BTP400 Lab Activity #2 (Individual Work) Version 2.0

Focus: a) Programming with Java arrays and ArrayList.
b) Creation of a composite class.

Notes:

1. **The lab is due by 11:58 pm, January 27 according to the course timeline. No extension will be given. You must submit your Java source files AND screenshots to Blackboard by 11:58 pm. You will receive a ZERO if you miss the due date. Only one submission is allowed. Internet issues and system problems will not be considered for any extension. Please submit your work as early as possible. Careless mistakes in submission will not be considered as well.**

2. **In order to receive full marks, you must give a DEMO and provide correct answers during CODE REVIEW. Otherwise you may receive 50% (maximum).**

Part A: Java Arrays.

1. Review the lecture about overriding the **equals( )** method in a Java class. You may also read about this topic in The Java Tutorials: https://docs.oracle.com/javase/tutorial/java/IandI/objectclass.html.

2. Reuse the **Account** class from Lab Activity #1. Override the equals( ) method in that class. Here are the rules of object equality: Two Account objects are equal if they have the same account names, same account numbers and same account balances.

   Notes

   1. **In Lab #1, the data type of the field balance should be double, not int. If not, please make the change now.**
   2. **When you implement the equals( ) method, you should convert the balances into BigDecimal objects. Then you will use the equals( ) method in the BigDecimal class. You should do a little bit of research reading here and find out about the BigDecimal class in the Java API. Here are the links:**
      https://www.javaworld.com/article/2075315/make-cents-with-bigdecimal.html
      https://stackoverflow.com/questions/3413448/double-vs-bigdecimal
      https://docs.oracle.com/javase/9/docs/api/java/math/BigDecimal.html.

3. Reuse the **AccountTester** class from Lab Activity #1.
   Add testing code to demonstrate that the equals() method returns
   a) true when two Account objects are equal,
   b) false when two Account objects are not equal, and
   c) false when it receives a null reference as the actual parameter.

4. Take a screenshot of the output of your testing program.

Part A

5. Documentation

You must use Javadoc comments to document the equals() method. Your documentation should state the rules of equality for Account objects.

6. Create a Java class named as **ArrayTester**. Implement the main method to do the following two tasks.

a) Create an array of seven **Account** objects (with duplicates).

Here are the testing data.

```
"Peter Liu", "A12345", 5000
"Peter Liu", "A67890", 6000
"Abraham Lincoln", "Z6789", 7777
"Peter Liu", "A12345", 5000
"<your full name>", "E3333", 9000
"Abraham Lincoln", "Z6789", 7777
"Abraham Lincoln", "Z6789", 7777
```

b) Display a summary of comparing the Account objects in the array.

```
COUNTING SUMMARY
+ total number of accounts: 7
1. Peter Liu, A12345, 5000 : 2
2. Peter Liu, A67890, 6000: 1
3. Abraham Lincoln, Z6789, 7777: 3
4. <your full name>, E3333, 9000: 1
```

Notes: a) You must use the length variable of an array object instead of an integer value.
b) You must follow the exact sequence shown in the summary.

7. Compile and run your Java program in a command prompt window, NOT IN ANY IDE.

8. Take a screenshot of the output of your program.

Part B: Create a Bank Class.

1. Reuse the **Account** class that you have developed at Part A.

2. Create a Java class called **Bank**. A Bank object uses an **ArrayList** to keep track of accounts. This class should be saved to a separate Java source file (Bank.java). Code the zero-argument constructor and a constructor that takes one argument, namely the name of a bank (string). The zero-argument constructor initializes a Bank object with "Seneca@York" and creates an empty ArrayList. The one-argument constructor initializes a Bank object with the actual parameter received and creates an empty ArrayList.

3. Code the **addAccount()** method that adds an account(i.e. Account object reference) to the bank.

   You should use the following method declaration:

   public boolean addAccount(Account newAccount).

   The method takes an Account object as an argument. It returns true if the account is added successfully. Otherwise it returns false. For example, it returns false if the account been opened with the bank already. (In other words, all accounts in the Bank object must have unique account numbers**. No duplicates are allowed.**) It also returns false if it receives a null reference.

   - Code the **toString()** method. Read the sample output below for details.

4. Code the **equals()** method.
   Two Bank objects are equal if they have the same bank names and identical ArrayLists of Account objects. (In other words, both ArrayLists have the same sequences of Account objects.)

5. Code the **searchByBalance( )** method that searches for all accounts that have the same balance (integer). The method receives the balance as an argument. It returns an array of the accounts that it has found. It returns an array whose length is zero if it cannot find any matching account.
   Note: The method could receive a negative account balance.

Part C: Create testing code.

1. Create a Java class called **BankTester**. You must use the following arrays to create five **Account** objects in the main method:

```
String accountNames[]   = { "John Doe", "Mary Ryan", "Peter Liu",
                            "John Doe", "Peter Liu" };
String accountNumbers[] = { "A1234", "B5678", "C9999", "A1234", "D8901" };
int accountBalances[]   = { 1000, 3000, 5000, 7000, 3000 };
```

Part C

2. You must code the following test cases.

a) Create a Bank object and display information about this bank. The output should be like this:

```
*** Welcome to the Bank of <your full name> ***
It has 0 accounts.
```

b) Use the three arrays of testing data (shown above) to open accounts with the bank. Display information about the bank. The output should be like this:

```
*** Welcome to the Bank of <your full name> ***
It has 4 accounts.
1. number: A1234, name: John Doe, balance: $1000
2. number: B5678, name: Mary Ryan, balance: $3000
3. number: C9999, name: Peter Liu, balance: $5000
4. number: D8901, name: Peter Liu, balance: $3000
```

c) Call the addAccount() method with a null reference. Display the boolean value returned by the method.

d) Call the searchByBalance( ) method with 3000. Display the result of the search as follows:

```
We have found 2 accounts whose balance is $3000.
 1. number: B5678, name: Mary Ryan
 2. number: D8901, name: Peter Liu
```

e) Call the searchByBalance( ) method with -1111. Display the result of the search as follows:

```
*** NO ACCOUNT FOUND ***
```

3. Test that the equals() method returns true and false correctly. You must include a test case in which the following accounts are added in a different order than the test case 2. b):

```
1. number: D8901, name: Peter Liu, balance: $3000
2. number: C9999, name: Peter Liu, balance: $5000
3. number: B5678, name: Mary Ryan, balance: $3000
4. number: A1234, name: John Doe, balance: $1000
```

4. Compile and run your Java program in a command prompt window, NOT IN ANY IDE.

5. Take a screenshot of the output of your program.

6. No javadoc comments are required to document the Bank class.

C. Submission Requirements.

1. Submit at Blackboard:
   a) four Java source files (i.e. Account, Bank, ArrayTester, BankTester) and
   b) two screenshots of the output of your testing programs
      (one for ArrayTester, another one for BankTester).