Data Structures

CSC 3280

Fall 2020

Program 2: *FSC Telecom*
Assigned: Tuesday, September 22, 2020
Due: Friday, October 2, 2020 by 11:55 PM

Purpose:

1. Make use of recursion to solve problems.

Read Carefully:

- This program is worth 7% of your final grade.

- **WARNING**: This is an individual assignment; you must solve it by yourself. Please review the definition of cheating in the syllabus, the FSC Honor Code, and the serious consequences of those found cheating.
    - **The FSC Honor Code pledge MUST be written as a comment at the top of your program**.

- When is the assignment due? The date is written very clearly above.
    - **Note:** once the clock becomes 11:55 PM, the submission will be closed! Therefore, in reality, you must submit by 11:54 and 59 seconds.

- LATE SUBMISSION: you are allowed to make a late submission according to the rules defined in the syllabus. Please see course syllabus for more information.

- **Canvas Submission**:
    - This assignment must be submitted online via Canvas.
    - You should submit a single ZIP file, which has **ONLY** your Java files inside it.
        - Do NOT submit the full NetBeans Project folder…ONLY the Java files.
    - Within NetBeans, your project should be named as follows:
        - `FSCtelecom`
        - And when you make the project, NetBeans will automatically create a package with that same name. This is what we want.
- Your zip should be named YourFSCIDnumber.zip

Program 2: FSC Telecom

Objective

Learn to implement the functionality of linked lists.

Program Description

Guess what? FSC College has started its own telecommunications company called FSC Telecom. The benefit is great prepaid calling rates for all FSC students! The calling rate is 5 cents for one minute, and this includes both in-network (FSC Telecom) or out-network. Also, SMS rates are only 2 cents each! The price is great, but this system is limited in functionality. Postpaid accounts are not allowed; only prepaid accounts are allowed. Additionally, the system has limited memory: it can only save into memory the last 10 outbound calls and the last 10 SMS messages that the user sends.

For this program, you will implement a basic simulation of this FSC Telecom system. Students will make accounts, make calls, send messages, recharge their phones, and more. You will create one linked-list, which will be used to manage all of the accounts (each node of the linked list will be an object of type `Student`). You will use File I/O to read commands from a file and then print the output to a file. Example commands are ADDACCOUNT, MAKECALL, SENDTEXT, RECHARGE etc. Then, depending on the command, you will either add a new account, process a phone call or text, search for a student account, and more. But instead of printing to the console window (screen), you will print to an output file.
**Sample input and output files have been provided for you on the website.*

For this program, you will create three classes:

- **`Student.java`** : This class will be used to create objects of type `Student`. Each Student object will store all the needed information for <u>one</u> student in a specific account (ID, first name, last name, mobile number, etc). These `Student` objects will be added (and deleted) from linked lists based on the commands from the input file.
- **`FSCTelecomAccounts.java`** : This is the Linked List class! You will use this class to make one object of type `FSCTelecomAccounts`. Again, this is the actual linked-list class. This is the class that will have the "head" and will include all the methods that are used to operate on a linked-list of `Student` objects.
- **`FSCTelecom.java`** : This is the class that will contain main.

****And before going any further, please note that I've included a skeleton Java file with this 7z zip file. You should use that as your main program and simply add content as needed.*

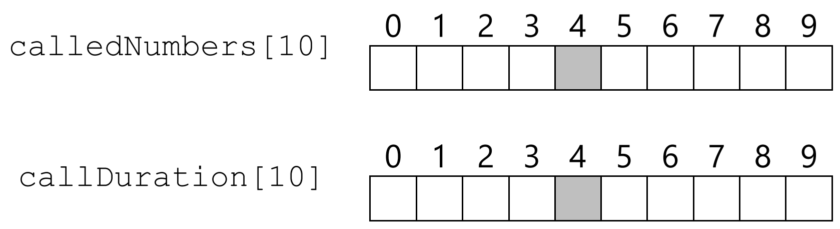
The Commands to be implemented are as follows

1. **ADDACCOUNT**
   This command will be followed by the following information in the input file: a student ID (a non-negative integer), first name, last name, and a String representing the new phone number (ex: 0547000331). When you process this command, you should make a new `Student` object, and then you should scan, from the file, the student ID, the first and last name of the student, and the new phone number. *Note: all new accounts receive a free initial balance of $20 for subscribing to the service. Finally, you must insert this object into the linked-list based on the sorted order of the student ID number.
   Example:
   If the following line was in the input file:
   ```
   ADDACCOUNT 1323188 Joe Smith 0547000331
   ```

   This would be the output:
   ```
   Command: ADDACCOUNT
         Name:          Joe Smith
         Student ID:    1323188
         Phone Number:  0547000331
         Balance:       $20
   ```

2. **MAKECALL**
   This command will be followed by a student ID, a non-negative integer, a String representing the number that was called, and the number of call minutes, also a non-negative integer value. If the record is found in the `FSCTelecomAccounts` linked-list and if that record has enough balance for the desired call minutes, it should be executed and a message printed to output file (see sample output file). The call rate is fixed at the rate of 5 cents per minute. If the ID of the calling student is not found in the linked list OR if the student has zero balance, an error message should print. Additionally, if the student has some balance, but not enough to complete the call, the call should be processed, but the length of the call will be limited based on the balance of the customer. A message should print stating that the call was terminated due to insufficient balance. (see sample output file).

