Program #2 (25 points)

Phase I due Feb. 16 (Monday) @10pm
Phase II due Feb. 20 (Friday) @10pm
Phase III due: Feb. 23 (Monday) @10pm
Grace date for all: Feb. 25 (Wednesday) @10pm

Program Description

You will write Java codes to implement an ATM (Auto Teller Machine). Let's assume that cash dispenser and deposit slot are not part of your ATM, and your ATM processes command line transactions. Your ATM is capable of handling Deposit, Withdraw, Open account, Close account, check Balance and Quit commands with the formats below, where ID is the bank account id and PIN is the password, and AMOUNT is the integer amount to be deposited or withdrawn. Note that the commands are case-sensitive, which means lowercase commands are considered invalid, and you must handle bad commands.

```
D ID|PIN|AMOUNT
W ID|PIN|AMOUNT
O ID|PIN|AMOUNT
C ID PIN
B ID PIN
Q
```

- **`D`** command—deposit AMOUNT dollars to the bank account identified by ID and PIN. Must check if the bank account exists.
- **`W`** command—withdraw AMOUNT dollars from the bank account identified by ID and PIN. Must check if the bank account exist, and if the account has enough balance for the withdrawal.
- **`O`** command—open (add) a new bank account with ID, PIN and AMOUNT dollars. Must check if the bank database contains the account before adding the new bank account. If the container is full, call grow() method.
- **`C`** command—close (remove) a bank account identified by ID and PIN. Must check if the bank account exists. You don't need to maintain the original order of the accounts.
- **`B`** command—check the balance of an bank account identified by ID and PIN. Must check if the bank database contains the bank account.
- **`Q`** command—turn off the ATM, display the list of current bank accounts, and display "`ATM stop running.`"

Program Requirement

1. Program 2 is an **individual assignment**. Sharing your solution could result in being accused of **plagiarism**.
2. You MUST follow the software development ground rules.
3. You MUST use **SE tools** to keep an up-to-date log of the time spent working. Up to **-5 points** if this is not done or you do not log all your time or you do not provide specific comments as to what you were working on.
4. Each class must go in a separate file. I will provide **Prog2.java**.
5. There are 3 phases for Program 2. **Each phase has a due date**. You MUST follow the instructions of each phase, and implement the classes and methods specified. **-2 points** for each method not implemented and/or not used. Up to **-5 points** for not completing each phase by the phase due date. You will get **0 points** if you submit Prog2 after **Feb. 25, 10pm**, or submit Prog2 with differences. You MUST submit Prog2 with 0 differences to pass this course.
6. Before your final submission, (a) copy all the codes from **BankDatabaseTest.java** to Prog2.java, and then (b) **COMMENT OUT all the codes you just copied**! (select the codes just copied, and select Source/Toggle Comment.)
7. Submit the following files to the grader:
   - BankAccount.java
   - BankDatabase.java
   - ATM.java
   - Prog2.java
8. If you need help from me, place your project in **S:\Courses\CSSE\changl\cs2430\your_login_id**. So I can access it from my office when you stop by.

Program #2 – Phase I

Due Date: Feb. 16 (Monday) @10pm

Phase I Requirement

At this phase, you will create the **BankAccount** class with a testbed main as discussed in the class. Up to **-3 points** if Phase I is not done by the due date **Feb. 16, 10pm**.

1. Log your time on SE tools, under Program2 as working on the **Phase I** activity.
2. Create a Java project **Prog2** and create a new Java class **BankAccount**, and write Java codes for the following methods. You CANNOT read (input) or write (output) in any methods, or **-2 points**. You CANNOT add other data members, or **-1 points** for each data member added.

```
import java.util.StringTokenizer;
public class BankAccount
{
   private String account;
   private String pin;
   private int    balance;

   public BankAccount(String acctno, String pn) //constructor; initialize balance to 0;
   public BankAccount(String acct) //constructor; parse acct into 3 tokens
   public String getAccount() //return the account
   public int getBalance() //return the balance
   public boolean debit(int amount) //subtract amount from balance if balance >= amount
   public void credit(int amount) //add amount to balance
   public boolean equals(Object obj) //two accounts are equal if IDs and PINs are the same
   public String toString() //return a single string with account, balance.
} //end of BankAccount class
```

3. Write a testbed main at the bottom of BankAccount class. This is a small main that you put at the bottom of your class that is used to test your class. You MUST include simple tests for ALL constructors and methods. You MUST print out the results to ensure the correctness of the results. All test cases must be documented with a Word document. For example:

Test Case	Description	Input	Expected result / output
1	Add an integer to the list	**A 21**	21 was added to the list.
2			
...			

