COEN 275: Object Oriented Analysis, Design, and Programming
Winter 2017
Individual Assignment 0
Due: January 22nd, @11:59 PM

Purpose:
This assignment is aimed at helping students who are not familiar with programming in Java practice writing code in Java.

The program requires the use of the following Java concepts:

Defining Variable Types
- integer
- double
- String

Control Structure
- if-else
- for, “foreach”
- while
- switch

Console I/O

File I/O

Classes
- Constructor
- Private methods
- Public methods
- Private data members

Java ArrayLists

Try Catch blocks / Exceptions

If you are already familiar with these concepts, you may choose to skip this assignment.

It is intended only to be practice, and as such, has been designed to have much more specific requirements, and a heavier emphasis on Java style than other Assignments will.

Assignment Overview:
The objective of this assignment is to simulate a bank. Your program will read account information from a file, and create account objects with the contained specifications. Afterwards, prompt the user with a menu with available actions. Upon closing, the banking application will write all updated information to the original file.

Bank DataBase file:
The Bank Database file will contain one account's information stored in a comma separated variable (csv) format on each line. There is no header line, only data. The variables contained will be (In order):

- Account Number
- PIN Number
- Savings Balance
- Checking Balance (Only if they have a checking account)
- Name of account holder

Ex:
An account with:

- Account Number = 1
- PIN Number = 1111
- Savings Balance = 1.11
- Checking Balance = 2.22
- Name of account holder = Abe

Will be represented by the line:
1,1111,1.11,2.22,Abe

An account without a checking account will be represented by the line:
1,1111,1.11,Abe

Program Design
Your program must have three classes; BankAccount, Bank, and BankTester.

<u>BankAccount Class</u>:
The BankAccount class stores the information about a bank account. It will need the following data members:

- Account Number (integer)
- PIN Number (integer)
- Savings Balance (double)
- Checkings Balance (double)
- Account holder name (String)
- Whether it has a checkings account or not (Boolean)

To serve its purpose as the storage class for BankAccount information, the BankAccount class requires a couple of methods. Users of the BankAccount class should be able to get the account number, account holder's name, and the checkings and savings balances from the class as well as set new balances for the checkings and savings accounts. In addition, the user should have methods for checking whether or not the BankAccount has a checkings account, checking whether or not the pin entered is correct or not, and printing the information in the class as comma separated string.

Getters Specification:
Getters should be public methods that return their respective data values. The method used to get the checkings account should verify that the user has a checkings account. If not, it should throw an "UnsupportedOperationException"

Setters Specification:
Setters should be public methods that take in a double as an argument, and sets the new value of the balance. The setter for the checking account should verify that the BankAccount has a checkings account. If the BankAccount requested does not, it should also throw an "UnsupportedOperationException"

Has a Checking Account Specification:
This public method should return a Boolean stating whether or not the bank account has a checkings account

PIN Verificaiton Specification:
This public method should take in an integer, and return a Boolean stating whether or not the PIN entered in matches the PIN stored in the account.

Printing CSV Specification:
The CSV printing method should return a String with the format of lines in the Bank DataBase file.

Setting whether or not a BankAccount has a checkings account should be determined through overloaded constructors.

Bank Class:
The Bank class manages the functions of the Bank, acting as an intermediary between outside users and BankAccounts. It should contain the following data members:

- A list of BankAccount objects (ArrayList<BankAccount>)
- Location of Bank DataBase file (final String)

As the intermediary between outside classes and BankAccounts the Bank must provide security with a login check, give navigate the user through their available actions with menu prompts, as well as reading and writing bank account information into the Bank DataBase file.

Login Specification:
The public login method should prompt the user for a bank account number, and search through the ArrayList to find a matching BankAccount. If no account matches, return control back to the BankTester. Once the account is acquired, the Bank should prompt the user for a password, and use the password verification method of the BankAccount class to check. Upon successfully logging in, the user should be greeted by their name. The login should also keep track of the number of attempts to login, and after three attempts, return control back to BankTester.

Menu Specification:
To navigate the user through available actions, menus should be provided. These should use switch statements to direct the flow of the program to different branches based off of user input. If the user has a checkings account, then the user should be prompted to enter which account should be accessed. Otherwise, your program should use the information from a savings account, and proceed to asking the user to deposit or withdraw. This should be a Private method.

Deposit and Withdraw Specification:
Instead of another menu to determine whether the user would like to deposit or withdraw, the user should be prompted to enter a positive number for deposit, and a negative number for withdrawals. That number will then be added to their current account balance. After their balance has been changed, return control back to BankTester. This should be a private method.

Read DB Specifications:
The Bank class should keep information about the location of the Bank DataBase file stored in the class as a constant (final in Java Terminology). When the class is initiated, it should read the file, create the bank accounts, and store them in the BankAccount list.

Write to DB Specifications:
Writing to the DB file should be contained in a method by itself. It should read through the BankAccount list, and invoke each BankAccount's method to generate a csv, writing each String to the file. It should only be invoked when the user selects the "Save and Close" option from the BankTester Class. This should be a public method.

BankTester Class:
The BankTester class runs the Bank class. It will contain a main method that instantiates a Bank Object, and prompts the user to "Login" or "Save and Quit". These options will invoke

the Bank's login and write to DB methods respectively. The class should continuously prompt the user to with the two options while the user continues to pick the login option. Once the "Save and Quit" option is selected, the program will exit

Example:

Bank DataBase before the transaction:
Before: 1,1111,1.11,2.22,Abe

Program Output (Bolded text are user inputs):
Options:
1) Login
2) Save and Quit
Choice: **1**

Enter in Bank Account Number: **1**
Enter in Pin: **1111**

Hello Abe, which account would you like to access:
1) Checkings
2) Savings
Choice: **1**

Your Checkings account has a balance of $2.22
How would you like to proceed?
(Enter in positive number to deposit, negative number to withdraw)
Amount: **100**

Thank you, your new Checkings balance is $102.22
Returning to login screen

Options:
1) Login
2) Save and Quit
Choice: **2**

Bank DataBase after the transaction:
After: 1,1111,1.11,102.22,Abe

Code Requirements:
Your program should have similar output to the example above.

Your code must adhere to proper Java Style. You are welcome to use a Java Style you are comfortable with, as long as you are ***consistent*** in your code. If you do not have a Java Style, a good starting point is the Google Java Style Guide. Ample commenting is welcome and encouraged. As a general rule of thumb, someone reading your code should be able to determine the function of any method within 30 seconds of reading it.

Submission Guidelines:
Your submission must contain the following files:

- Java Source Files
- Combined Source File
- README.txt
- Any other files required to run your program not listed above

Combined Source Files:
You must include with your submission a file that contains the contents of all your source files concatenated together. The file should be named firstInitialLastName.txt

README.txt File:
You must also include with your submission a README file. This file is used to help debug setup related issues. The README file should use the following format:

Name: <Your first and last name>
JDK Used: <JDK Provider> <Version>
IDE Used: <IDE and Version>
Main File: <Filename of file that should be run>
Load File: <Instructions on loading in the move file, including the filename of where you hardcoded the location if applicable.>
Other Instructions: <Optional special instructions specific to your program>

All files EXCEPT the Combined Source file should be compressed into a .zip file. There will be two Assignments created in Camino, one for your .zip file, one for your Combined Source file. Submit both the .zip file and the Combined Source File to their respective DropBoxes.

Grade Breakdown:

Functionality	70%
Style	30%