   Each Student object will maintain two arrays to hold call history for the last 10 called numbers. The first array is called `calledNumbers`, and it will maintain the history of the last 10 called numbers. The second array is called `callDuration`, and it will save the time duration (length of call) for the last 10 calls. Below is an example of the two arrays. The value at index "i" of the `calledNumbers` array is related to the value at index "i" of the `callDuration` array. As in the example below, index 4 is shaded in both arrays to show you that they belong together.



***<u>Note</u>**: this array <u>**must**</u> stay in <u>sorted order based on the latest call</u>. It means the latest call (the last call sent/made by the user) will be saved at index 0. Therefore, you will be required to <u>**SHIFT**</u> the existing data of all cells to the right, thus creating a new space in the array at index 0 for the latest call. In this process, the saved call at the highest index (i.e. index 9) will be deleted since the FSC Telecom system has limited memory; we are keeping call records for only the last 10 calls.

(see input/output examples for more details)

3. **SENDTEXT**
This command will be followed by a student ID, and the destination number, a string value. If the student ID is found in the `FSCTelecomAccounts` linked-list, and that record has enough balance for the text message, the command should be executed and a message printed to output file (see sample output file). The text message rate is fixed at the rate of 2 cents/message. If the ID is not found in any linked list or if the student does not have enough balance, an error message should be printed instead (see sample output file).

   Each Student object will maintain an additional array to hold the history of the last ten texted numbers. This array is called `textedNumbers`, and it will maintain the last 10 numbers to whom any text messages were sent. The most recent number texted will be saved into index 0, and the oldest number texted will be saved into index 9. Again, this will require the use of forward shifting the String values each time a new text has successfully been made.

4. **USEDATA**
This command will be followed by a student ID, a non-negative integer, and number of data in **KB** (for example 20 means 20 KB of data), a non-negative integer value. If the student ID is found in the `FSCTelecomAccounts` linked-list, and that record has enough balance for the desired amount of data, it should be executed and a message printed to output file (see sample output file). USEDATA should only be processed when the customer has <u>enough balance for the **full** amount of the data</u>. If the balance is not enough to cover ALL the data, then no data will be used; you should print an error message (see output). The data cost is fixed at the rate

of $1 for 100 MB. This means you must calculate the actual cost of the transmission to see if the student has enough balance. As an example, if the student wants to use 2744 KB of data, the price would be:

$$\text{Price for 2744 KB} = \frac{2744\,KB}{1} \times \frac{1MB}{1024\,KB} \times \frac{\$1}{100\,MB} = \$0.02679$$

The cost must be TWO decimals. So when calculating, you must use the CEIL function to calculate the value. For example, if the value came to 0.02679, then the value that you must subtract from the customer balance is 0.02. Here is the equation used to compute the dataCost:

```
// this first variable is a global at the top of your class
public static final double dataRatePerKB = (1.0 / 1024.0) / 100.0;

// this next variable is inside your useData() method
double dataCost = Math.ceil(kb * dataRatePerKB * 100000.0) / 100000.0;
dataCost = ((int)(dataCost * 100))/100.0;
```

5. **RECHARGE**
   This command will be followed by a student ID, a non-negative integer and amount to be recharged, a double value. If the record is found in the `FSCTelecomAccounts` linked-list, it will be recharged by adding the amount to the existing balance and a message printed to output file (see sample output file). If the ID is not found in any linked list, an error message should be printed instead (see sample output file).

6. **DELETEACCOUNT**
   This command will be followed by a student ID, a non-negative integer. If the record is found in any linked-list, it should be deleted and a message printed to output file (see sample output file). If the ID is not found in any linked list, an error message should be printed instead (see sample output file).

7. **SEARCH**
   This command will be followed by a student ID, a non-negative integer. If the record is found, it is printed to the file (see sample output file). If it is not found, an error message should be printed instead (see sample output file).

8. **DISPLAYDETAILS**
   This command will be followed by an integer value, the student ID, and it will display last 10 call details (called number with duration) and last 10 numbers for sent messages. If the student ID is not found in the linked list, an error message should be printed instead (see sample output file).

