Assignment 4 - Bank Transactions

# Introduction

Merit America Bank is pleased with the current progress with the software system and has some additional requirements they would like you to implement. The additional requirements might require new code or amending/refactoring previously written code.

In this assignment, you will be working in a pair with a fellow cohort colleague using Pair Programming.

# Requirements

Merit America Bank requires the software system to keep track of the history of transactions in each bank account (e.g. deposits and withdrawals).

The bank has a policy that transactions over $1,000 must be reviewed by the fraud team. In the future, a fraud team will monitor a fraud queue for transactions that require review.

In addition to depositing and withdrawing funds, the bank requires functionality to enable funds to be transferred between two accounts. Similar constraints follow that:

1. Transfer amount should be less than balance in source account
2. Transfer amount should be a positive number
3. Transfer amount exceeding $1,000 must be reviewed by the fraud team

The bank is concerned about the readability, reliability, and usability of the software system and requires that proper exception handling is used instead of booleans such that it is clear what failures are occurring.

The bank is also concerned that future Junior Software Developers joining the team might be able to instantiate a BankAccount. The requirement is that BankAccount be an abstract class and to use abstract classes and/or interfaces for abstract entities.

Finally, the bank is concerned that the Math.pow() method may have a bug in certain situations. To alleviate this concern, the bank requires that all methods that utilize Math.pow() be rewritten to use recursion instead of using Math.pow() when computing future values.

## Details

Create the following classes with the following methods (note that the state variables are not spelled out here, you will need to create these as needed by the various accessor methods):

1. public class NegativeAmountException extends Exception
2. public class ExceedsAvailableBalanceException extends Exception
3. public class ExceedsCombinedBalanceLimitException extends Exception
4. public class ExceedsFraudSuspicionLimitException extends Exception
5. public abstract class Transaction
   1. public BankAccount getSourceAccount()
   2. public void setSourceAccount(BankAccount sourceAccount)
   3. public BankAccount getTargetAccount()
   4. public void setTargetAccount(BankAccount targetAccount)
   5. public double getAmount()
   6. public void setAmount(double amount)
   7. public java.util.Date getTransactionDate()
   8. public void setTransactionDate(java.util.Date date)
   9. public String writeToString()
   10. public static Transaction readFromString(String transactionDataString)
   11. public abstract void process() throws NegativeAmountException, ExceedsAvailableBalanceException, ExceedsFraudSuspicionLimitException
   12. public boolean isProcessedByFraudTeam()
   13. public void setProcessedByFraudTeam(boolean isProcessed)
   14. public String getRejectionReason()
   15. public void setRejectionReason(String reason)
6. public class DepositTransaction extends Transaction
   1. DepositTransaction(BankAccount targetAccount, double amount)
7. public class WithdrawTransaction extends Transaction
   1. WithdrawTransaction(BankAccount targetAccount, double amount)
8. public class TransferTransaction extends Transaction
   1. TransferTransaction(BankAccount sourceAccount, BankAccount targetAccount, double amount)
9. Public class FraudQueue
   1. FraudQueue()
   2. public void addTransaction(Transaction transaction)
   3. public Transaction getTransaction()

Add the following methods:

1. MeritBank
   1. public static double recursiveFutureValue(double amount, int years, double interestRate)
      1. Existing futureValue methods that used to call Math.pow() should now call this method
   2. public static boolean processTransaction(Transaction transaction) throws NegativeAmountException, ExceedsAvailableBalanceException, ExceedsFraudSuspicionLimitException
      1. If transaction does not violate any constraints, deposit/withdraw values from the relevant BankAccounts and add the transaction to the relevant BankAccounts
      2. If the transaction violates any of the basic constraints (negative amount, exceeds available balance) the relevant exception should be thrown and the processing should terminate
      3. If the transaction violates the $1,000 suspicion limit, it should simply be added to the FraudQueue for future processing
   3. public static FraudQueue getFraudQueue()
   4. public static BankAccount getBankAccount(long accountId)
      1. Return null if account not found
2. abstract BankAccount
   1. public void addTransaction(Transaction transaction)
   2. public List<Transaction> getTransactions()

Amend the following methods:

1. AccountHolder:
   1. CheckingAccount addCheckingAccount(double openingBalance) throws ExceedsCombinedBalanceLimitException
      1. If combined balance limit is exceeded, throw ExceedsCombinedBalanceLimitException
      2. Should also add a deposit transaction with the opening balance
   2. CheckingAccount addCheckingAccount(CheckingAccount checkingAccount) throws ExceedsCombinedBalanceLimitException
      1. If combined balance limit is exceeded, throw ExceedsCombinedBalanceLimitException
      2. Should also add a deposit transaction with the opening balance
   3. SavingsAccount addSavingsAccount(double openingBalance) throws ExceedsCombinedBalanceLimitException
      1. If combined balance limit is exceeded, throw ExceedsCombinedBalanceLimitException
      2. Should also add a deposit transaction with the opening balance
   4. SavingsAccount addSavingsAccount(SavingsAccount savingsAccount) throws ExceedsCombinedBalanceLimitException
      1. If combined balance limit is exceeded, throw ExceedsCombinedBalanceLimitException
      2. Should also add a deposit transaction with the opening balance
   5. CDAccount addCDAccount(CDOffering offering, double openingBalance)
      1. Should also add a deposit transaction with the opening balance
   6. CDAccount addCDAccount(CDAccount cdAccount)
      1. Should also add a deposit transaction with the opening balance
2. MeritBank
   1. static boolean readFromFile(String fileName)
      1. Should also read BankAccount transactions and the FraudQueue
   2. static boolean writeToFile(String fileName)
      1. Should also write BankAccount transactions and the FraudQueue

This is an example data file for the scenario described above:

11

3

1,0.018

3,0.019

5,0.02

2

Doe,,John,1234567890

1

1,900,0.0001,01/01/2020

2

-1,1,1000.0,01/01/2020

1,2,100,01/03/2020

2

2,10000,0.01,01/02/2020

3

-1,2,5000,01/02/2020

-1,2,-2000,01/03/2020

-1,2,7000,01/04/2020

3,15100,0.01,01/02/2020

2

-1,3,15000,01/02/2020

1,2,100,01/03/2020

0

Doe,S,Jane,0987654321

2

4,100,0.0001,12/01/2019

1

-1,4,100,12/01/2019

5,200,0.0001,12/15/2019

1

-1,5,200,12/15/2019

3

6,15000,0.01,12/01/2019

1

-1,6,15000,12/01/2019

7,2100,0.01,12/15/2019

1

-1,7,2100,12/15/2019

8,9000,0.01,01/01/2020

1

-1,8,9000,01/01/2020

2

9,5000,0.02,01/01/2020,5

1

-1,9,5000,02/01/2020

10,5000,0.025,01/01/2020,10

1

-1,10,5000,01/01/2020

Explanation:

|  |  |
| --- | --- |
| 11 | Next account number is 11 |
| 3 | 3 CD offerings |
| 1,0.018 | 1st CD offering: 1 year term at 1.8% |
| 3,0.019 | 2nd CD offering: 3 year term at 1.9% |
| 5,0.02 | 3rd CD offering: 5 year term at 2.0% |
| 2 | 2 account holders |
| Doe,,John,1234567890 | 1st account holder: John Doe (no middle name) with SSN: 1234567890 |
| 1 | 1st account holder has 1 checking account |
| 1,1000,0.0001,01/01/2020 | Checking account #1 has $1,000 balance, 0.01% interest rate, and was opened on 1/1/2020 |
| 2 | Checking account #1 has 2 transactions |
| -1,1,1000.0,01/01/2020 | Initial deposit of $1,000 on 1/1/2020 |
| 1,3,100,01/03/2020 | Transfer of $100 to account 3 on 1/3/2020 |
| 2 | 1st account holder has 2 savings accounts |
| 2,10000,0.01,01/02/2020 | 1st savings account #2, has $10,000 balance, 1% interest rate, and was opened on 1/2/2020 |
| 3 | Savings account #2 has 3 transactions |
| -1,2,5000,01/02/2020 | Initial deposit of $5,000 on 1/2/2020 |
| -1,2,-2000,01/03/2020 | Withdrawal of $2,000 on 1/3/2020 |
| -1,2,7000,01/04/2020 | Deposit of $7,000 on 1/4/2020 |
| 3,15000,0.01,01/02/2020 | 2nd savings account #3, has $15,000 balance, 1% interest rate, and was opened on 1/2/2020 |
| 2 | Savings account #3 has 2 transactions |
| -1,3,15000,01/02/2020 | Initial deposit of $15,000 on 1/2/2020 |
| 1,3,100,01/03/2020 | Transfer of $100 from account 1 on 1/3/2020 |
| 0 | 1st account holder does not have any CD accounts |
| Doe,S,Jane,0987654321 | 2nd account holder: Jane S Doe with SSN 0987654321 |
| 2 | 2nd account holder has 2 checking accounts |
| 4,100,0.0001,12/01/2019 | 1st checking account #4, current balance of $100 with 0.01% interest rate opened on 12/1/2019 |
| 1 | Account #4 has 1 transaction |
| -1,4,100,12/01/2019 | Initial deposit of $100 on 12/1/2019 into account #4 |
| 5,200,0.0001,12/15/2019 | 2nd checking account #5, current balance of $200 with 0.01% interest rate opened on 12/15/2019 |
| 1 | Account #5 has 1 transaction |
| -1,5,200,12/01/2019 | Initial deposit of $200 on 12/1/2019 into account #5 |
| 3 | 2nd account holder has 3 savings accounts |
| 6,15000,0.01,12/01/2019 | 1st savings account #6, current balance of $15,000 with 1% interest rate opened on 12/1/2019 |
| 1 | Account #6 has 1 transaction |
| -1,6,15000,12/01/2019 | Initial deposit of $15,000 on 12/1/2019 into account #6 |
| 7,2100,0.01,12/15/2019 | 2nd savings account #67, current balance of $2,100 with 1% interest rate opened on 12/15/2019 |
| 1 | Account #7 has 1 transaction |
| -1,7,2100,12/15/2019 | Initial deposit of $2,100 on 12/15/2019 into account #7 |
| 8,9000,0.01,01/01/2020 | 3rd savings account #8, current balance of $9,000 with 1% interest rate opened on 1/1/2020 |
| 1 | Account #8 has 1 transaction |
| -1,8,9000,01/01/2020 | Initial deposit of $9,000 on 1/1/2020 into account #8 |
| 2 | 2nd account holder has 2 CD accounts |
| 9,5000,0.02,01/01/2020,5 | 1st CD account #9, opening balance of $5,000 with 2% interest rate and a 5 year term, opened on 1/1/2020 |
| 1 | Account #9 has 1 transaction |
| -1,9,5000,02/01/2020 | Initial deposit of $5,000 on 2/1/2020 into account #9 |
| 10,5000,0.025,01/01/2020,10 | 2nd CD account #10, opening balance of $5,000 with 2.5% interest rate and a 10 year term, opened on 1/1/2020 |
| 1 | Account #10 has 1 transaction |
| -1,10,5000,01/01/2020 | Initial deposit of $5,000 on 1/1/2020 into account #10 |
| 2 | Fraud Queue has 2 transactions pending review |
| 2,4,5000,01/05/2020 | Transfer of $5,000 from account #2 to account #4 on 1/5/2020 |
| -1,2,2000,01/06/2020 | Deposit of $2,000 into account #2 on 1/6/2020 |

# Instructions

1. Visit this [GitHub repository](https://github.com/MeritAmerica/assignment4) and follow the instructions to work with the provided starter code:
   1. Fork the repository
   2. Clone your fork (consider adding your partner as a collaborator to your forked repository)
   3. Import the project into Eclipse
   4. Run the application in Eclipse
   5. Run the test cases in Eclipse
2. Complete the assignment requirements such that all test cases are passing.
   1. Note: you may copy your existing files from Assignment 3 as a starting point.
3. Upload your zipped ‘assignment4’ folder to [HackerRank](https://www.hackerrank.com/tests/17k8h2gero8/43d79ef8fa941ba685d2eb7298e67ba3?try_test=true) for submission.

# Expectations

1. Use try-with-resources when reading from a file
2. Functionality from prior assignments should still work except where amended for this assignment’s requirements
3. Write unit tests for all of the requirements in this assignment (think about edge cases)
4. Code should be readable
   1. For example: use meaningful variable names, use proper naming conventions, properly indent code, comment your code
5. Use the “this” keyword to reference instance variables/methods