CS 1050, Mr. Kramer Programming Project #5 points: 100

Purpose Calculate payroll data using multiple classes.

Due Date As discussed in class. Cannot be resubmitted.

Submissions In this order: printed copies of the source code with line numbers, the input file and the output file. You can create the input file by copying the data in step 9 and pasting it to text file YourName\_S\_05\_Input.txt.

Java Topics Classes, arrays of classes, processing partial arrays, loops.

References Textbook – use the index for relevant topics. The programming standards and guidelines as discussed in class.

Files Included 1050\_Project\_05\_PayrollCalculationsClasses.docx (This file)

Employee.java Employee data tracked

EmployeeParameters.java Options used in the program

EmployeeParameters.txt The parameters for EmployeeParameters

The Java and text files should all be in the same folder as your program and Toolkit.

Testing We will discuss in class how to test this program.

Specification After reading the specification, begin the project by writing method calls in the main method for all methods. Use stubs for the called methods. Complete each method one at a time. Except for the constants noted below, do ***NOT*** use static (global) variables. Also, all output to the output file should be echoed on the console. You can use static final variables ONLY for formatting like New Line or Tab, like:

static final String NL = “\r\n”; // A safe “\n”

1. Assume that a company has N employees. Your program should handle the situation if there are more than N rows of employee data by letting the user know, then terminating the program.

2. Use an array of the Employee class. The array might be partially filled. This separate class has this definition:

public class Employee {

String name; // Name of the employee

public double hoursWorked; // Hours worked in the payroll period

public double payRate; // Hourly pay rate

public double grossPay; // Gross pay based on the number of hours worked

public double adjustedGrossPay; // Gross pay less amount that goes into the IRA

public double netPay; // Gross pay less taxes

public double savingsAmount; // Amount of gross pay that goes to savings

public double iraAmount; // Amount of gross pay that goes into the IRA

public double taxAmount; // Amount of tax based on gross pay and tax rates

public double wealth; // Savings amount + IRA amount

// You can add fields to this class if you want.

} // End class

In the main method’s declaration section, you will define an array of the above class as:

Employee[ ] empl = new Employee[maxEmployees];

where maxEmployees is a number read from the Employee Parameter file as described in step 3.

Using the index i as an example, an instance of Employee can be created with a statement like:

emp[i] = new Employee();

Using the index i as an example, an instance variable can be accessed in a method with a statement like:

empl[i].netPay = empl[i].grossPay – empl[i].taxAmount;

The main program should consist mostly of but not necessarily all method calls. Use appropriate parameter passing between methods and no non-local (static) variables.

3. This program reads from a text file named YourName\_S\_05\_Input.txt and writes to a text file named YourName\_S\_05\_Output.txt. The input file has the employee data. Each line of employee data contains the number of hours worked, the pay rate (both type doubles), followed by the name. One or more spaces separate the fields. Use a try/catch construct to catch a “file not found” condition.

There are five parameters that are used in some of the calculations used in the program. The main method creates an instance of the class EmployeeParameters and uses the instance method getEmployeeParameters() to read these values from the parameter file. The variables in the class are defined as shown in EmployeeParameters.java.

4. Gross pay is calculated in a separate method as follows:

hours ≤ 40.0 Paid at the pay rate.

40.0 < hours ≤ 50.0 hours > 40.0 and less ≤ 50.0 are paid at one and a half times the rate. The first 40 hours are paid at the pay rate.

50.0 < hours hours > 50.0 are paid at twice the pay rate. The first 40 hours are paid at the pay rate. Hours 40 to 50 are paid at time and a half.

Examples: with a pay rate is $10.00 an hour, gross pay is calculated as follows:

# Hours Gross Pay

30 30 x $10 = $300

40 40 x $10 = $400

46 40 x $10 + ( (46-40) x $10 x 1.5 ) = $400 + (6 x $15) = $490

50 40 x $10 + ( (50-40) x $10 x 1.5 ) = $400 + (10 x $15) = $400 + $150 = $550

58 40 x $10 + ( (50-40) x $10 x 1.5) + ( (58-50) x $10 x 2) = $400 + $150 + $160

= $710

Note how the pay rate ($10 in the example above) can be factored out, which you should do in the program. For example, the gross pay for 46 hours is

$10 x (40 + (46-40) x 1.5) = $10 x (40 + (6 x 1.5)) = $10 x (40 + 9) = $490

Do you see why?