9. **QUIT**
   This command will not have any other information after it (in the input file). When you scan this command, the program should print a message to the file and then quit.

Input File Guarantee/Promise

You are guaranteed that no other commands will be in the input file. You are also guaranteed that the input will be correct, which means you do not have to worry about mistakes in the input. Finally, you are guaranteed that each `ADDACCOUNT` command will be independent of all other commands (there will not be duplicate `ADDACCOUNT` commands in the input).

Input File & Output File

You must read from an input file called **`FSCTelecom.in`**. The input will consist of many lines, each starting with a command. Each command will have the required information, on the same line, as described above in the write-up. You should stop reading and processing input when you read the `QUIT` command.

Sample Input & Output File

You must print your output to an output file called **`FSCTelecom.out`**. We have provided you a sample input with matching output.

*****WARNING*****

Your program MUST adhere to the EXACT format shown in the sample output file (spacing capitalization, use of dollar signs, periods, punctuation, etc). The graders will use very large input files, resulting in very large output files. As such, the graders will use text comparison programs to compare your output to the correct output. If, for example, you have two spaces between in the output when there should be only one space, this will show up as an error even through you may have the program correct. You WILL get points off if this is the case, which is why this is being explained in detail. Minimum deduction will be 10% of the grade, as the graders will be forced to go to text editing of your program in order to give you an accurate grade. Again, your output MUST ADHERE EXACTLY to the sample output.

Grading Details

Your program will be graded upon the following criteria:

1) Adhering to the implementation specifications listed on this write-up.

2) Your algorithmic design.
3) Correctness.
4) **<u>Use of Linked-Lists. If your program does not use linked-lists, you will get a zero.</u>**
5) The frequency and utility of the comments in the code, as well as the use of white space for easy readability.
    a. This is your third programming course. By this point, these warnings should be unnecessary. Please do know that we're not kidding here. If your code is poorly commented and spaced and works perfectly, you could earn as low as 80-85% on it. Meaning, you can lose 15 to 20% due to lack of commenting/spacing.
6) Compatibility to the **<u>newest version</u>** of NetBeans. (If your program does not compile in NetBeans, you will get a large deduction from your grade.)
7) Your program should include a header comment with the following information: your name, **<u>email</u>**, student ID number, etc.
8) The FSC Honor Code must be written at the top of each java file.
9) All three java files must belong to the same package within NetBeans. This package should be called FSCtelecom.
10) Your output MUST adhere to the EXACT output format shown in the sample output file.

<u>Deliverables</u>

You should submit a zip file with <u>THREE</u> files inside:

1. `Student.java`
2. `FSCTelecomAccounts.java`
3. `FSCTelecom.java` (this is your main program)

***These three files should all be INSIDE the same package called **`fsctelecom`**. If they are not in this specific package, you will lose points.

<u>Suggestions</u>:

- Read AND fully understand this document BEFORE starting the program!
- Make your project in Netbeans with the name `FSCtelecom`. Make sure the box for create main class is checked and you do not change the defaults. This will result in your getting a package called `fsctelecom`.
- Next, start by making the Student class to create `Student` objects:
    - So make this class, with all the data members, constructors, and accessor/mutator methods.
- Now, make your Linked Lists classes, `FSCTelecomAccount`, and make all appropriate methods (using the actual Linked List code from Blackboard as a guide).
- Now, after you have made these two classes, begin to implement your main program file:
    - `FSCTelecom.java`

- This is your main program that will read the commands from the file, process the commands, and print to the output file.
- Finally, one by one, implement the commands

Hope this helps.

HINT: use a text compare program to compare YOUR output to the sample output. There will be thousands of lines in the output. You can Google "text comparison", and many websites will show in the results. You can past your output and the sample output provided at these websites. They will compare the two files and highlight any differences. Better than the above is to download a text comparison software. One good option is WinMerge. With WinMerge, you can specify two files, and the software will open both files up, side by side, and will highlight any differences.

UML Diagrams:

For this program, you will create **three** Classes (UML diagram shown below):

Student
Data Members private int ID private String firstName private String lastName private String phoneNumber private double balance private String[] calledNumbers private int[] callDuration private String[] textedNumbers private ***static*** int numStudents private Student next
Operations/Methods Student() // one or more Constructors ALL getter and setter methods. And any other methods you need.

- This is the class for the individual nodes

FSCTelecomAccounts
Data Members private Student head
Operations/Methods FSCTelecomAccounts() // one or more Constructors ALL getter and setter methods. ALL necessary methods for linked-list operations And any other methods you need.

- This is the linked-list class.

FSCTelecom
This is your MAIN. You read the file, process the commands, and print to the file from this class.

Final suggestion: START EARLY!