 1**  **Open Eclipse or JCreator**

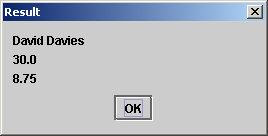
If it is not open, launch Eclipse or JCreator and open the file associated with the lab project that you created in the prior lab exercise.

**STEP 2**  **Add a New File to the Application**

Open Eclipse or JCreator. From the Eclipse or JCreator main menu, click File , choose New and select Java Project . Name the project and Java file as

**ReadData** since this is the class name for this project. Click OK to close the **New** dialog box. When the editor opens, type or copy the program code shown in **Figure 1** ,which follows, exactly as it appears, except substitute your own name in place of Sammy Student.

The program code is written such that when the program is executed, the user will be displayed the various records comprised the data file, one at a time. An example record is shown below.



**STEP 3**  **Compile, Execute and Test the Program**

Move the **payroll.txt** data file such that it will be located within your project **ReadData** workspace folder and compile your program code.

Once you have successfully compiled your program, run your program and observe the various records displayed to you as you sequence through the message boxes.

**PROJECT** ( **Reading from a Sequential File** )

**Figure 1 Program Code for Read File**

|  |
| --- |
|  |

|  |
| --- |
| **import javax.swing.JOptionPane;**  **import java.io.\*;**    **public class ReadData { //Sammy Student, Programmer**  **public ReadData ()**  **{**  **try {**  **String[] firstLine = new String[100],**  **secondLine = new String[100],**  **thirdLine = new String[100];**    **double hours[] = new double[100], wages[] = new double[100];**    **int index;**  **for (index = 0; index < 100; index++) {**  **firstLine[index] = "";**  **secondLine[index] = "";**  **thirdLine[index ] = "";**  **hours[index] = 0.0;**  **wages[index]= 0.0;**  **}**  **FileReader file = new FileReader("payroll.txt");**  **BufferedReader buffer = new BufferedReader(file);**  **index = 0;**  **String line;**    **while((line = buffer.readLine()) != null)**  **{**  **firstLine[index] = line;**  **secondLine[index] = buffer.readLine();**  **thirdLine[index ] = buffer.readLine();**    **hours[index] = Double.parseDouble(secondLine[index]);**  **wages[index] = Double.parseDouble(thirdLine[index]);**    **JOptionPane.showMessageDialog(null, firstLine[index] + "\n"**  **+ hours[index] + "\n" + wages[index], "Result",**  **JOptionPane.PLAIN\_MESSAGE );**    **index++;**  **}**  **buffer.close();**  **System.exit(0);**  **}**  **catch (IOException e ) { System.out.println(e); }**  **}** |

|  |
| --- |
|  |

**PROJECT** ( **Reading from a Sequential File** )

**public static void main(String[] args)**

**{**

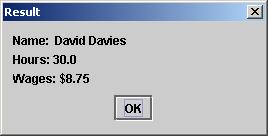
**new ReadData();**

**}**

**}**

**STEP 4**  **Modify the File to Include Currency Formatting**

Modify your code for this Java file such that the data file records will individually display, as shown in the example message box given below.



Each data value is to be labeled as to whether it is a name, number of hours or wage amount. Also, the wage amount is to be displayed with a currency format, with a dollar sign and two decimal points.

One way to incorporate a currency format in your code is given below.

First use the following **import** directive.

**import java.text.DecimalFormat;**

Then within the your constructor at the top of the method, incorporate the statement below to declare a **DecimalFormat** class object.

**DecimalFormat twoDecimal = new DecimalFormat("0.00");**

Apply the format method of the **DecimalFormat** object to each variable for which you want to display with the currency format. For example, for the variable named **wages[index]** , you would use:

**twoDecimal.format(wages[index])**

Remember to concatenate a $ dollar sign with the decimal output to complete the currency formatting.

**STEP 5**  **Modify the File to Include Overtime Pay**

Next, alter your code such that not only the name, hours and wages will be displayed in the message box but also the individual’s gross pay. The gross

pay is calculated in the usual way by multiplying the hours by the wages. An additional amount of 1.5 of the wages is added to gross pay for each hour over 40 hours. A sample output screen is given below.

