Concordia University
Computer Science & Software Engineering
Faculty of Engineering and Computer Science

Concordia University
Comp 249 – Winter 2017
Object-Oriented Programming II
Assignment 4 – Due Monday April 10, 2017@ 11:00 am

Important Note: no deadline extension will be granted
and no late assignments will be accepted (not even with a penalty!)

Purpose: The purpose of this assignment is to allow you practice Object Oriented Design, File I/O, Exception Handling, Interfaces, ArrayList, and Linked Lists.

The objective of this programming assignment is to create a fully functioning system that handles Concordia University payment system. The system will be used by the university to calculate the payment of the different *employees* (e.g. full time and part time instructors, teaching assistants and staff); and also the payment of different *services* such as electricity, phone, snow removal, cleaning, etc.
Based on the following narrative, you need to come up with an Object-Oriented design to represent the different entities in the system. Like in real world, some of the narratives are just noise and has nothing to do with the payment system. It is your task to filter out the noise.
It is expected that your implementation handles the appropriate exceptions if the user tries to enter information of the wrong (unexpected) type. That is, you need to take into account any possible incorrect entries by the user. Additionally, you should take advantage of code reuse through inheritance and polymorphism whenever needed.

You should notice that all information provided in this assignment (names, salary, rules of payment, and all other specifics) are merely fictitious and have no real merits whatsoever in reality. This information is only provided to allow simulation of the system. Additionally, when you add further records, please avoid using any real names!

The assignment has **two parts**. **Part 1 is mandatory**, whereas **Part 2 is optional**; however completing it entirely and correctly can earn you **4 bonus points.**

Part 1: Employees (10 pts)

At Concordia University, we have *students*, *faculty*, and *staff members*. Below are the details for the different categories of employees:

- In order for a *student* to qualify for a teaching assistant (TA) position, the student must be a current registered student; an alumnus cannot be a TA. The TA is paid by a fixed-rate times the number of hours in his/her contract. We have two kinds of TA's: a graduate teaching assistant (GD_TA) and an undergraduate teaching assistant (UD_TA). The rate is slightly different for each; the GD_TA's rate is 1.2 the UD_TA's rate. The current fixed rate for an UD_TA is 18.25$/hr.

- For *faculty* members, we have permanent and part-time members. A permanent faculty is paid a fixed salary, whereas, the part-time faculty is paid based on the number of hours in the contract. For part time faculty members, if the class size is between 40 and 60 students, they get an extra $500. If the class size is >60, they get an extra $1000. Each part time member can teach exactly one course per term. The term is 4 month.

- All university *staff* members are permanent and have a fixed annual salary, which increases every 3 years by a bonus percentage based on a performance code. The performance code is assigned to the employee by his/her direct supervisor. Here are the different performance codes, and their effects on the bonus calculation:

Code	Bonus %
A	8%
B	6%
C	3%
D	1%
E	0%

(*Note:* an x% bonus indicated x% increase of the salary)

- The initial information of *full-time faculty* members are maintained in a file called ***Full-Time-Faculty.txt***. Each record in this file is composed of: `Employee ID, First Name, Family Name, City of Residence, Hire Year, and Salary.`

- The initial information of *part-time faculty* members are maintained in a file called ***Part-Time-Faculty.txt***. Each record in this file is composed of: `Employee ID, First Name, Family Name, City of Residence, Hire Year, Hourly Rate, Number of Hours of Current Term, and Number of Student in Class.` The Hire Year indicates the first year the professor joined the university.

- The initial information of *staff* members are maintained in a file called ***Staff.txt***. Each record in this file is composed of: `Employee ID, First Name, Family Name, City of Residence, Hire Year, Salary, and performance code.`

Now, while the information in the above three files are correct, the information of the file related to the TAs, ***TAs.txt***, is somehow corrupt and hence it includes extra records that are incorrect. In specific, the file includes records that shows some alumnus as TAs, which is not permitted. These records must be treated correctly by your code, as explained below. The general format of the records of the ***TAs.txt*** file is as follows:
`Employee ID, First Name, Family Name, City of Residence, Hire Year, Classification of TA` (which can be , Grad, UGrd, or Alum), `Current Number of Classes` the TA is involved with, and `Total Number of Working Hours` for these Courses. The Hire Year indicates the first year the TA was first assigned any TA contract at the university.

