CST8130: Data Structures --- Assign #1- Bank Simulator

Dynamically Allocated Array/Using Files/Exception Handling

DUE: Submission in Blackboard by Tuesday September 27 at 10PM SHARP!!

Problem Description:

In this assignment, you will complete an object-oriented version of the software managing Bank customers' Bank Accounts. Note you may **NOT** use the **ArrayList** class in Java for this assignment. We will use this assignment as the base for many following assignments. You must use the classes as named in this assignment, but you can use different methods and data member names as long as your solution follows all Object Oriented Principles.

Requirements:

1. Our Bank Simulator will consist of information about customer Bank Accounts. I have simplified the requirements for this assignment to set boundaries on your time.
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 (up to 8 digits long), customer name and a double balance. A Savings account has two additional fields - a rate of interest and a minimum balance amount (at the end of the month, the customer is given interest based on their account balance as long as the account balance is more than the 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
   - updating the balance for a specific bank account (withdrawal or deposit)
   - running the monthly update on all accounts
   - loading Bank Account data from a file
   - (optional) saving Bank Account data to 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 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 capital and lower case letters when appropriate, etc. All data members MUST be private (except in BankAccount class where they are protected).

9. Do not use ONLY the get/set design pattern in this assignment. In order to emphasize OOP, I would like all processing of data members in a class to be handled in the class – and the get/set pattern allows this to be broken. See description for methods for each class below which will help this requirement.

Hints:

- Start this assignment WELL BEFORE it is due. I do not accept late assignments. Check the submission requirements and make sure you follow them.
- Work incrementally. Do not write more than approx. 20 lines of code without testing your code. EVER. My recommendation is that you start with the code for the menu in method main…then add each menu selection one at a time. I have listed the menu selections in order that would be easiest to implement!!
- Do not just blindly follow my descriptions for each of the methods in the class. I suggest you start with Assign1 class and write the loop for the menu. Then, add one menu item processing at a time, adding the appropriate methods to each class.
- Enjoy ☺

Submission:

You must submit to the assignment link in Blackboard by the due date and time a zip file (named YourLastnameYourFirstNameAssign1) containing:

- all source code – ie .java files (Note – I may choose to re-compile your program….so all code must be available to me) with header information (containing your name, course and section info, date, description of class, description of each data member and method in class)
- all .class files
- Your test plan in either .docx or .xls format

Failure to provide any of the above will have an effect on your grade for this assignment. Marking guide will be published shortly.

Class Assign1:

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

Class BankAccount:

- This class will be the base class and contain the common data members for all Bank Accounts (ie accountNumber, firstName, lastName, balance)
- 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 allower user to continue with bad data
  - isEqual (int): boolean - compares int parameter to account number in object and returns true/false appropriately
  - isGreater (double): boolean - compares the double parameter to the balance in object and returns true/false appropriately
  - updateBalance (double) - updates the balance in the object by the parameter amount
  - monthlyUpdate() - processes the object with monthly update (empty for base class)
  - 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 allower user to continue with bad data
  - monthlyUpdate() - processes the object with monthly update of adding interest (as long as bank balance is more than minBalance, else displays error message
  - readFile(Scanner): boolean - uses Scanner object parameter to fill object with data- returns false if bad data is encountered, else returns true

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
  - monthlyUpdate() - processes the object with monthly update of withdrawing the fee (as long as bank balance is more than fee, else displays error message)
- readFile(Scanner): boolean - uses Scanner object parameter to fill object with data- returns false if bad data is encountered, else returns true

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:
  - default constructor () – allocates default size of 1000
  - initial 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
  - toString() – String – prompts user to enter an account number to display, then returns data formatted to display or an error message
  - update():String – prompts user to enter which account number to update, and by how much and then updates the balance appropriately – returns success message or error message
  - toFind () :int – prompts user to enter which account number they wish to find and returns array index of where it is found otherwise returns -1
  - monthlyUpdate() – process through each current account in the array and updates the balance appropriately
  - readFile():boolean – prompts user for name of file to process, opens that file , then reads through the file and adds accounts to the array if there is room. Returns false if any bad data is encountered, else returns true
  - openFile():Scanner – returns the Scanner object if a file (name input by user) is opened, else returns null

Sample Output -THIS IS NOT A TEST PLAN!!:

```
Enter your choice:  a - add new
account; d - display an account;
u - update balance on account; m -
run month end update; f - enter
info from file;  q - quit: a

Enter an s for Savings Account or c
for Chequing Account: s
Enter account number: 123123
Enter customer first name: Linda
Enter customer last name: Crane
Enter balance: 500.00
Enter monthly minimum balance:
250.0
Enter monthly interest rate: .01

Enter your choice:  a - add new
account; d - display an account;
u - update balance on account; m -
run month end update; f - enter
info from file;  q - quit: d

Enter account number to find:
123123
Account:123123 Linda Crane Balance
$500.0 minimum Balance $250.0
interest rate $0.01

Enter your choice:  a - add new
account; d - display an account;
u - update balance on account; m -
run month end update; f - enter
info from file;  q - quit: a

Enter an s for Savings Account or c
for Chequing Account: c
Enter account number: 222222
Enter customer first name: Donald
Enter customer last name: Duck
Enter balance: 1000.00
Enter monthly fee: 10

Enter your choice:  a - add new
account; d - display an account;
u - update balance on account; m -
run month end update; f - enter
info from file;  q - quit: d

Enter account number to find:
222222
Account:222222 Donald Duck Balance
$1000.0 monthly fee $10.0


Enter your choice:  a - add new
account; d - display an account;
u - update balance on account; m -
run month end update; f - enter
info from file;  q - quit:  u

Enter account number to find:
222222
Enter amount to update (negative
for withdrawal, positive for
deposit): -300
Account updated

Enter your choice:  a - add new
account; d - display an account;
u - update balance on account; m -
run month end update; f - enter
info from file;  q - quit: d

Enter account number to find:
222222
Account:222222 Donald Duck Balance
$700.0 monthly fee $10.0

Enter your choice:  a - add new
account; d - display an account;
u - update balance on account; m -
run month end update; f - enter
info from file;  q - quit:  m

Adding interest of 0.01% to account
123123.  New balance is $ 505.0
Deducting fee of $10.0 from account
222222.  New balance is $ 690.0

Enter your choice:  a - add new
account; d - display an account;
u - update balance on account; m -
run month end update; f - enter
info from file;  q - quit: f


Enter name of file to process:
c:\bank.txt
Enter your choice:  a - add new
account; d - display an account;
u - update balance on account; m -
run month end update; f - enter
info from file;  q - quit: d

Enter account number to find:
45645645
Account:45645645 Linda Crane
Balance $2000.0 monthly fee $10.0
```