Employees and Paychecks

CS 5004 Object Oriented Design

1. **Goals:** 
   * Get more practice creating useful class objects
   * Creating your own test file
   * Override toString()
   * Use Static variables in a meaningful way
   * Explore a useful way to use exception handling
2. **In Recitation:**

In test driven development we create the test file first and then the application. In recitation your TA will go over this assignment and offer you some advice and insight to its completion. Your submission in recitation will be to create the testing file for the below assignment and the testing file only. You are free to work in groups, but don’t implement any code as a group. You are also welcome to examine and discuss the provided driver file.

Submit as much of the testing file as you were able to complete.

If you are in an asynchronous section, you may submit your testing file as proof

of completion. The due date for this will likely be before the lab. I want to see you did this prior to starting work on the actual code. Otherwise, you pass it off in a recitation.

1. **Instructions:**

Volunteer work is admirable, but many people enjoy being paid for the work they do for an employer. Depending on the type of employee, said employee may be paid by the hour, a fixed rate (salaried), or be on some bonus/commission payment scale.

This lab will focus on building two classes to manage and calculate hourly employee pay.

##### Part 1: Employees and Paychecks:

Employee knows their names (String), their employee id(integer), the number of hours they worked in a given week(double), and their pay rate(double).

Paycheck knows an employee name, a rate, and the hours worked. Paychecks calculate the weekly pay based on rate \* hours if the hours are 40 or less. If the hours worked exceeds 40, the Paycheck applies an overtime rate of 1.5x for all of the hours in excess of 40.

Employee:

* An Employee constructor that takes a name, and pay rate as parameters and sets the instance values appropriately.
  + This constructor should also initialize the hours worked to zero (0) upon instantiation.
  + This constructor should also set the Employee id based on how many employee objects have been created. You’ll need a static variable for this.
* An Employee method addHoursWorked() which takes a parameter (double) and adds the value of that parameter to the current number of hours the Employee has worked this week.
* An Employee method resetHoursWorked() that resets the employee’s hours worked for the week to zero.
* An Employee method getWeeklyPay() which, when called, returns a new Paycheck object that is initialized to the current Employee name, rate and hours worked and then resets the hours worked.
* An Employee method getWeeklyPay(Paycheck Paycheck) which accepts a Paycheck object, resets the hours worked and returns that check
  + it should also verify that the check is not null before resetting the hours as a means of protection. It should throw an appropriate exception and halt the program if this occurs
* Add a payRaise(double) method that will increase the pay by a set percentage
* A getNumEmployees() that returns the total number of employees that have been created. (You’ll need a static variable for this.)

Pacheck

* A Paycheck constructor that takes the employee name, id, rate, and hours worked as parameters and calculates (and stores) the total pay for the week.
* A Paycheck constructor that takes an employee object and extracts the name, rate, id, and hours worked as parameters and calculates (and stores) the total pay for the week.
* A paycheck constructor that takes nothing and does nothing
* A loadCheck method that accepts an Employee object and uses it to load its internal information
  + add some protection here. If someone attempts to load a check with a null employee, throw the appropriate exception that will halt the program
* A Paycheck method getTotalPay() that returns the total pay for the week.

Both

* A toString method for BOTH classes, allowing Employee objects to be printed by the employee name, and Paycheck objects by the totalPay (in proper dollars/cents $xxx.yy format)..

Part 2: Testing and Documentation

For each method within Employee and Paycheck:

* Create a testing file and get to as close to 100% coverage as possible
* Include adequate documentation for all submitted files

**Note: You don’t have to test getNumberOfEmployees()**

Part 3: Exception Handling

Analyze the code and consider what Employee or Paycheck creation wouldn’t make sense. Also consider any other methods that should not accept certain values. For example, should you be able to add negative hours. I’ve also put in two separate places with required exception handling.

You can also reverse engineer this from the provided driver.

Make sure to include exception handling for any added functionality.

Part 4: Driver

I’ve included the driver for you this time. In return, I’d like to see some good exceptions. I’ve maxed this assignment out at 85% instead of 90%. Make sure you are clear in your report what exceptions you added.

