Review Topic: *static* Variables and Methods

A Bank Account Class

(Source: Bloss, A and Ingram, N.J., Lab Manual Java Software Solutions, Pearson, 2009)

1. *Account.java* contains a partial definition for a class representing a bank account. Study it to see what methods it contains. Then complete the Account class as described below. Note that you won't be able to test your methods until you write *ManageAccounts* 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.
   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.
2. Program *ManageAccounts.java* contains a skeleton program that uses the Account class above. Create it in your project, and 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 can either store it in a variable and then print the value of the variable, or embed the method call in a *println()* statement.

Consolidating Accounts

Modify *Account.java* by adding the following code:

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 *TestAccounts.java* contains a simple program that creates the specified number of bank accounts then uses the *getNumAccount()s* method to find how many accounts were created. Save it to your directory, then 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. You may use either of the consolidated accounts' account number for the new account number.

   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. Write another test program *TestConsolidatedAccounts.java* that prompts for and reads in three names and creates an account with an initial balance of $100 for each. Print the three accounts, 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.

Counting Transactions

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 methods to return the values of each of the variables you just added, e.g., public static int getNumDeposits().
3. Modify the withdraw and deposit methods to update the appropriate static variables at 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. Modify this program as follows:

- 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.
- Imagine that this loop contains the transactions for a single day. Embed it in a loop that allows the transactions to be recorded and counted for many days. At the beginning of each day print the summary for each account, then have the user enter the transactions for the day. When all of the transactions have been entered, print the total numbers and amounts (as above), then reset these values to 0 and repeat for the next day. Note that you will need to add methods to reset the variables holding the numbers and amounts of withdrawals and deposits to the Account class. Think: should these be static or instance methods?

Review Topic: Parameter Passing

Changing People

ChangingPeople.java, given below, is a program that illustrates parameter passing. The program uses *Person* objects defined in *Person.java.* Do the following:

1. Trace the execution of the rogram and determine what is printed by the program.
2. Compile and run the program to see if your trace was correct.
3. Modify the *changePeople* method so that it does what the documentation says it does, that is, the two Person objects passed in as actual parameters are actually changed. Print out a copy of your source codes and demonstrate your running program to the lab instructor.