You are required to design and write the implementation of the employees' payment subsystem, these are the needed requirements:

1. Write the code of **3** methods called ***addFTRecords()***, ***addPTRecords()*** and ***addTARecords()***. Each of these methods must work exactly as follows:
   a. Open the proper text file (i.e. for Full-time, Part-time, TA) and reads its contents into an **ArrayList**. This method (and all following methods) may choose to accept an already open stream of the files, or open them inside the method; this is left for you to decide.)
   b. Prompt the user requiring the information of the new record to be added. The user is expected to enter an employee ID, followed by the rest of the record. The user enters -1 to indicate that he/she has no more records to add. However the user may mistakenly enter an Employee ID that already exists. Your code must detect this problem and reject the entry, then keep looping until the user enters a non-existing ID. You should notice that the entered ID must not match any of the existing IDs in any of the files (Hint: You should take an advantage of the *contains()* method of ArrayList).
   c. All these entered new records must be kept in the ArrayList. Once the user enters -1, the ArrayList must be stored in the related text file; consequently the file is permanently updated with the new records.

2. Write the code of a method called ***findTermSalary()***, which calculates the combined total salary of part-time faculty and teaching assistants (TAs). This method must exactly work as follows:
   a. Open the proper text files and read them into a **Linked List**. <u>You are required to design</u> this Linked List (i.e. you must write the class and all the methods of this linked list). <u>Do NOT</u> use any data types provided by the Java programming language. Additionally, you should take advantage of inner classes when designing this Linked List class.
   b. Iterates through the two linked lists and calculate the combined total salary of part-time faculty and TAs
   c. Display a message with this information (i.e. just use *System.out.println()*).

3. Write the code of a method called ***findHighest_and_Lowest_FT_Salary()***, which finds the highest and lowest salary for any full-time faculty. This method must exactly work as follows:
   a. Open the *Full-Time-Faculty.txt* file and read it into a Linked List, you should use the same linked list class that you already created above.
   b. Search the list for the highest and lowest salaries, and display the full record of these employees. In case multiple records exist with highest and lowest salaries (i.e. 2 or more employees have the same highest salary; you need to display all of them.)

4. Write the code of a method called ***Increase_Staff_Salary()***.This method must exactly work as follows:
    a. Open the *Staff.txt* file and read it into a Linked List similar to what you did above.
    b. Iterate into all records and increase the salaries based on the performance evaluation code. For example, if the salary of a staff is 62,000$ and the last evaluation code is B, then the salary must be changed to 65,720$, which reflects the 6% increase.
    c. You must also reset all salary increase codes afterwards to E, so further calls for this method before the evaluation take place again (in another 3 years) do not result in multiple incorrect increases.
    d. Save the contents of the linked list into the *Staff.txt* file; hence permanently changing the salaries to reflect the increases, and resetting bonus codes.

5. Write an interface called **Ordered**. This interface must have two abstract methods, called ***precedes()*** and ***follows()***. Each of these methods expects one parameter, of type Object, and returns a Boolean value. The semantics of these methods are as follows:
    a. ***precedes()*** returns *true* if the calling object precedes the passed object, based on a given rule that will eventually be set by the class implementing the interface; otherwise the method returns *false*;
    b. ***follows()*** returns *true* if the calling object follows the passed object, based on a given rule that will eventually be set by the class implementing the interface; otherwise the method returns *false*;
    c. The Employee class must implements the **Ordered** interface. The semantics of the two methods are as follows:
        i. ***precedes()*** returns *true* if the Hire Year of the calling Employee precedes the Hire Year of the passed Employee (for example 1998 precedes 2017); otherwise the method returns false.
        ii. ***follows()*** returns *true* if the Hire Year of the calling Employee follows the Hire Year of the passed Employee (for example, 2016 follows 2009); otherwise the method returns false.

6. You must submit a UML class diagram that shows your classes, their hierarchy and their relations.

<u>Part 2: Services [Bonus 4pts]*</u>

Concordia does not only pay employees, it has to pay suppliers bills too. There are two types of suppliers' bills: *subscription* and *service* bills.

➢ *Subscriptions* are regular payment bills with a fixed amount of money paid weekly, bi-weekly, monthly, trimester, semester, or yearly for services used by the university from providers such as electricity, phone, internet, e-books, magazines, etc. (take advantage of enumerator to design this information).

