Week 3 Lab Exercise

You may begin working on the lab exercise as soon as it is released to the class – you can start working at home and you don't have to wait until your scheduled lab time. We recommend that you read all of the instructions until the end before writing any code.

You can come in during scheduled lab times for CSCI 235 to ask questions and get help from the lab instructors and teaching assistants. You may discuss the tasks with your classmates and friends, but you are NOT allowed to share code, or even show your code to one another.

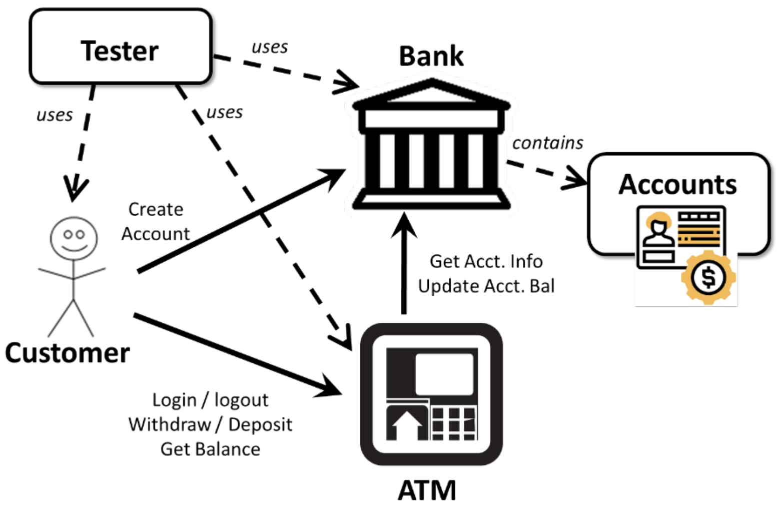
When you are done and ready to submit your lab to Moodle (on the lecture page, not the lab page), find your **src** folder within the project directory, zip it up, and submit the resulting archive file. Any other submissions, such as code copied and passed into text files will not be accepted.

Submission deadline is Friday, September 2, at 11:00 pm.

The maximum grade for this lab is **2 points**. To get the maximum, all of the following tasks should be completed, and your code should compile and run properly. Late submissions will receive 0 points.

Instructions

For this lab, you will be creating a banking system with five classes, Bank, ATM, Customer, Account, and Tester which are conceptually related in the following way:



Part 1: First, implement all classes within a single package called banking_system, using the following details:

1. A Bank can contain several Accounts and several ATMs. The Bank should start with zero Accounts, and creates and stores them when new Customers request new accounts with the Bank. New ATMs are created separately from the Bank object, and are attached to the Bank via another method.

2. Accounts have an account number (stored as an integer), and they maintain a record of the account balance (another integer). Note that nobody except a Bank should have access to Accounts!
3. Each ATM is associated with one Bank -- it should have a field holding onto a Bank reference, which should be set when it is attached to a Bank.
4. The Bank should implement the following methods:
    1. createAccount(), which is intended to be called by Customers, which generates a new Account object with a unique account number, and zero balance, which is stored in the Bank. This method returns the account number to the caller.
    2. attachATM(ATM atm), which is intended to be called by main, which adds the ATM to the Bank's associated ATMs, and also gives the ATM a reference to the Bank itself so it can communicate with it later.
    3. accessAcctInfo(int acctNum), which is intended to be called by an ATM, which returns the current balance for the given Account. An exception should be thrown if the given account number has no corresponding Account in the Bank.
    4. updateAcctBal(int acctNum, int diff), which is also intended to be called by an ATM, which adds the given diff value (which could be negative) to the balance for the given Account. An exception should be thrown if the Account cannot be found, or if the resulting balance would be negative for the given diff value.
5. The ATM class should implement the following methods, which are all intended to be called by a Customer. Note that none of these methods should throw an exception!
    1. loginToAccount(int acctNum) which tells the ATM to check if the Bank has the given Account – if it does, keep track of the account number for this session and return true; otherwise return false
    2. deposit(int amount) which tells the ATM how much money to add from the current account, and return true if successful, and false otherwise. Also return false if nobody is logged in.
    3. withdraw(int amount) which tells the ATM how much money to deduct from the current account, and return true if successful, and false otherwise. Also return false if nobody is logged in.
    4. getBalance() which should return the balance in the current account. Return 0 if nobody is logged in.
    5. logout() which should end the current ATM session for the customer.

    In addition, the ATM class should implement the setter setBank(Bank b), which should only be called by the attachATM method in Bank.
6. For simplicity, a Customer is associated with one Bank and one ATM which should both be passed in and set in its constructor. Furthermore, the Customer class should have the following methods which simulates the activities of a real customer:
    1. openAccount(), which should open a new account with the Bank. Don't forget to keep track of your account number!
    2. depositMoney(int amount), which should login to the ATM, deposit the specified amount of money, and logout

    3. withdrawMoney(int amount), which should login to the ATM, withdraw the specified amount of money, and logout
    4. checkBalance(), which should login to the ATM, get the current balance in the account, logout, and return the balance

7. Create a Tester class with a main method which creates a Bank and two ATMs that are attached to that Bank. Main should then create three Customers for that Bank with the ATMs, and then have the Customers open accounts and perform several transactions.
8. Use System.out.println statements throughout your code to see what is going on during execution.

Part 2: You may have noticed something horribly insecure about this particular banking system – anybody can create their own Accounts, and you can spoof the behavior of ATMs by making ATM-specific calls directly to the Bank.

To remedy this, first create a new package called outside and move the Customer and Tester classes inside of it -- this models the conceptual separation between bank stuff and non-bank stuff.

Then, modify the necessary access control levels on the constructors, methods, and classes to prevent anyone outside of banking_system from creating or doing things they shouldn't be able to do. It is okay to be able to construct Banks and ATMs from the outside, though, since we need to be able to create these "globally" to set up such simluations.