**PROJECT** ( **Reading from a Sequential File** )



**STEP 6**  **Run the Modified Application**

Before you run your modified program, add new records to your payroll.txt file. Add in a new class to your source (src) folder and call it **CreateData**. The code will be like you did for the last lab but with a slight modification. So add in the code below into your file so your app overall will execute properly for this lab

(you’ll see what I mean later).

|  |
| --- |
| **//Sammy Student: programmer**  **import javax.swing.JOptionPane;**  **import java.io.\*;**  **import java.io.FileWriter;**    **public class CreateData {**    **public static void main(String[] args)**  **{**  **new CreateData();**  **}**  **public CreateData()**  **{**  **int repeat = 1;**  **String answer;**    **do**  **{**  **Write();**  **answer = JOptionPane.showInputDialog ("write payroll " +**  **"data?\n" + "enter 1 to continue or 0 to exit");**  **repeat = Integer.parseInt(answer);**  **}while(repeat == 1);**    **System.exit(1);**  **}**    **static void Write()**  **{**  **try {**    **String firstLine, secondLine, thirdLine, number = "";**  **File check = new File("payroll.txt");**  **FileWriter file;**  **if(check.exists())**  **file = new FileWriter("payroll.txt", true);**  **else**  **file = new FileWriter("payroll.txt");**  **BufferedWriter buffer = new BufferedWriter(file);**  **int size, count = 1;**  **while(number == null || number.equals(""))**  **number = JOptionPane.showInputDialog("how many records?");**  **size = Integer.parseInt(number);**    **do {**  **firstLine = JOptionPane.showInputDialog("Enter name");**  **secondLine = JOptionPane.showInputDialog("Enter hours");**  **thirdLine = JOptionPane.showInputDialog("Enter wage");**  **buffer.write(firstLine);**  **buffer.newLine();**  **buffer.write(secondLine);**  **buffer.newLine();**  **buffer.write(thirdLine);**  **buffer.newLine();**  **count++;**  **}while(count <= size);**  **buffer.close();**  **JOptionPane.*showMessageDialog*(null, "data processed",**  **"Result", JOptionPane.*PLAIN\_MESSAGE* );**  **}**  **catch (IOException e) { System.*out*.println(e); }**  **}**  **}** |
|  |

|  |
| --- |
|  |

Now execute the **CreateData** Java file and add the records below to the

**payroll.txt** file

|  |  |  |
| --- | --- | --- |
| **name** | **hours** | **wage** |
| Betty Boop | 66 | 22.00 |
| Tom Ceina | 41 | 15.45 |
| Kathy Kay | 43 | 8.90 |
| Larry Lance | 52 | 11.35 |

After you add the records, execute your **ReadData** Java file and view the

**PROJECT** ( **Reading from a Sequential File** )

individual payroll records and the corresponding computed gross pay. Ensure that any overtime amounts are correctly computed and added to the gross pay.

Take screen snapshots of your modified program in action and submit them for credit.

**STEP 7**  **Perform a Second Modification to this Application**

With your modification successfully computing overtime pay, add additional program code in your **ReadData** file which, for each employee, will save both the name of the employee and the employee’s respective overtime pay into an new data file called **Overtime.txt** . Each employee’s overtime pay is to be written to the overtime file regardless of the amount of overtime pay, that is, even if the employee receives $ 0 overtime pay.

Test your completed program to ensure that the new **Overtime.txt** data file is created.

Snapshot the contents of the Overtime.txt file and your completed code and submit it for credit.

**STEP 8 Proceed to the Next Project**

The next project has you creating a java file to generate a single employee’s data.

**PROJECT** ( **Report Writing** ) **Total points=20**

**Objective** To write a program that generates a payroll report.

***PROJECT DESCRIPTION***

Write a program that has the capability of generating a report for each individual employee whose payroll data is requested.

***Information about This Project***

This particular program illustrates the importance of report writing.

***Steps To Complete This Project***

**STEP 1**  **Open Eclipse or JCreator**

If it is not open, launch Eclipse or JCreator and open the folder or drive location associated with the lab project files that you created in the prior lab exercises.

**STEP 2**  **Add a New Class File to the Application**

From the Eclipse or JCreator main menu, create a new Class file. In this case, use the file name **Report** since this is the class name for this project.

