Software Development

Mini Assignment 2
Due: February 4, 2018

This mini assignment practices the software techniques covered in lectures #2 through #5, all the lectures on "Well-Designed Objects". Primarily this is a Java programming assignment that asks you to create a solution that follows best practices: generalized code, abstraction, information hiding, code reuse, and taking advantage of similarity. Added to this, we continue to use the skills we practiced from Mini 1.

The problem:

We wish to create a well-designed bank account application. It is up to you to determine the optimal number of objects and the naming of those objects. It is up to you to determine the optimal API, abstraction technique, and the use of best practices as covered in the lectures. I will only give you some basics. YOU will have to design and implement an optimal application.

The main method for your application will be housed in the Bank.java file. All the other files are up to you but must be based on optimality and best practices as covered in class.

The application can store an infinite number of bank accounts and customers. There are two kinds of bank accounts: checking and savings. Each bank account has a single customer as its owner.

A customer has a name, a unique customer number, and a discount percentage.

A bank account has a balance. Operations that can be performed are: finding out the balance, making a deposit, making a withdrawal, and transferring money to another account (both accounts must be owned by the same customer). Creating a bank account also initializes the balance to an amount determined by the customer.

A checking account charges for withdrawals. The default charge is $1 but it is modified by the customer's discount percentage.

A savings account only permits withdrawals greater than or equal to $1000. Every deposit is awarded $1 but it is increased based on the customer's discount percentage.

The main method implements a menu with the following options: (a) create customer, (b) create bank account, (c) Get balance, (d) Deposit, (e) Withdrawal, (f) Make a transfer, (g) Quit.

The UI is text-based and does not need to be pretty, but it should be easy to read.

Make sure your application is easy to use.

Make sure you protect your objects.

Do not implement a testing class to validate your code.

You are creating a well-designed program. I expect to see optimality, pretty code, and comments as documentation.

WHAT TO HAND IN

- Bank.java
- A minimal number of other .java files belonging to the application
- You can zip everything into a single file

HOW IT WILL BE GRADED

For your assignment to be graded your program (a) must run, (b) did not use tools, and (c) followed the assignment instructions. You are doing this assignment on your own.

- +5 – Optimality (Big Oh)
- +5 – Optimality (memory)
- +5 – Simplicity of solution algorithms
- +5 – Correctness of the application
- +5 – Well written code that is easy to read (see lecture 2 for examples)
- +5 – Comments as documentation (see lecture 2 for examples)
- +5 – Application is easy to use
- +5 – Uses encapsulation, information hiding, generalized code, code reuse, code similarity, APIs