Program 1 - Review of IT 168

100 points

IMPORTANT

Before you start working on this Assignment:

If you are not familiar with using **Visual Studio Code (VSC),** please read (and practice using) the file of Lab01 assigned in IT 168. You will find it under Assignments

IMPORTANT

Staring from this assignment and until the end of the semester, you will have to write code while carefully following the "IT 179 Program Grading Guidelines" described in the file available along under this assignment. Please note that, although I will not add this file to the remaining assignments, you are still required to follow the guidelines included in this file for all assignments.

Set-Up

Create a new Java project named: **`P1`**. In the src folder, create a Java package named: **`edu.ilstu`**. This package will contain all classes that you will write in this assignment.

Objective: Review of IT 168 (or equivalent). Practice with classes, arrays, and files.

The Problem

You are going to write classes to handle a (greatly simplified) company payroll. You will also complete test programs that exercise each of those classes. You will be given a starting point for each of the test programs which you will fill in following the instructions in the comments.

The Classes

You will write two data classes: an `HourlyEmployee` class that manages information a single employee and a `Payroll` class that contains an array of `HourlyEmployee` objects and manages payroll for them. You will also complete the provided `EmployeeTester` and `PayrollTester` classes to thoroughly test your classes. Make sure that your classes follow the specifications below precisely and that your output matches the sample output provided. You may create additional private methods and constants as you deem appropriate. Additional public methods must be discussed with the instructor first. Additional instance variables are not permitted.

The `HourlyEmployee` Class

This class must have exactly the following instance variables (attributes/fields):

- `name`: the full name of the employee
- `employeeID`: the employee's company ID number (note that this may include letters)
- `hourlyRate`: the amount the employee is paid per hour
- `accruedLeaveHours`: the number of hours of leave the employee has available to use. These will be accrued at the rate of 1 per 8 hours worked (including overtime). Note that partial hours are possible.
- `annualHoursWorked`: the number of hours the employee has worked year-to-date. Only hours actually worked are included, not leave.
- `annualEarnings`: the amount the employee has earned year-to-date.

This class must have at least the following methods:

- A default constructor
- A constructor that accepts only the name, id, and hourly rate, setting other fields to 0. This would be appropriately used for new employees.
- A constructor that accepts parameter for all instance variables.
- A getter for the id.
- A `toString` method that returns the data with labels and formatting that matches the sample output.
- A `read` method that accepts a Scanner and reads the data for the employee from the Scanner. Information about an employee will be stored on two lines. The first will contain the employee's name. The second will have the remaining data in the order listed above. See the sample files provided. You should assume that the id will never contain spaces.
- A `write` method that accepts a PrintWriter and writes the data to that PrintWriter in the same format as it is read.
- A `handlePay` method that accepts two parameters: the number of hours worked for the week, and the amount of leave used that week. The method will compute pay; update leave accrued, annual hours worked, and annual earnings; and output the following information:
  <name> earned <pay for the week> and has <accrued leave> hours of leave accrued.
  Rules for the week's work are:
  - Can't use more leave that was previously accrued. Any earned this week doesn't count. Must limit the leave used to no more than what is available.
  - Can't use leave to get overtime. If the hours worked and leave used sum to more than 40, the amount of leave must be limited so that no more than 40 hours of pay is received or leave is 0.

- Work above 40 hours is paid double. Someone who makes $20/hour and works 42 hours will receive $880.
- Overtime work does count toward leave accumulation.
- Used leave does not count toward leave accumulation.

Other notes:

- Any time you are printing to the screen, hours should be formatted with at least 1 digit before and exactly 1 digit after the decimal point. So 0 will be 0.0. 2.28 will be 2.3.
- Any time you are printing to the screen, you should be formatting currency properly.
- Data written to the file should not be formatted.
- Make sure you look at the sample input and output files as well as the sample console output in "sample EmployeeTester output.txt" to help you understand expectations and formatting.

The `EmployeeTester` Class

I've provided a starting point for this class with a lot of comments. Follow the instructions in the comments, referring to the sample output for clarification (and reaching out to me for further clarification as needed).

You should be completing this class as you work on the `HourlyEmployee` class, using the tester class to check your work on the employee class as you go. Do not write the whole `HourlyEmployee` class and then start work on the tester class. That may sound faster, but it will almost certainly be much slower.

The `Payroll` Class

The Payroll class must have the following instance variables:

- An array of HourlyEmployee objects that the Payroll object manages
- The number of employees currently in the Payroll

You may assume that there will never be more than 100 employees.

The Payroll class must have the following methods:

- A default constructor
- A `readFile` method that accepts a file name parameter (as a String) and reads the Employee data in the file into the array of employees. The file will data will be in the format expected by the HourlyEmployee's read method. This method must replace any previous contents of the array.
- A `writeFile` method that accepts a file name parameter (as a String) and writes out the information in the employee array to the corresponding file in the format used by the HourlyEmployee's read and write methods.

- An `addEmployee` method that accepts an `HourlyEmployee` parameter and add that employee to the array.
- A `displayEmployee` method that accepts a String parameter that is an employee id and displays the employee to the screen in the format of the `HourlyEmployee toString`.
- A `displayAllEmployees` method that displays all of the employees in the array to the screen in the format of the `HourlyEmployee toString`.
- A `runPayroll` method that accepts a file name parameter (as a String) and handles weekly payroll based on the information in the file. Each line of the file will have an employee id followed by the number of hours worked and the number of hours of leave used that week. The method will handle the pay for each employee who worked that week using the information from the file. If the file is not found, the method should print an error message, but should not exit the program.
- A private `findEmployee` method that accepts a String parameter that is an employee id and returns the corresponding `HourlyEmployee` from the employee array or void if not found. This should be a well-written linear search using a while loop.

The `PayrollTester` Class

I have provided a program to test your `Payroll` class in `PayrollTester`, along with sample output for the program. I strongly encourage you to work through the `Payroll` class piece by piece, testing each piece of functionality. Note that you can comment out parts of the `PayrollTester` code to allow you to focus on specific pieces of the `Payroll` class. You are also encouraged to write your own tests of the `Payroll` class and its parts.

Make sure that the full `PayrollTester` program works correctly, producing output that matches my sample output before you submit.

Again, you should use the sample input and output files to help you understand expectations for the program. Your output should match the provided output exactly.

Provided Files

I have provided the `EmployeeTester.java` (which you need to fill in) and `PayrollTester.java` files, along with input files, sample output files (which start with mec_) and sample output files for the tester programs which show what should be written to the console. These are all in a .zip file attached to the program 1 assignment in ReggieNet.

Submission

The **final** submission will consist of a zip file containing the files:

```
HourlyEmployee.java
EmployeeTester.java
Payroll.java
```