Project 14

Bank Accounts

Part 1: A Bank Account Class

1. File *Account.java* contains a partial definition for a class representing a bank account. Study it to see what methods it contains.

   ****Note that you won't be able to test your methods until you modify *ManageAccounts.java* in question #2.

   a. Fill in the code for method `toString`, which should return a string containing the name, account number, and balance for the account. For example:

   ```
   Name: Sally Fields
   Acct #: 1111
   Balance: 940.0
   ```

   b. Fill in the code for method `chargeFee`, which should deduct a service fee from the account.
   c. Modify `chargeFee` so that instead of returning void, it returns the new balance. Note that you will have to make changes in two places.
   d. Fill in the code for method `changeName` which takes a string as a parameter and changes the name on the account to be that string.
   e. Create a method: `getAcctNumber`

2. File *ManageAccounts.java* contains a shell program that uses the Account class above. Complete it as indicated by the comments.

3. Modify `ManageAccounts` so that it prints the balance after the calls to `chargeFees`. Instead of using the `getBalance` method like you did after the deposit and withdrawal, use the balance that is returned from the `chargeFees` method. You need to modify the `chargeFees` method. Then you can either store it in a variable and then print the value of the variable, or embed the method call in a `println` statement.

Part 2: Opening and Closing Accounts

Write the following additional code to *Account.java.*

1. Suppose the bank wants to keep track of how many accounts exist.

   a. Declare a private static integer variable `numAccounts` to hold this value. Like all instance and static variables, it will be initialized (to 0, since it's an `int`) automatically.
   b. Add code to the constructor to increment this variable every time an account is created.
   c. Add a static method `getNumAccounts` that returns the total number of accounts. Think about why this method should be static – its information is not related to any particular account.
   d. File *TestAccounts1.java* contains a basic program that creates the specified number of bank accounts then uses the `getNumAccounts` method to find how many accounts were created. Use it to test your modified `Account` class.

2. Add a method `void close()` to your `Account` class. This method should close the current account by appending "CLOSED" to the account name and setting the balance to 0. (The account number should remain unchanged.) Also decrement the total number of accounts.

3. Add a static method `Account consolidate(Account acct1, Account acct2)` to your `Account` class that creates a new account whose balance is the sum of the balances in `acct1` and `acct2` and closes `acct1` and `acct2`. The new account should be returned. Two important rules of consolidation:

   - Only accounts with the same name can be consolidated. The new account gets the name on the old accounts but a new account number.

- Two accounts with the same number cannot be consolidated. Otherwise this would be an easy way to double your money!

Check these conditions before creating the new account. If either condition fails, do not create the new account or close the old ones; print a useful message and return null.

4. In *TestAccounts2.java,* create three new accounts that has the user enter the names of the people and default the initial balance to $100 each. Print all three accounts. Try to consolidate the first account with itself (this should fail). Then print this account to verify that nothing changed. Then close the first account, and try to consolidate the second and third into a new account. Now print the accounts again, including the consolidated one if it was created. You'll need to run the code twice (once when the consolidation works and once when it doesn't).

Part 3: Counting Transactions

Now modify *Account.java* to keep track of the total number of deposits and withdrawals (separately) for each day, and the total amount deposited and withdrawn. Write code to do this as follows:

1. Add four private static variables to the `Account` class, one to keep track of each value above (number and total amount of deposits, number and total of withdrawals). Note that since these variables are static, all of the Account objects share them. This is in contrast to the instance variables that hold the balance, name, and account number; each Account has its own copy of these. Recall that numeric static and instance variables are initialized to 0 by default.

2. Add public static methods to return the values of each of the variables you just added. Use the following identifiers for the methods: `getNumDeposits`, `getAmtWithdrawals`, `getAmtDeposits`, and `getNumWithdrawals`.

3. Modify the `withdraw` and `deposit` methods to update the appropriate static variables for each withdrawal and deposit.

4. File *ProcessTransactions.java* contains a program that creates and initializes two `Account` objects and enters a loop that allows the user to enter transactions for either account until asking to quit. After the loop, print the total number of deposits and withdrawals and the total amount of each. You will need to use the `Account` methods that you wrote above. Test your program.

Part 4: Transferring Funds

Then write the following additional code to *Account.java*:

1. Add a method `public void transfer(Account acct, double amount)` to the Account class that allows the user to transfer funds from one bank account to another. If `acct1` and `acct2` are `Account` objects, then the call `acct1.transfer(acct2,957.80)` should transfer $957.80 from `acct1` to `acct2`. Be sure to clearly document which way the transfer goes!

2. In *TransferTest.java* create two bank account objects and enters a loop that does the following:
   a. Asks if the user would like to transfer from account1 to account2, transfer from account2 to account1, or quit.
   b. If a transfer is chosen, asks the amount of the transfer, carries out the operation, and prints the new balance for each account.
   c. Repeats until the user asks to quit, then prints a summary for each account.

3. Add a static method to the `Account` class that lets the user transfer money between two accounts without going through either account. You can (and should) call the method `transfer` just like the other one – you are overloading this method. Your new method should take two `Account` objects and an amount and transfer the amount from the first account to the second account. The signature will look like this:

```
public static void transfer(Account acct1, Account acct2, double amount)
```

   Modify your `TransferTest` class to use the static transfer instead of the instance version.