LINKED LISTS

1. OBJECTIVE

(a) To review and strengthen concepts about linked lists through an application development.
(b) To review techniques of dynamic data structure creation and manipulation.

2. LABORATORY

This lab will be conducted in the Computing Lab 1 (N4-B1-8) in SCE. This is an individual experiment. You cannot use the classes that implement the **Collection** interface (please refer to http://java.sun.com/docs/books/tutorial/collections/), except for Array class. We are going to implement these classes (like Vector, LinkedList, Stack, TreeSet, etc.) in CPE/CSC105, instead of using them directly from Java.

No make-up is allowed for students who have missed their stipulated lab classes without any acceptable excuse like having a valid leave of absence from the school or on medical leave. The students will be deemed to have failed the particular lab work. In case you have valid reasons to do makeup, you must inform the lecturer in charge. The makeup should be done within the same week, during the lab sessions attended by other groups.

3. EQUIPMENT

Hardware: The PCs running under the LINUX environment in Computing Lab 1.
Software: NetBeans, SUN JAVA compiler (**javac**) and interpreter (**java**). Your programming will only be tested by the lab markers using javac and java on the lab PCs.

4. EXPERIMENT

Re-implement lab 4 using a linked list, i.e. no array is allowed for customer records but one still can use arrays for names, addresses, etc. We also want to supplement the customer account records with the transaction records as shown in the following diagram. At the beginning of each month (**assuming current month has 30 days**), all the customer records are read into the computer from the hard disk. Throughout the month, customers may deposit (credit) into their saving accounts, checking accounts or fixed accounts. They can also withdraw (debit) from their saving and checking accounts but not from the fixed accounts (for simplicity). Each transaction will be recorded by the computer. At the end of the month, the remaining balance of each customer will be computed and all customer records will be saved back into the file. Each transaction consists of 6 fields: an **account id**, a **date** (including the **time**) at which the transaction took place, the **type** of transaction (credit or debit), the **amount** of money involved, and the **location** (represented by a number).

Note that the set of customer records forms a linked list. The set of transactions for each customer also forms a linked list. Therefore, there are two types of linked lists involved. Your task now is to translate this structure into declarations in Java.

You are requested to provide the following additional methods:

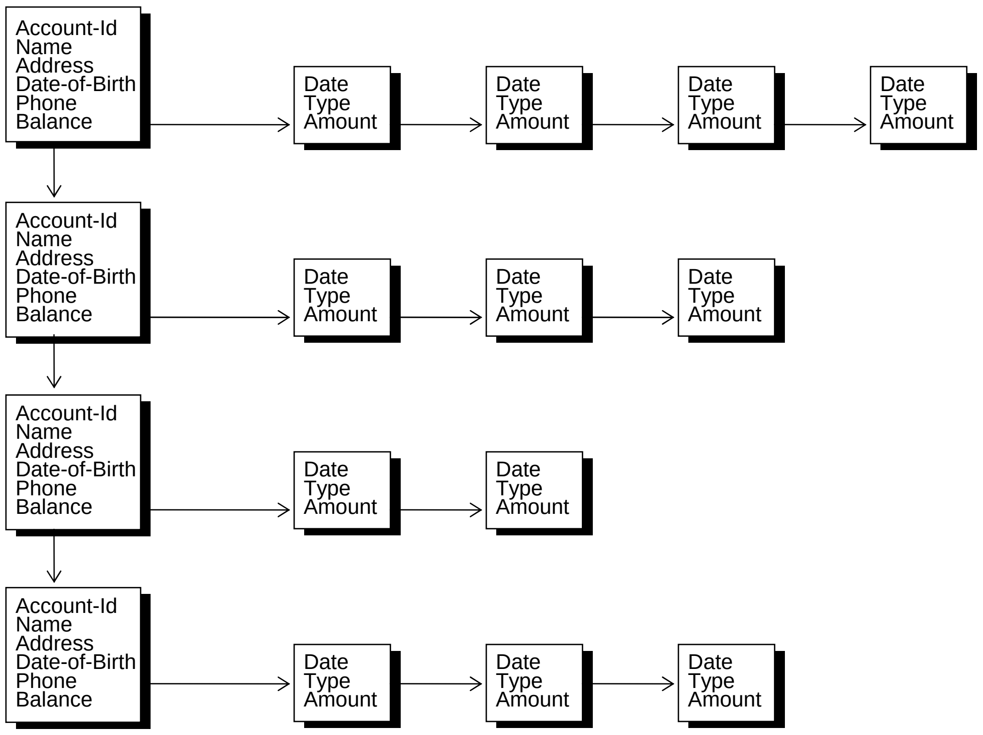
(a) **insert_trans**: This method appends a new transaction to a customer's linked-list of transactions. All transactions should appear in chronological order. Each should belong to some existing customers; otherwise, appropriate error message should be printed.