Save the file in the same folder and / or file storage drive location as the file you created in the last project. Click OK to close the **New** dialog box.

When the Eclipse or JCreator editor opens, write the program code which will

create a payroll report for a particular employee. A sample payroll report is given in **Figure 1** , which follows.

**PROJECT** ( **Report Writing** )

**Figure 1 Sample Payroll Report**

|  |
| --- |
|  |

|  |
| --- |
| .  **\*\*\*\*\*\*\*\*\*\*\*\* Payroll Report \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Employee Name: David Davies**  **Hours: 30**  **Wages:**  **$8.75**  **Gross Pay: $262.50**  **Overtime Pay: $0.00 (included in gross pay)**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** |

|  |
| --- |
|  |

Write your program code such that when the user is allowed to enter an employee name from the given list of employees, the program will search for

the employee’s payroll data and then generate a report in a data file (as shown in figure 1 above), whose file name is the employee’s first initial of his / her first name, followed by their complete last name.

For example, if a report is generated for David Davies, the report will be located in the text file bearing the name **DDavies.txt** .

If the employee’s name is not in the given list and their payroll information cannot be located, then a message is to be displayed to the user indicating this.

Include the following starter code for your file and finish the code starting in your constructor as you see appropriate:

**import java.io.\*;**

**import javax.swing.JOptionPane;**

**public class Report { //Sammy Student, Programmer**

**public Report ()**

**{**

**//code here the logic to create a report for the user**

**}**

**public static void main(String[] args)**

**{**

**new Report();**

**}**

**}**

**STEP 3**  **Run the Application**

Run the application and verify that your employee search and write file program is completely functional. Snapshot a few payroll reports, one for an employee without overtime pay and another for an employee with overtime pay.

**PROJECT** ( **Report Writing** )

Also take snapshots of an example search request when an employee’s name, which is not on the list, is requested, and your message stating that the employee cannot be located is shown.

Submit the snapshots and your completed program code for credit.

**STEP 4 Proceed to the Next Project**

The next project has you utilizing a menu to *ease* the navigation between the three files that you just created and other files you will soon create.

**PROJECT** ( **Creating a Menu Dialog Box** ) **Total points=5**

**Objective** To write a program that creates a menu.

***PROJECT DESCRIPTION***

Write a program that creates a menu which allows the user to select from various functions involving payroll processing.

***Information about This Project***

This particular program illustrates the importance of a switch / case selection structure.

***Steps To Complete This Project***

**STEP 1**  **Open Eclipse or JCreator**

If it is not open, launch Eclipse or JCreator and open the file associated with the lab project files that you created in the prior lab exercises.

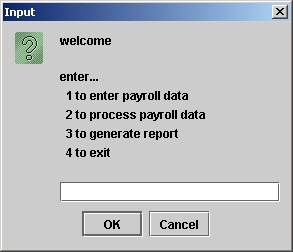
**STEP 2**  **Add a New File to the Application**

From the Eclipse or JCreator main menu, create a new Java application file and use the file name **Menu** since this is the class name for this project. Save the file in the same folder and / or file storage drive location as the file you created in the last project. Then, type the program code shown in **Figure 1** ,which follows, exactly as it appears, except substitute your own name in place of Sammy Student.

The program code is written such that when the program is executed, the user will be displayed the menu screen shown below, allowing the user to select one of the various functions of the application.

