University of New Brunswick
CS1083 – Open Access Version

Module 6 Assignment

The purpose of this assignment is to give you practice:

- reading and writing with text files;
- reading and writing with object files; and
- handling instances of IOException.

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 Account Data Files

This assignment uses the following classes, which you created for the Module 5 Assignment:

- BankAccount
- InsufficientFundsException
- OverpaymentException

Create an updated version of the BankAccount class that implements the Serializable interface.

Write an **UpdateAccounts** class that includes a main method and does the following.

(For your own sake, I trust that incremental development is now second nature for you.)

The file **`opening_balances.txt`** is provided for download with this assignment. Here is an excerpt of the data contained in that file:

```
1324    117060
1672    189590
1903    50505
2157    163556
2519    177340
2744    237920
```

Each line contains the account number and current balance for one BankAccount. Monetary amounts are stated as pennies, represented as integers.

Your UpdateAccounts program reads this entire file. Create a BankAccount object for each line of data. Write your code so it is independent of the amount of data in the file – that is, use a “while … hasNext()” loop.

Store these account objects in an array, in the order you read them from the file. (You’ll notice they are already in order by account number.) Use an array that holds up to 100 accounts, but write the code to ensure you never attempt to put more than 100 objects in the array.

Include the phrase "throws IOException" on the first line of the main() method, which enables the IO operations your program does to work with the text files.

Updating Accounts

The file **`transactions.txt`** is also provided with this assignment. Here is an excerpt of the data contained in that file:

```
2744    D    45329
4848    W    25942
2157    W    28561
2519    W    49739
```

Each line includes an account number, transaction type (“D” for Deposit or “W” for Withdrawal), and a transaction amount in pennies. For example, the first line of data shown above indicates a transaction that deposits $453.29 into account # 2744.

Extend your UpdateAccounts class to read this entire file, performing each transaction on the appropriate account object in your array.

After you read an account number from the file, you will have to search for the corresponding account object in the array. Include a private findAccount() method in your UpdateAccounts class that uses the binary search algorithm to perform this search.

For each transaction, call the deposit or withdraw method (provided by the BankAccount class) to perform the transaction indicated by the data. Enclose these calls in a single “try” block, then “catch” any InsufficientFundsException instances thrown as a result of these calls.

Also include an empty catch block for OverpaymentException. This is necessary only because the BankAccount deposit() method throws this exception to be consistent with the deposit() method in the LineOfCredit class. We're not using LineOfCredit in this assignment, but the deposit() method in BankAccount mentions OverpaymentException so the compiler will insist that you catch it.

Each time you catch an exception, display information on the screen exactly as shown below. To help you debug your program, the following are the first three exceptions that should be thrown. For each exception that occurs, display:

- The type of exception, enclosed in “****” just like I have done;
- Use toString() to display the status of the account; and
- The type and amount of the transaction that failed.

```
**** Insufficient funds exception ****
BankAccount[accountNumber=3936,balance=$312.72]
Transaction type: W  Transaction amount: $395.72

**** Insufficient funds exception ****
BankAccount[accountNumber=5931,balance=$279.35]
Transaction type: W  Transaction amount: $391.20

**** Insufficient funds exception ****
BankAccount[accountNumber=4149,balance=$88.67]
Transaction type: W  Transaction amount: $405.25
```

Creating Data Files

Extend your UpdateAccounts class to create two new files.

The first is called **`closing_balances.txt`**. This file is in the same format as `opening_balances.txt`, and will end up containing similar data, except the account balances have now been updated by the transactions. To help you debug your program, here are the first few records you should see when you manually inspect the resulting new data file (for example, by opening it with your text editor):

```
1324    132392
1672    175384
1903    56188
```

On each line in the new file, separate each data value with a tab character (`"\t"`).

As you loop through the array of BankAccount objects to write lines of text into closing_balances.txt, at the same time write each account as an object into a file called **`accounts.obj`** using the ObjectOutputStream class.

Embed the code for opening and writing those two files within the same try-catch pattern used in EmployeeObjectIO.java, which is provided earlier in this module.

After creating those two files, display the following message:

```
Files created: closing_balances.txt
               accounts.obj
```

Finally, use the ObjectInputStream class to read all BankAccount objects from accounts.obj and use tostring() to display the status of each BankAccount you read from the file. Here are the first few lines of output you should see:

```
Accounts retrieved from accounts.obj
====================================
BankAccount[accountNumber=1324,balance=$1,323.92]
BankAccount[accountNumber=1672,balance=$1,753.84]
BankAccount[accountNumber=1903,balance=$561.88]
```

Notice that the values in this output match the first few lines of closing_balances.txt as shown above. Visually inspect the entire output to make sure all the values are the same.

Embed the code for opening and reading from accounts.obj within the same try-catch pattern used in EmployeeObjectIO.java.

(Submission instructions are provided on the next page.)

Submission Instructions

Include the following in your submission document:

- Your name, student number, CS1083, Module 6 Assignment, and the date
- Complete code for the following classes:
  - The updated BankAccount class (which now implements the Serializable interface)
  - UpdateAccounts
- Output from running UpdateAccounts
- The contents of closing_balances.txt

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.