CST8132: Object Oriented Programming

Lab 9 - Files

Requirements:

1. Our Bank Simulator will consist of information about customer Bank Accounts. (In this lab, we concentrate on Files)
2. The Bank Simulator consists of a dynamically allocated array of Bank Accounts of two types – either Savings accounts or Chequing accounts. A bank account consists of an account number (type: long), customer name and a double balance. A Savings account has two additional fields - a rate of interest and a minimum balance amount. A Chequing account has one additional field compared to a Bank Account – that is the monthly fee that is deducted.
3. The Simulator will give the user – presumably the user is a bank employee - a menu of choices of actions (ie this is **not** simulating a customer using a bank machine). These actions will include
    - adding a new Bank Account
    - displaying the information for a specific bank account
    - display all bank accounts
    - **loading Bank Account data from a file**
4. Your program should build a Bank "database" – for now this is a dynamically allocated array. The required size of this array is unknown – but we will implement it as a dynamically allocated array of objects of type Bank Account (instantiated with either a Savings Account or a Chequing Account object**). How you handle the size of this array is an important part of the assignment.**
5. The file format will be as follows for each line in the file:
    - a char – either s (for savings) or c (for chequing)
    - an int - Account Number – max 8 digits
    - a string – first name of customer
    - a string – last name of customer
    - a long - phone number
    - a string - email address
    - a double – Bank Balance
    - for savings account – a double interest rate (represented by percent – ie 1% is 0.01) followed by a double minimum balance OR for chequing account – a double – monthly fee
6. **Your program should handle ALL errors, it should never stop executing without giving a message about why it is doing so.** Do NOT ever use the command System.exit() – instead you should issue an appropriate message and return to the calling method (ultimately back to method main). If you are having the user enter data – do not continue until proper data has been entered. If you are reading from the file and encounter bad data, then exit gracefully with messages. This means all your methods which handle data should return a boolean – true if the data was ok – false if it was not – and all calls to these methods should check the return value.
7. You do NOT need to resize the array once you have made it should it become full. You should not allow more entries than you have space for – an error message is sufficient in this situation.
8. All Java conventions MUST be followed – ie upper and lower-case letters when appropriate, etc. All data members MUST be private (except in BankAccount class where they are protected).

Class Lab9:

- This class will contain method main which will contain the menu.

Class Person:

- This class will contain common data members for a person - firstName, lastName, phoneNumber, emailAddress.
- Methods:
    - constructor () to create accHolder object.
    - Required getters

Class BankAccount:

- This class will be the base class and contain the common data members for all Bank Accounts (ie accountNumber, accHolder, balance). Here, accHolder is an object of class Person.
- Methods:
    - toString():String – returns the data of the account formatted to display
    - addBankAccount(): boolean - prompts user to enter data for this object from keyboard - edits data, and doesn't allow user to continue with bad data
    - readFile(Scanner): boolean - uses Scanner object parameter to fill object with data- returns false if bad data is encountered, else returns true

Class SavingsAccount:

- This class will be inherited from BankAccount and contains the data members for a savings account (ie double interestRate, double minimumBalance)
- Methods:
    - toString(): String - returns the data of the account formatted to display
    - addBankAccount(): boolean - prompts user to enter data for this object from keyboard - edits data, and doesn't allow user to continue with bad data

Class ChequingAccount:

- This class will be inherited from BankAccount and contains the data members for a chequing account (ie double fee)
- Methods:
    - toString(): String - returns the data of the account formatted to display
    - addBankAccount(): boolean - prompts user to enter data for this object from keyboard - edits data, and doesn't allow user to continue with bad data

Class Bank:

- This class will contain the array of BankAccount objects (which are instantiated with either SavingAccount or ChequingAccount objects); You will need to keep two ints as well - a maxSize and numAccounts
- Methods:
    - constructor () – allocates default size of 1000
    - constructor (int) – parameter is size of array to be allocated
    - addAccount:boolean – success add or not; prompts user to enter data for an account which is added to array – either chequing or savings account is added if there is room
    - displayAccount() – String – prompts user to enter an account number to display, then returns data formatted to display or an error message. This should use toString() from BankAccount class.
    - printAccountDetails() - prints details of all accounts