You must follow the instructions in the "Test Specification" section of the Software Development Ground Rules. Name your Word document with your login name followed by an underscore followed by **p2test.docx**. For example**, johnsona_p2test.docx.** When you are done, put the **test document** and **BankAccount.java** into a folder called your_login_name_phaseI (for example, **johnsona_phaseI**,) and copy the folder to the dropbox folder: **S:\Courses\CSSE\changl\cs2430\1Dropbox**.

4. Don't forget to PUNCH OUT of SE tools.

Program #2 – Phase II

Due Date: Feb. 20 (Friday) @10pm

Phase II Requirement

At this phase, you will create the **BankDatabase** class with a JUnit test discussed in the class. Up to **-2 points** if Phase II is not done by the due date **Feb. 20, 10pm**.

1. Log your time on SE tools, under Program2 as working on the **Phase II** activity.
2. Add a new Java class **BankDatabase** to your Prog2, and write Java codes for the following methods. No read (input) or write (output) in any methods except print(), or **-2 points**. You CANNOT add other data members, or **-1 point** for each data member added. This is the container class for BankAccount class. You MUST make it **growable**.

```
public class BankDatabase
{
   private final static int GROW_SIZE = 4;
   private BankAccount [] bankDB;
   private int numAcct;

   public BankDatabase() //constructor, create an instance with size 0
   private int find(BankAccount bk) //return the index if found, -1 otherwise
   private void grow() //grow the container by GROW_SIZE
   public boolean open(BankAccount bk) //add the given account if account doesn't exist
   public boolean close(BankAccount bk) //remove the given account if found
   public int balance(BankAccount bk) //return the balance, return 0 if account not found
   public boolean deposit(BankAccount bk)//deposit money to the given account if it exists
   public int withdraw(BankAccount bk) //return 0 if the withdrawal is successful;
   //return -1 if account not found, and return 1 if balance is not enough for withdrawal.
   public void print() //output all accounts in the container
   public boolean contains(BankAccount bk); //check if a given account exists
} //end of BankDatabase class
```

3. Create a JUnit test for **BankDatabase** class. Follow the steps.
   a) Close BankDatabase class. (Right click on the class and select Close.)
   b) Right click on BankDatabase class, and select Tool, and then select Create/Update Tests.
   c) In the Create Tests Window, make sure the location is "Test Packages" and click OK. If prompted, select JUnit 4.x. This should create a new class in Test Packages called BankDatabaseTest.
   d) Open BankDatabaseTest.java if it is not opened already. (It is in the default package of the Test Packages folder.)
   e) Ignore BankDatabaseTest(), setUpClass(), tearDownClass(), setUp(), and tearDown(). There is a method in BankDatabaseTest to test each public method in BankDatabase. JUnit has tried to set up one test for each method. In some cases, the automatically-generated test is OK. In other cases, it is NOT OK and need to be modified. In all cases, one test is not enough to thoroughly test a particular method. You are going to write more tests for each method.
   f) Each test method in BankDatabaseTest declares a variable instance and creates an object of BankDatabase and assigns it to instance. It is a standard name that JUnit uses. You can call any instance methods such as:

```
if ( instance.open(new BankAccount("myacct", "mypin");
...
if ( instance.close(new BankAccount("myacct", "mypin");
...
```

   g) Each test method in BankDatabaseTest also uses two other variables expResult and result to assist you to write JUnit tests. You don't have to use them and could remove them if you do not use them.

JUnit also has three methods that allow you to check if the values returned are what they are supposed to be: `assertTrue(), assertFalse()` and `assertEquals().` Some usage examples follow:

```
// It says that the test expects that the new account can be added to the list.
    assertTrue( instance.open(new BankAccount(“myacct”, “mypin”));

// It says that the test expects that removing the new account will fail,
// in other words, the new account is not in the list.
    assertFalse(  instance.close(new BankAccount(“myacct”, “wrongpin”));
```

If the tests for a method all pass, then when you run the test (right click the class file and select Run File), they show up as green, if a test fails, it shows up as red.

h) Run the tests: select Run/test Project (Prog2), in the output window, you will see the JUnit test results. In particular, you should see that all tests have FAILED and are shown in red. Under testOpen(), remove the fail() command and change the code as follows.

