Laboratory Assignment № 8

Dr. Duncan, CSC 1350, Louisiana State University 11/12/2019

Implementing a Basic BankAccount Class

Learning Objectives

1. Implementing a Class that Describes an Object,
2. Implementing Accessors, Mutators and Constructors, and
3. Writing a Program to Test the Implementation of a Class

In the lab exercise this week, you will write a program that demonstrates the following concepts: implementing a user-defined class with accessors, mutators and constructors and writing a program that uses objects of a user-defined class.

The BankAccount Class

Define the *BankAccount* class as described in the Javadoc details section that is an addendum to this lab exercise. This class will be very minimal: it makes no assumption about the currency that the bank account uses. Be sure to document the class and its methods and fields using the relevant Javadoc tags. This class should be added to the project after *BankAccountTester*.

The *BankAccountTester* Program

Create a project *BankAccountTester* containing the BankAccountTester class. This class should consist of only one method, the *main*. The main method should perform the following tasks:

1. It prompts the user for the initial balance to be used in creating a savings account. Using that amount it creates *savings*, an instance of the *BankAccount* class, using the relevant method of the class.
2. It then creates a second account *checking*, also an instance of the *BankAccount* class, whose initial balance is $500.00 more than the *savings* account balance.
3. It then displays the balances of both the *savings* and *checking* accounts.
4. It then prompts the user for an amount to transfer from the *checking* to the *savings* account. Using the relevant methods of the class, it transfers the amount between the accounts.
5. Again, it displays the balances of both the *savings* and *checking* accounts.
6. It then determines whether the *savings* account balance is the same as the *checking* account balance, invoking the relevant method of the *BankAccount* class.

7. Using the copy constructor of the *BankAccount* class, it creates a *youngAdult* account equivalent to the *savings* account.

8. Finally, implicitly invoking the *toString* method of the *BankAccount* class, it displays the String representations of the *savings*, *checking* and *youngAdult* accounts.

Other Requirements

Remove all Netbeans auto-generated comments. Your program's interactivity must be as shown in the sample runs. Include the Javadoc documentation for each method in *BankAccount*, as given in the API documentation handout. Write header comments for both *BankAccount* and *BankAccountTester* using this Javadoc documentation template:

```
/**
 * Explain the purpose of this class; what it does <br>
 * CSC 1350 Lab # 8
 * @author YOUR NAME
 * @since DATE THE CLASS WAS WRITTEN
 * @see BankAccount (ADD THIS TAG ONLY IN THE BankAccountTester Javadoc HEADER COMMENTS)
 */
```

Here are sample program interactions:

Listing 1: Sample Run

```
Enter the initial balance for a savings account -> 1200.00
Your savings account balance is $1200.00.
Your checking account balance is $1700.00.

Enter an amount to transfer from checking to savings -> 475.25
Your savings account balance is $1675.25.
Your checking account balance is $1224.75.

Does the savings and check accounts have the same balance? false

Savings a/c: BankAccount[balance = 1675.25]
Checking a/c: BankAccount[balance = 1224.75]
Young Adult a/c: BankAccount[balance = 1675.25]
```

Listing 2: Sample Run

```
Enter the initial balance for a savings account -> -250.00
Exception in thread "main" java.lang.IllegalArgumentException: Initial ↩
   balance must be non-negative
at bankaccounttester.BankAccount.<init>(BankAccount.java:50)
at bankaccounttester.BankAccountTester.main(BankAccountTester.java:22)
Java Result: 1
```

Listing 3: Sample Run

```
Enter the initial balance for a savings account -> 500.00
Your savings account balance is $500.00.
Your checking account balance is $1000.00.

Enter an amount to transfer from checking to savings -> 1725.35
Exception in thread "main" java.lang.IllegalArgumentException: overdraft ↩
   withdrawal is not allowed
at bankaccounttester.BankAccount.withdraw(BankAccount.java:74)
at bankaccounttester.BankAccountTester.main(BankAccountTester.java:28)
Java Result: 1
```

Listing 4: Sample Run

```
Enter the initial balance for a savings account -> 900.54
Your savings account balance is $900.54.
Your checking account balance is $1400.54.

Enter an amount to transfer from checking to savings -> 250.00
Your savings account balance is $1150.54.
Your checking account balance is $1150.54.

Does the savings and check accounts have the same balance? true

Savings a/c: BankAccount[balance = 1150.54]
Checking a/c: BankAccount[balance = 1150.54]
Young Adult a/c: BankAccount[balance = 1150.54]
```


Submitting Your Work for Grading

Navigate your way to the ...\NetBeansProjects\BankAccountTester\src\bankaccounttester folder using Windows file explorer. You should find *BankAccount.java* and *BankAccountTester.java*, files containing your source code for the program. Click one of the files and then hold down the shift key and click the other so that both files are selected. Right-click the selected files and create a compressed (zipped) folder containing a copy of each file. Rename the zip file *PAWSID_lab08.zip*, where *PAWSID* is the prefix of your LSU/Tiger email address - the characters left of the @ sign. Double-click the zip file to verify that both *BankAccount.java* and *BankAccountTester.java* are included the zip file. If the zip file does not contain both files, delete the zip file and repeat the steps. Upload the zip file to the digital drop box on Moodle.