➢ *Services* are payment bills with varying amount of money paid based the hourly rate of the service such as snow removal, cleaning, painting, plumbing, security, etc.

The bookkeeping information of the different bills is maintained in a file called ***Bills.txt***. Below are the details for the two suppliers' bills:

- *Subscription* bills: Each subscription supplier has `Supplier ID, supplier Name, Company Name, start Year, Bill number, subscription type, Subscription amount.`

- *Service* bills: Each service supplier has `Supplier ID, Service Name, Company Name, Start Year, Bill number, Number of hours, hour rate, and Total bill.`

You are required to design and write the implementation of the bills payment subsystem, you should take advantage of inheritance, polymorphism and interfaces as needed. These are the requirements:

1. Write the code of **3** methods called ***addBill()***, ***updateBill()*** and ***RemoveBill()***. Each of these methods must work exactly as follows:
    a. Opens the Bills.txt file and reads its contents into an **ArrayList** or a **linked List.** Check that the bill does not exist, and add it, then save the content when entering the information of the added bills is finished.

b. Opens the Bills.txt file and reads its content to look for a specific bill (using bill number for example) and update the bill attributes (i.e. for *subscription* bill type you update subscription type for example, or for *service* bill you update the number of hours and total bill).

c. Opens the Bills.txt file and iterate though it looking for information that match the passed parameter to this method. If any match is found, the bill must be deleted (e.g., passing supplier ID and delete the bill(s) of this suppliers).

2. Write the code of a method calls ***findSupllierTotal Bills()***, which calculates the combined total bills for a specific supplier.
3. Write the code of a method called ***find Highest _and _Lowest Service(),***which iterates through the **Bills.txt**, and finds the services suppliers with the highest and the lowest hourly rate, then displays the name and the service of such supplier.
4. You must submit a UML class diagram that shows your classes, their hierarchy and their relations.

General Guidelines When Writing Programs

- Include the following comments at the top of each class you are writing.

```
// -----------------------------------------------------
// Assignment #4
//
// Written by: (include your name(s) and student ID(s))
// -----------------------------------------------------
```

- When commenting your code provide on the top a general and clear explanation of what the piece of code is doing; and within that piece of code if there is any method or any loop specify briefly what is doing. Include comments as needed.
- Display clear prompts for the user whenever you are expecting the user to enter data from the keyboard.
- All outputs should be displayed with clear messages and in an easy to read format.
- End your program with a closing message so that the user knows that the program has terminated.

Submission & Demo

When you have finished the program, you must submit the assignment online to the EAS system.

1. Create **one** zip file, containing the two parts: I and II separately, where each part contains the files (.java and .html and test cases).

   Please name your file following this convention:

   – If the work is done by 1 student: Your file should be called *a#_studentID*, where # is the number of the assignment *studentID* is your student ID number.

   – If the work is done by 2 students, ONLY ONE of the members can upload the file (do not upload twice): The zip file should be called *a#_studentID1_studentID2*, where # is the number of the assignment *studentID1* and *studentID2* are the student ID numbers of each student.

2. Assignments must be submitted in the right folder of the assignments. Upload your zip file at the URL: https://fis.encs.concordia.ca/eas/ as *Programming Assignment **4.*** **Assignments uploaded to an incorrect folder will not be marked and result in a zero mark. No resubmissions will be allowed.**

3. **Finally, notice that a demo is required for the assignment. Failure to do your demo, or missing you reserved demo time, will result in a zero mark for the assignment regardless of your submission. *There will be no substitution for a missed demo time.* Please see course outline for further details.**

<u>Evaluation Criteria for Assignment 4</u> <u>(10 points)</u>

Part 1 - **Mandatory** (10 points)	
UML	0.5 pts
addFTRecords(), *addPTRecords()*, and *addTARecords()* methods	1.5 pts
Correct Linked List Implementation	2.0 pts
findTermSalary(),Increase_Staff_Salary(), and findHighest_and_Lowest_FT_Salary()	3.0 pts
Ordered Interface, *precedes()* and *follows()* methods implementation	2.0 pts
General correctness, style, comments, etc.	1 pt
Part 2 - **Optional** (4 bonus points)	
Full correct and complete implementation of requirements of this part	4 bonus points