University of New Brunswick
CS1083 – Open Access Version

Module 5 Assignment

The purpose of this assignment is to give you practice:

- handling exceptions in Java using try, catch, and throw.

For all assignments:

- Follow the instructions in the document "Java Coding Guidelines.pdf" available within the "Start Here" module, under the "Assignments" topic.
- For each .java file you create, include a javadoc comment (one that begins with /**) at the beginning of the file that describes that Java class. This comment block should include a line that begins with @author followed by your name and student number on the same line. (Note: inclusion of this comment is part of the instructions given in "Java Coding Guidelines.pdf")
- Include comments throughout your programs to explain any non-obvious portions of your code.
- It is recommended that you create a separate folder on your computer for each assignment. You might even wish to create separate sub-folders within an assignment folder if the assignment has multiple parts. Keeping your work organized makes it easier to find things later when you want to review what you have done.
- Few things in life are more frustrating than losing your work while working on an assignment. Get in the habit of saving frequently when working on an assignment. Also, regularly make backup copies of any files you create as part of your work for this course.
- Each module has a link to the discussion forum where you can ask questions and seek help in completing your assignments.
- Submitting your assignment involves creating a pdf file and uploading that single file to the assignment drop box on D2L. In general, that file will tend to include all Java code you wrote for the assignment, plus any output from running your programs that the assignment instructions specify you should capture. Specific submission instructions are included at the end of each assignment.
- To create the pdf file for each assignment, begin by opening a new document using the word processing program of your choice. Save the document using the name “Your Name CS1083 Module x Assignment Submission.docx” Replace x with the correct module number. “docx” may be different depending on which word processing software you choose to use.

- At the beginning of your submission document enter your name, your student number, CS1083, the assignment name (this one is “Module 2 Assignment”), and the date. It doesn’t matter if the date is when you started working on the assignment or when you submit it – either will do.
- You can add content to your submission document as you work on the various questions in the assignment. Clearly label each part of the assignment (“Part A” etc.) in your document. When your document is complete, save / export as a pdf file. Always make sure your pdf file opens properly before uploading it.
- To include Java code in your submission document, copy all the text in your .java file and then paste that text into your submission document. Use a monospaced font (e.g.: Consolas, Courier) for your code to maintain proper indentation.
- To include output from running your program in your submission document, you have two choices. You can either (a) copy and paste the text from your command prompt window, or (b) capture a screen shot of the output and include that as a picture / image in your submission document. Either approach is fine.
- To copy text from your command prompt window, try selecting the desired text and then pressing either command-c (Mac) or control-c (Windows or Linux). If you have issues, you can always use Google to see how to do this on your specific type of computer.
- To capture a screen shot of a selected portion of your command prompt window, try command-shift-4 (Mac), WindowsKey-shift-s (Windows), or shift-PrtScrn (Linux).
- Once you have pasted your output text into your submission document, make sure it is in a monospaced font (e.g.: Consolas, Courier) so you retain the alignment of output.

Bank Accounts

Documentation for the **BankAccount** class is provided for download with this assignment. This documentation was produced using javadoc. Code this class so it is consistent with this documentation. Use a "throwaway" test class to test the various parts of your BankAccount class as you incrementally code it a bit at a time.

Then do the same for the **LineOfCredit** class, for which javadoc documentation is also available.

For this application, each monetary amount is stored as an integer: a number of pennies. The toString() methods for both of these classes display these integer amounts in currency format. To accomplish this, use the getCurrencyInstance() method in the NumberFormat class. Divide the amount by 100 (to change to dollars instead of pennies) before formatting. Be careful of the data type so you don't lose the pennies.

Also code the **InsufficientFundsException** and **OverpaymentException** classes so you can compile and use the BankAccount and LineOfCredit classes with the test classes described below.

Write an **AccountTest1** class with a main() method that performs the following testing:

Create data for four test transactions as follows:

```
char[] transactionType  = {'d',  'd',  'w',   'w'};
int[] transactionAmount = {5000, 5000, 15000, 10000};
```

These values represent a deposit of $50.00, another deposit of $50.00, followed by two withdrawals of $150.00 and $100.00, respectively.

Apply this sequence of four transactions to a BankAccount instance that has an initial balance of $0.00.

Then apply the same sequence of four transactions to a LineOfCredit instance with an initial balance of -$75.00 and a credit limit of -$200.00. (These types of negative amounts are explained in the javadoc documentation for the LineOfCredit class.)

You might notice that these scenarios create problematic situations. For instance, the first two deposits raise the BankAccount balance to $100.00. This is not enough money for the third transaction – an attempt to withdraw $150.00 – so this attempt results in an InsufficientFundsException.

The AccountTest1 class handles exceptions only by defining the main() method as follows:

```
public static void main(String[] args) throws
          InsufficientFundsException, OverpaymentException
```

In other words, include no try-catch blocks in this program. This strategy means that any exceptions will stop the execution of AccountTest1 and result in an error message from the Java environment.