(b) **compute_balance**: At the end of every month, this method will be executed once to compute each customer's account balance. It either increases the balance if there is a credit transaction or decreases it if there is a debit transaction. We assume that the saving account can go negative but not for the checking account. It should also compute the interest based on a fix rate of 0.0003 or -0.002 per day for positive or negative daily balance for the saving account type. Hence there is no need to read interest rate again as have been done previously. For positive interest, it is based on the money that has been in the account for the last 24 hours (i.e. from midnight to midnight). For example, if you have $100 at hour 0 but you have withdrawn $50 at hour 1 and deposited back $50 at hour 2, you will be seen as having only $50 staying in your account for 24 hours at end of the day (hour 24). At hour 24, the money in your account will be $100 plus the daily interests computed according to $50.

For negative interest, it is based on the sum of money that you owe the bank on that day. If you borrow money from the bank, they will charge you interest even if you return the money 1 minute later. For example, if your saving account has $100 at hour 0 but you withdraw $200 at hour 22 and then deposit $1000 back at hour 23. You will not be paid any positive interest by today mid-night but will be charged for negative interest for borrow $100 from the bank for today.

One can assume that the transactions file is read only once per month. No need to worry about reading it the second time in a month. Current month and year are 09 (September) and 2007 respectively.

The negative interest rate (-0.002) is for overdrawing from the saving account (assuming that all the saving accounts here have the overdrawing ability), i.e. you are paying interest to the bank for borrowing their money. Hence, the deposit will become a larger negative number. No debit transaction is allowed for the account type of "fixed". You just need to display an error message for each debit transaction on "fixed" account type. Checking account cannot go negative. Again you just print an error message for transaction of this nature.



(c) Write a method **menu** that displays all the methods for test run. The lab markers can select method 1, 2, 3, 4, 5, 6 or Q to input data, display data, output data, delete record, update (compute balance), insert transaction and quit in any order. For command 2, it should display the customer data and the transaction data. For command 6, the program should prompt for a file name for the transaction records. Command 6 should be executed only once.

Although each of the above tasks is to be implemented as a single method, it may be necessary to create additional **private** sub-method(s) to handle portions of the method. This is especially true if the original method is too long or contains more than one functionally related group of statements. When you write a method, remember that this method is to work for all possible inputs. Not on just your test inputs. You must test for all conditions that might possibly arise; print out error messages as needed.

A text file **commands** has been created in the directory **/home/staff1/csc105/lab5/** to assist the development and testing of your program. Its contents and even the filenames will be changed (while still adhering to the same data format) during grading in order to test your programs' robustness. The objective is to ensure that you do not hardcode your program to work exclusively for the given sample customer records. Run your program as "**java Lab5 < commands**" where **Lab5** is the name of your class file and **Lab5** reads inputs from the standard input (i.e. keyboard). Check also the **readme** file for any last minute hints or changes.

Create a sub-directory called **cpe105/lab5** or **csc105/lab5** (depending on whether you are taking CPE105 or CSC105) in your home directory. Please note that the grader will only look into this sub-directory to find, compile and test run your program. You should provide a readme file that gives instructions to compile and run your program. You should not have irrelevant source files in the directory.

5. ERROR HANDLING

You can assume that the input file format is correct. No error checking is needed.

6. REPORT

(a) This lab is due six working days after your lab session. A working day includes Monday to Friday but excludes Saturday, Sunday and public holidays.

(b) For the report, submit hard copies of the readme file, program listing and **script** file(s) recording results of the testing of your program. (Type "man script" in Computing Lab 1 to learn more about script)

(c) The grading of your work will be based on the criteria listed in lab1 (c) and (d).

7. ACADEMIC HONESTY AND COLLABORATION

Cooperation is recommended in understanding various concepts and system features. But the actual solution of the assignments, the programming and debugging must be your individual work, except for what you specifically credit to other sources. (Your grade will be based on your own contribution.) For example, copying without attribution any part of someone else's program is plagiarism, even if you modify it and even if the source is a textbook. You can document the credit to other sources at the start of your program code listing. The University takes acts of cheating and plagiarism very seriously: first time violators may fail the coursework component of CPE105 / CSC105. Any wholly (or partly) copied (or being copied) programs will receive zero mark.

8. REFERENCES

[1] Text and reference books for CSC/CPE 105.

[2] http://java.sun.com/docs/index.html; http://java.sun.com/j2se/