Change

- `BankAccount bk = null;`

to

- `BankAccount bk = new BankAccount("myacct", "mypin");`

Change

- `boolean expResult = false;`

to

- `boolean expResult = true;`

Run the JUnit tests again. Run / Test Project (Prog2) and this time testOpen() should be marked as PASSED. Of course, the remaining tests should FAIL. Put several calls to testOpen() enough to make the list “grow”.

i) For every other method, remove the fail() line, modify the test created by JUnit, if necessary, and put in some additional good tests. The tests should attempt to THOROUGHLY test the method. The test should set up the list, then assert something. For testPrint(), just put elements in the list and call the print method WITHOUT asserting anything.

4. When you are done, put **BankDatabaseTest.java** and **BankDatabase.java** into a folder called your_login_name_phaseII (for example, **johnsona_phaseII** and copy the folder to the dropbox folder: **S:\Courses\CSSE\changl\cs2430\1Dropbox**.

5. Don’t forget to PUNCH OUT of SE tools.

Program #2 – Phase III

Due Date: Feb. 23 (Monday) @10pm

This is the FINAL phase of program 2. You must get all Java files done and run them as a project. All Java files MUST be submitted to the grader by the **grace date Feb. 25 @10pm**. Download **ATM.java** and **Prog2.java** from CSSE webpage, and complete the methods in ATM.java.

1. Log your time on SE tools, under Program2 as working on the **Phase III** activity.
2. Copy ATM.java and Prog2.java to Prog2\src folder. Complete the methods in ATM.java.
3. Run and test your program with the sample input below, and compare your output with the sample output below. A sample input file **P2.in** is provided in the course files folder on CSSE webpage. If your program is getting a correct output, submit all java files to the grader: (1) ATM.java (2) BankDatabase.java (3) BankAccount.java, and (4) Prog2.java.
4. When you are getting 0 differences, comment all classes and methods and fix your programming style accordingly.
5. DO NOT forget to copy the codes from BankDatabaseTest.java to Prog2.java, and **COMMENT OUT** all the codes before your **final submission**! The **grace date** for the final submission of all Java files is **Wednesday, Feb. 25 @10pm.** Submission after the grace date is worth 0 points. You MUST submit the program with 0 differences to pass this course.
6. Don't forget to PUNCH OUT of SE tools.

Sample Input

```
D WILSON|cs243|1500
C WILSON cs243
O WILSON|cs243|1219
D WILSON|cs234|1500
D WILSON|cs243|1500
W WILSON|cs243|5000
W WILSON|cs243|50
a
B WILSON cs243
O ZUKERBERG|FACEBOOK|2430
W WILSON|cs243|2000
B WILSON cs243
W ZUKERBERG|cs243|200
W WILSON|FACEBOOK|200
W ZUKERBERG|FACEBOOK|200
B ZUKERBERG FACEBOOK
B ZUKERBERG GOOGLE
O ZUKERBERG|FACEBOOK|1000
O NEWGUY|FACEBOOK|200
O NEWGUY|FACEBOOK|200
C NEWGUY FACEBOOK
B NEWGUY FACEBOOK
c
C WILSON cs234
C WILSON cs243
O PAGE|GOOGLE|12345
O CHAMBER|CISCO|1200
O CHANG|PLATTEVILLE|2342
O ULLRICH|CSSE|1000
O ULSVIK|ADMIN|30000
W ULLRICH|CSSE|500
W PAGE|GOOGLE|345
Q
```

Sample Output

```
Account information incorrect!
Account closing failed.
Account opening successful.
Account information incorrect!
New balance: 2719
Balance low! Current balance: 2719
New balance: 2669
Command 'a' not supported!
WILSON, your current balance is 2669
Account opening successful.
New balance: 669
WILSON, your current balance is 669
Account Information incorrect!
Account Information incorrect!
New balance: 2230
ZUKERBERG, your current balance is 2230
Account information incorrect!
Duplicate account! Account opening failed.
Account opening successful.
Duplicate account! Account opening failed.
Account closing successful.
Account information incorrect!
Command 'c' not supported!
Account closing failed.
Account closing successful.
Account opening successful.
Account opening successful.
Account opening successful.
Account opening successful.
Account opening successful.
New balance: 500
New balance: 12000

Bank Database...
ZUKERBERG, 2230
PAGE, 12000
CHAMBER, 1200
CHANG, 2342
ULLRICH, 500
ULSVIK, 30000
ATM stop running.
```