5. Use a method to calculate:

a. IRA amount = gross pay \* IRA investment rate / 100.0

b. Adjusted gross pay = gross pay - IRA amount

c. Taxes = adjusted gross pay \* (the sum of the tax rates as decimals)

d. Net pay = adjusted gross pay – taxes

e. Savings amount = net pay \* savings rate / 100.0

6. A method to create a report -- this method will be called after all input and all calculations are done and in turn calls three methods as follows:

a. A method to print heading information as shown in the sample output below. The information includes the report title, column headings for the name, gross pay, net pay, wealth accumulation, taxes for that employee, hours worked, and pay rate, in that order. Also shown here are the first few lines of the data in input file order and the respective totals and average that you can use to ensure you are getting the correct results. Here’s how the report should look (partial list):

In Input Order

Mobile Apps Galore, Inc. - Payroll Report

Name Gross Pay Net Pay Wealth Taxes Hours Pay Rate

------------------ --------- ------- ------ --------- ----- --------

Hancock John 487.21 347.38 73.71 100.85 41.00 11.74

Light Karen L 583.00 415.68 88.21 120.68 50.00 10.60

Fagan Bert Todd 637.20 454.32 96.41 131.90 52.00 10.80

Antrim Forrest N 966.96 689.44 146.30 200.16 62.00 12.24

Camden Warren 566.02 403.57 85.64 117.17 38.40 14.74

Mulicka Al B 559.80 399.14 84.70 115.88 44.33 12.04

etc.

Totals: 14,729.71 10,502.28 2,228.60 3,049.05 1,103.14

Average: 12.36

Number of records read: 25

Note that the totals are accurate; compare yours to these. This heading method has a parameter that indicates the print order of the data. At the end of the list, display the number of records read.

b. A method to print the above detail lines for all employees, matching the headings indicated in part a. Single space the detail lines. The wealth column has the sum of the savings amount and the IRA amount.

c. A method that calculates the totals for gross pay, net pay, wealth, taxes and hours, and the *average* (not sum) pay rate. Print a summary line with those totals and the average. (The summary line should print the totals at the bottom of the appropriate columns.) Also, print a line with a message that contains the number of employees processed which could be less than 30. You can store the totals in an array or in a separate class.

7. After the report has been printed, sort the employees alphabetically and print out the report again with the same print report method. The sort used is located in the Toolkit class and is called selectionSortArrayOfClass which is discussed at the end of this document and will be reviewed in class. Use the print report method described above to print the data in this order.

8. Next, sort the employees in ascending order by gross pay and print out the information again. Use the same selectionSortArrayOfClass for the sort. Use the print report method described above to print the data in this order.

9. The Data. The input data file format is the number of hours (type double), the pay rate (type double) and a name (which may or may not have an initial). One or more spaces separate the fields. Remember to trim the name once you’ve extracted it from the input file. The String method name is ‘trim()’. Use this data:

41.00 11.74 Hancock John

50.00 10.60 Light Karen L

52.00 10.80 Fagan Bert Todd

62.00 12.24 Antrim Forrest N

38.40 14.74 Camden Warren

44.33 12.04 Mulicka Al B

41.75 13.40 Lee Phoebe

24.00 11.40 Bright Harry

41.00 10.40 Garris Ted

43.00 12.00 Benson Martyne

31.90 12.40 Lloyd Jeanine D

44.00 13.50 Leslie Bennie A

48.40 14.40 Brandt Leslie

42.00 12.90 Schulman David

50.10 10.84 Worthington Dan

70.40 12.66 Hall Gus W

40.10 12.74 Prigeon Dale R

43.00 12.44 Fitzgibbons Rusty

50.00 12.24 Feistner Merle S

23.00 12.34 Hallquist Dottie

43.33 10.90 Bolton Seth

43.00 12.20 Taylor Gregg

42.00 12.94 Raskin Rose

50.10 12.44 Kenwood Pat

44.33 14.64 Slaughter Lew

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Using the Sort Routine in the Employee Class

Here’s how to use the sort routine that’s in the Employee class.

// Assume

// sortResult is set when the sort routine is called. It is:

// 0 if the sort ended correctly

// 1 if the employee array is empty

// 2 if the sort type is not one of “Name” or “Gross Pay”

int sortResult = 0;

Employee[ ] empl = new Employee[maxEmployees];

// Note: nNames is an integer with the number of data

// lines read which could be less than maxEmployees.

…

…

// Here are the steps call for sorting the empl array. To sort by name:

sortResult = Employee.selectionSortArrayOfClass(empl,nNames,“Name”);

// To sort by gross pay:

sortResult = Employee.selectionSortArrayOfClass(empl,nNames,“Gross Pay”);