<https://www.dropbox.com/scl/fi/ibgc6ki5hsvpsqmhbw35z/Driver.java?rlkey=2f7f0xra0bgl2zwc5ty9dpc8x&dl=0>

If you are successful, your output should look similar to this:

<https://www.dropbox.com/scl/fi/9wvk3wy81myvjnxuomah0/sample.txt?rlkey=52ydhbndozn5t4zse8vi0c63a&dl=0>

1. **Extensions:**

Outside academics you will not get specific requirements. Each lab assignment is worth 100 points, but the base requirements will only get you to 85% - 90%. If you want an A, you’ll have to find a way to go above and beyond what is asked. I’ll often make some suggestions to you in this section, but it is entirely up to you what you’d like to add to the assignment. Make sure you know who your grader is and discuss extension expectations with them. You won’t have to do all of the extensions to get 100% credit.

Extension suggestions:

* Add more than the requested exception handling
  + I would say any added exception handling would be worth 5 points. Significant additions would be 10.
* Add additional functionality not requested
  + Minimal addition 5 points, significant 10 points
* Reach the highest possible coverage for testing
  + 100% or the highest available 10 points
* Add more prototype testing and functionality to the driver
  + Minimal addition 5 points, significant 10 points
* Look ahead and see if you can figure out how to have employees store old Paychecks with array lists when a get weekly Paycheck is called
  + Done to completion and with quality 10 points

1. **Report:**

Each assignment must include a short report. The generation of this report should take you no more than 15 minutes. This gives you a chance to reflect back on what you learned and it makes grading easier on your grader. For this report, I want the following sections:

1. Reflection (*What did you learn?)*
2. **Extensions (*What extensions are you requesting?)***
3. Grading Statement (*Based on the rubric, what grade do you feel you deserve? Be honest.)*
4. Academic integrity statement *(found on the landing page)*
5. **Submission:**

Please read carefully. Failure to follow submission instructions can result in a reduced score.

Submit all files on Canvas under the appropriate assignment. Make sure to include the following named as follows:

Submit your files as a single zip file named: “Your Name”\_”Assignment”.zip

Unless your grader requests it, do not submit your entire project folder. Do not include any JavaDocs, and make sure you remove any package statements. Include the following:

* Employee.java
* Paycheck.java
* Driver.java
* EmployeeTest.java
* PaycheckTest.java
* Report.pdf

Submission checklist:

* Did you include adequate comments?
* Did you include comment blocks at the top of each file?
* Did you name your files as requested?
* Does your code compile?
* Did you remove any package lines generated by your IDE?
* Did you take care of any warnings presented by your IDE?

| Employee | | |
| --- | --- | --- |
| constructors | 5 | 0 |
| addHoursWorked() | 5 | 0 |
| resetHoursWorked() | 5 | 0 |
| getWeeklyPay() & getWeeklyPay(paycheck) | 5 | 0 |
| payRaise() | 5 | 0 |
| toString() | 5 | 0 |
| getNumEmployees() | 5 | 0 |
| Paycheck | | |
| constructors | 5 | 0 |
| getTotalPay() | 5 | 0 |
| loadCheck() | 5 | 0 |
| toString() | 5 | 0 |
| JUnit Tests | | |
| all constructors | 5 | 0 |
| addHoursWorked() | 5 | 0 |
| resetHoursWorked() | 5 | 0 |
| getTotalPay() | 5 | 0 |
| payRaise() | 5 | 0 |
| Misc | | |
| minimum of 4 exceptions handled | 5 | 0 |
| Not included in total possible: | | |
| Poor code quality | -20 | 0 |
| Does not compile | -100 | 0 |
| Extensions (Not calculated without report) | 10 | 0 |
| Late penalty | -20 | 0 |
| Creative or went above and beyond | 10 | 0 |
| Code contains warnings | -20 | 0 |
| Report missing or missing sections | -100 | 0 |
| Missing academic integrity statement | -100 | 0 |
|  | |  |
| TOTAL POINTS POSSIBLE out of 100 | 85 | 0 |