**PROJECT** ( **Creating a Menu Dialog Box** )

**Figure 1**

****

**Figure 2 Program Code for Menu.Java File**

|  |
| --- |
|  |

|  |
| --- |
| **import java.awt.Graphics;**  **import javax.swing.JOptionPane;**  **//programmer: Sammy Student**  **public class Menu {**  **public Menu()**  **{**  **String message = "welcome" + "\n", response;**  **message += "\n" + "enter...";**  **message += "\n" + " 1 to enter payroll data";**  **message += "\n" + " 2 to view payroll data";**  **message += "\n" + " 3 to generate report by employee";**  **message += "\n" + " 4 to exit" + "\n" + " ";**  **char answer = 'Y';**  **do {**    **try {**    **response = JOptionPane.showInputDialog(message);**  **int choice = Integer.parseInt(response);**    **switch (choice) {**  **case 1: CreateData cd = new CreateData();**  **answer = 'N'; System.exit(1);**  **break;**  **case 2: ReadData rd = new ReadData();**  **answer = 'N'; System.*exit*(1);**  **break;**  **case 3: Report rpt = new Report();**  **answer = 'N'; System.*exit*(1);**  **break;**  **case 4: answer = 'N'; System.exit(1);**  **break;**  **default: { answer = 'Y'; choice = 0;**  **JOptionPane.showMessageDialog(null,"enter a number: 1 - 4");**  **}**  **}//end switch**  **}//end try**  **catch (Exception e ) { System.out.println(e); }**  **}while(answer == 'Y' || answer == 'y');**  **}**  **public static void main(String[] args)**  **{**  **new Menu();**  **}//end main**  **}//end class** |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
|  |

Hope you stop and think a bit about the interesting switch-case above. Notice you’ve got objects of each class you wish to fire when the user selects a particular menu item. The objects instantiate here for you which starts the class file! Further notice that the three classes you created above had an instantiation of each class in **main** so you can also fire the classes individually when testing and running each java file.

**Continue the same logic here when adding future menu options in your switch case to trigger any java file(s) you’ll be adding into your menu for further menu options!**

**STEP 3**  **Compile, Execute and Test the Program**

After you have typed the given program code, build, compile and execute the program. Observe the menu input dialog box that opens when the application is executed.

Submit a screen snapshot of your menu input dialog box.

**STEP 4**  **Modify the Program Code**

Modify your program code for the **Menu.java** file such that the program will not crash if a non - numeric character is accidentally entered by the user. That is, if the user types the letter A into the Menu input dialog box, the program will catch that a non - numeric character was accidentally entered, display an error message dialog box to the user and allow the user the chance to correctly

enter a number into the dialog box.

Your program is to catch this character exception before any other exception that may happen in the program.

After adding the new program code, run your program again and test the program when a non - numeric character is entered.

Submit a screen snapshot of your error message and completed code for this project.

**STEP 5 Proceed to the Next Project**

The next project has you incorporating a login entry screen to prevent unauthorized users from using the applications within the program.

**PROJECT** ( **Creating an Login Entry Dialog Box** ) **Total points=5**

**Objective** To write a program that requires a login entry screen to access a main program.

***PROJECT DESCRIPTION***

Write a program that will create a menu which allows the user to login or enter an application.

***Information about This Project***

This particular program illustrates the importance of a login screen to precede an application.

***Steps To Complete This Project***

**STEP 1**  **Open Eclipse or JCreator**

If it is not open, launch Eclipse or JCreator and open the folder or drive location associated with the lab project files that you created in the prior lab exercises.

**STEP 2**  **Add a New Class File to the Application**

From the Eclipse or JCreator main menu, create a new Java Class file and use the file name **Login** since this is the class name for this project. Save the file in the same folder and / or file storage drive location as the file you created in the last project. When the Eclipse or JCreator editor opens, type the program code shown in **Figure 7** which follows, exactly as it appears, except substitute your own name in place of Sammy Student.

The program code is written such that when the program is executed, the user will be displayed the login screen shown below, allowing the user to enter their login name.

|  |  |  |
| --- | --- | --- |
|  |  | **Figure 1** |

If the correct login name is entered, SAMMY in this case, this screen appears.

|  |  |  |
| --- | --- | --- |
|  |  | **Figure 2** |

Otherwise, this screen appears.

|  |  |  |
| --- | --- | --- |
|  |  | **Figure 3** |

If the correct login name was entered, this screen also appears, prompting the user for the correct password, which is AUTUMN, according to the following code.

|  |  |  |
| --- | --- | --- |
|  |  | **Figure 4** |

If the correct password is entered, the menu screen that you created in a prior project is opened.

|  |  |  |
| --- | --- | --- |
|  |  | **Figure 5** |

Otherwise, the screen below appears and the application terminates.

