COSC 201 – Assignment #1
Fall 2017

Objective: Write a program that will take in a text filename via the command line. That file will contain a list of accounts that will need to be placed in a database based on the type of account, one per line, formatted name, account type, current balance, class specific information. See the implementation details below in regards to the database and the other specific classes that you'll need.

Implementation Details:

- A *Bank* class, which stores an ArrayList that can contain accounts of all types including savings and checking accounts, some of which are interest bearing and some of which are not. Bank contains a method called *totalAssets* that returns the sum of all the balances in all the accounts. It also contains a method called *addInterest* that invokes the *addInterest* method for all the accounts in the bank that are interest-bearing. You should also create a method to *openAccount* and *closeAccount*. *openAccount* should take in information from the user on the amount of the initial deposit and what kind of account, and *closeAccount* should get the account number from the user. Finally, the *toString* method for *Bank* should return a string containing a list of the accounts, with all account information, one per line, sorted by name.
- *Account* is an abstract class. Each account stores the name of the account holder, an account number (sequentially assigned automatically upon opening), and the current balance, along with an appropriate constructor to initialize these fields, and methods to add and subtract from the current balance. NOTE: All these methods are implemented in the Account class, so even though *Account* is abstract, none of the methods you are implementing are abstract. *Account*s should be Comparable by name.
- The *InterestBearingAccount* interface declares a single method *addInterest* (no parameters, void return type) that increases the balance by the interest rate that is appropriate for the particular account.
- An *InterestCheckingAccount* is an *Account* that is also an *InterestBearingAccount*. Invoking *addInterest* increases the balance by the interest value. This value defaults to 3%, but could be higher or lower on an account-by-account basis. Lines from the file will contain the percentage as a float (e.g. 3.0f) as the *class specific information* entry.
- A *PlatinumCheckingAccount* is an *InterestCheckingAccount*. Invoking *addInterest* increases the balance by double the rate for an *InterestCheckingAccount* (whatever that rate happens to be).
- A *NonInterestCheckingAccount* is an *Account* but it is not an *InterestBearingAccount*. It has no additional functionality beyond the basic *Account* class and will have no *class specific information* from the file.
- *AccountTest*, a driver for your *Bank*. This should get a filename from the command line arguments, create appropriate *Account*s, and upon instantiation of *Bank*, send whatever container of *Account*s via the constructor. You should then make calls to *Bank*'s totalAssets, addInterest, totalAssets (again), openAccount, toString, closeAccount, toString. See testing below for additional details.

Expectations: Your code will need to be neat, concise, well documented and above all, correct (see Testing). All classes should have headers and each method should have comments describing the method's function. **You must do error checking on any user input.** For every

class, appropriate constructors, accessors, mutators, and toString methods must be implemented. Any novel or possibly confusing code should be explained, as I do get confused and distracted easily.

Testing: I will be testing this with variously sized files, including an empty file, no file passed via command line and a file that does not exist. You will need to perform basic error handling, but you can assume that the file is properly formatted. There will be a sample input file, and sample output posted.

Quick Note: No prints should occur in any of the classes other than *AccountTest* and the *openAccount* and *closeAccount* methods of *Bank*.

Grading rubric will be given out at least a week ahead of the due date. You may work in teams of up to three for this assignment. If you choose to work in teams, one member (and only one) of the team must email me that information by 5pm on September 27th. Failure to do so (and continuing to work together) will result in a 30 point deduction.

Learning Targets: inheritance, abstract classes, interfaces, Comparable, file I/O, String manipulation

DUE: October 16th, 11:59 pm Eastern via Blackboard

Credit: Adapted from Exercise 4.53 from "Data Structures and Problem Solving Using Java, 4e."