As described above, the first exception happens when attempting the $150.00 withdrawal, so executing AccountTest1 should produce the following output:

```
******* Testing a bank account *******

BankAccount[accountNumber=1000,balance=$0.00]

 Type      Amount   Balance
========  ========  ========
Deposit     $50.00    $50.00
Deposit     $50.00   $100.00
Withdraw   $150.00Exception in thread "main"
InsufficientFundsException: Insufficient funds
     at BankAccount.withdraw(BankAccount.java:85)
     at AccountTest1.main(AccountTest1.java:32)
```

As mentioned above, AccountTest1 includes code to first test a BankAccount, then a LineOfCredit. In this case, however, execution never makes it as far as the LineOfCredit testing. The insufficient funds exception crashes the program in the midst of trying to withdraw $150.00.

The message `"Insufficient funds"` appears in the above output because that is the message included in the constructor of the InsufficientFundsException class.

From the output above, you can see that AccountTest1:

- First displays a `******* Testing a bank account *******` message;
- Displays the initial status of the BankAccount using toString();
- Displays some column headings; and then
- loops to perform the four transactions, displaying transaction type, transaction amount, and resultant account balance for each one.

The transaction type and transaction amount are printed BEFORE calling either the deposit() or withdraw() method, which is why the exception message follows the display of those two values.

The purpose of this AccountTest1 class is to show what happens when exceptions (even those of our own making) are not handled by our code.

Once you have AccountTest1 completed and producing the output shown above, modify this class to produce **AccountTest2**. For this new class, add an appropriate try-catch block to contain the deposit() method call for the BankAccount instance, and another try-catch block to contain the withdraw() method call for the BankAccount instance. In each case, handle the specific type of exception thrown by that method.

Do not alter the code for testing the LineOfCredit instance.

Executing AccountTest2 should produce the following output: *(shown on the next page)*

```
******* Testing a bank account *******

BankAccount[accountNumber=1000,balance=$0.00]

 Type       Amount   Balance
========  ========  ========
Deposit     $50.00    $50.00
Deposit     $50.00   $100.00
Withdraw   $150.00  (Insufficient funds)
Withdraw   $100.00     $0.00

******* Testing a line of credit *******

LineOfCredit[accountNumber=1001,balance=-$75.00,creditLimit=-$200.00]

 Type       Amount   Balance
========  ========  ========
Deposit     $50.00   -$25.00
Deposit     $50.00Exception in thread "main" OverpaymentException:
Overpayment
      at LineOfCredit.deposit(LineOfCredit.java:92)
      at AccountTest2.main(AccountTest2.java:65)
```

Five things are worth noting about that sample output:

1. Due to the try-catch blocks for the BankAccount testing, this portion of the program is now able to complete its execution;
2. When an exception occurs, the affected transaction does not happen, so the balance remains the same;
3. Use the getMessage() method provided by the Exception class to display messages like `Insufficient funds`.
4. The resultant balance is not printed when an exception occurs. You can accomplish this with a boolean variable that keeps track of whether a given transaction is valid. Set this variable to true at the beginning of each transaction. Change this variable to false inside each catch block. Only print the resultant balance if this variable is true (in other words, if no exception occurred).
5. No try-catch blocks have yet been added to the LineOfCredit test code, so the program crashes when the second deposit transaction attempts to increase the balance of the line of credit to be greater than $0.00.

The final step in completing this assignment is to modify AccountTest2 to produce the **AccountTest3** class. For this new class, add an appropriate try-catch block to contain the deposit() method call for the LineOfCredit instance, and another try-catch block to contain the withdraw() method call for the LineOfCredit instance. In each case, handle the specific type of exception thrown by that method.

Executing AccountTest3 should produce the following output:

```
******* Testing a bank account *******

BankAccount[accountNumber=1000,balance=$0.00]

 Type       Amount   Balance
========  ========  ========
Deposit     $50.00    $50.00
Deposit     $50.00   $100.00
Withdraw   $150.00  (Insufficient funds)
Withdraw   $100.00     $0.00

******* Testing a line of credit *******

LineOfCredit[accountNumber=1001,balance=-$75.00,creditLimit=-$200.00]

 Type       Amount   Balance
========  ========  ========
Deposit     $50.00   -$25.00
Deposit     $50.00  (Overpayment)
Withdraw   $150.00  -$175.00
Withdraw   $100.00  (Insufficient funds)
```

The final withdrawal attempt for the line of credit fails because this transaction would take the balance below the credit limit of -$200.00. In other words, the owner of this line of credit is not permitted to borrow more than $200.00.

Now the program is able to finish executing. The exceptions are handled in a more graceful way than simply allowing the program to crash.

(Submission instructions are provided on the next page…)

Submission Instructions

Include the following in your submission document:

- Your name, student number, CS1083, Module 5 Assignment, and the date
- Complete code for the following classes:
    - BankAccount
    - LineOfCredit
    - OverpaymentException
    - InsufficientFundsException
    - AccountTest1
    - AccountTest2
    - AccountTest3
- Output from executing each of AccountTest1, AccountTest2, and AccountTest3. Label your output so it is clear which class produced which output.

D2L DROPBOX SUBMISSION INSTRUCTIONS

Remember to save / export your submission document as a pdf file, and then upload that one pdf file to D2L as follows:

1. In the top-navigation bar on the course screen, select 'Assessments' and then 'Assignments'.
2. Select the assignment title and follow the instructions to upload your submission document.