|  |  |  |
| --- | --- | --- |
|  |  | **Figure 6** |

**Figure 7 Program Code for Login.Java File**

|  |
| --- |
|  |

|  |
| --- |
| **import java.awt.Graphics;**  **import javax.swing.JOptionPane;**  **import java.lang.String; //programmer: Sammy Student**  **public class Login {**  **public static void main(String[] args)**  **{**  **boolean access = false;**  **String message = "welcome" + "\n", response;**  **message += "enter your name";**  **message += "\n" + " ";**  **String name = JOptionPane.showInputDialog(message);**  **String password;**  **name = name.trim();**  **name = name.toUpperCase();**    **if (name.equals("SAMMY"))**  **{**  **JOptionPane.showMessageDialog(null,"hello " + name);**  **message = "enter your password";**  **message += "\n" + " ";**  **password = JOptionPane.showInputDialog(message);**  **password = password.trim();**  **password = password.toUpperCase();**  **if (password.equals("AUTUMN"))**  **{**  **access = true;**  **}**  **else**  **JOptionPane.showMessageDialog(null, "incorrect password");**  **}**  **else**  **{**  **JOptionPane.showMessageDialog(null, "incorrect login name");**  **System.exit(1);**  **}**  **if(access == true)**  **{**  **try {**  **Menu m = new Menu();**  **System.exit(1);**  **}**  **catch (Exception e) { System.out.println(e);}**  **}**  **}//end main**  **}//end class** |
|  |

|  |
| --- |
|  |

**STEP 3**  **Compile, Execute and Test the Program**

After you have typed the given program code, build, compile and execute the program. Observe the login name dialog box that opens when the application is executed. Enter the login name SAMMY. Then in the appropriate input dialog box, enter the password AUTUMN. When the menu input dialog box appears, type 4 to exit the application.

**STEP 4**  **Modify the Program**

After you have successfully executed the Login program, modify your program code to allow the user three opportunities to enter both the correct name and / or password. If the user, within the three tries, types the correct login name or password, the entrance to the menu application (i.e., Menu.java) will be successful, otherwise the login program will terminate.

**STEP 5**  **Run the Modified Program**

Test the operation of your modified program. Take screen snapshots of the operation of the program with a successful login and when the user fails to login correctly (show 2 failure snapshots). Submit the screen snapshots and

your program code for this file for credit.

**STEP 6 Proceed to the Next Project**

The next project has you generating a “Control-Break” payroll report.

**PROJECT** ( **Implementing Control - Break Logic** ) **Total points=10**

**Objective** To write a program that uses control - break logic.

***PROJECT DESCRIPTION***

Write a program that will demonstrate control - break logic, according to the following instructions.

***Information about This Project***

This particular program illustrates the importance of control - break logic. Control breaks are typically used to summarize data into groups. As data is accumulated within some looping structure, there is a " break " in the program to report a particular subtotal.

***Steps To Complete This Project***

**STEP 1**  **Open Eclipse or JCreator**

If it is not open, launch Eclipse or JCreator and open the folder or drive location associated with the lab project files that you created in the prior lab exercises.

**PROJECT** ( **Implementing Control - Break Logic** )

**STEP 2**  **Add a New Class File to the Application**

From the Eclipse or JCreator main menu, create a new Class file. In this case, use the file name **Summary** since this is the class name for this project. Save the file in the same folder and / or file storage drive location as the file you created in the last project. Click OK to close the **New** dialog box.

**STEP 3**  **Add a New File to the Application**

When the file opens, write the program code that will allow the user to summarize the overtime pay data from the **Overtime.txt** file, that you

created earlier. Apply control - break logic to your code to add the overtime pay of the employees listed in the file to report a particular subtotal, indicated below.

Your program is to **find/display** the overtime subtotals of all employees whose first name begins with A through F and then break to **find/display** the overtime subtotals of all employees whose first name begins with total G through L . Show also a **grandtota**l of all overtime at the very end of your report.

Add this application to the menu program that you created earlier.

**STEP 4**  **Run the Application**

Run the application and verify that the proper subtotals and the grandtotal are computed accurately.

Take a screen snapshot of the operation of your program. Submit your source code as well.

**STEP 5**  **Overall Submission Requirements the Application**

Submit for credit, all your screen snapshots in an easily recognizable (aka label each snapshot accordingly) and orderly fashion as well as your completed program code for each accompanying java file listed for this lab.