CSE3040: Java Programming (Fall 2020)
Final Project (Replacement for the Final Exam)
Due: December 23, 11:59PM (KST)

1. Introduction

In this project, you are going to develop a library manager program. You need to write a server and a client program, in which the client and the server communicate through a TCP socket. Carefully go through the requirements described below. Each of the requirements (e.g. R1, R2, etc.) will be evaluated.

2. Requirements

R1. The server program (Server.java) makes use of a file named "books.txt". It is a text file that records all the books in the library. Each line in the books.txt file is a book entry that consists of title, author, and ID of the user who is currently borrowing the book (borrower). The fields are separated by tab ('\t'). We assume that the tab character does not appear in the title, author, or borrower. The books are sorted in case-insensitive alphabetical order of the titles. An example books.txt is as follows.

books.txt

```
A Promised Land	Barack Obama	trump
Green Lights	Matthew McConaughey	-
Harry Potter and the Sorcerer's Stone	J.K. Rowling	trump
How to Catch a Unicorn	Adam Wallace	joebiden
I Love You to the Moon and Back	Amelia Hepworth	-
```

• In the first line, "A Promised Land" is the title of the book. Then, followed by a tab character, "Barack Obama" is the author of this book. Then, also followed by a tab character, "trump" is the ID of the user who borrowed this book from the library.

• If the book is not borrowed by anyone and is available at the library, then the third field (borrower) is shown as '-'.

• A book must always have a title and an author.

• Note that the books are sorted in case-insensitive alphabetical order of their titles. For example, if four books are named “aaa”, “Abc”, “bbb”, “CCC”, then they should be sorted in the following order.

```
aaa
Abc
bbb
CCC
```

R2. Whenever the content of books.txt changes, the server must update the books.txt file. For example, a new book may be added to the file, a book in the file may be deleted, and the borrower of a book may change. If one of these events occur, the books.txt must be updated immediately, in order to save the transactions even if the server program terminates abruptly.

R3. The server program takes one user argument, which is the port number. An example command to run the server program is as follows.

```
C:\Users\jso1\eclipse-workspace\cse3040fp> java -classpath .\bin cse3040fp.Server 7777
```

• If the number of user argument is not 1, the program should print the following message and terminate.

```
C:\Users\jso1\eclipse-workspace\cse3040fp> java -classpath .\bin cse3040fp.Server
Please give the port number as an argument.
C:\Users\jso1\eclipse-workspace\cse3040fp>
```

R4. The client program takes two arguments. The first argument is the server IP address, and the second argument is the server port number. An example command to run the client program is as follows.

```
C:\Users\jso1\eclipse-workspace\cse3040fp> java -classpath .\bin cse3040fp.Client 127.0.0.1 7777
```

• If the number of user argument is not 2, the program should print the following message and terminate.

```
C:\Users\jso1\eclipse-workspace\cse3040fp> java -classpath .\bin cse3040fp.Client
Please give the IP address and port number as arguments.
C:\Users\jso1\eclipse-workspace\cse3040fp>
```

R5. When the client program starts, it must establish a TCP connection with the server. The connection request may fail if the server is not listening at the other end. In this case, the program should print the following message and terminate.

```
C:\Users\jso1\eclipse-workspace\cse3040fp> java -classpath .\bin cse3040fp.Client 127.0.0.1 7890
Connection establishment failed.
C:\Users\jso1\eclipse-workspace\cse3040fp>
```

R6. Once the client successfully connects to the server, the program should show a prompt and wait for user input. The prompt should be like this:

```
C:\Users\jso1\eclipse-workspace\cse3040fp> java -classpath .\bin cse3040fp.Client 127.0.0.1 7777
Enter userID>>
```

• The user will enter his/her user ID. We assume that the user ID is a single word and consists of lowercase alphabets and numbers. If the user does not follow this ID format, then the program should print a message (shown in the example below) and show the prompt again. If the user enters the ID in the correct format, the user is logged in with that user ID. In this case, the program shows a greeting message, and the prompt should change to the user ID. (In the example shown below, the user enters nothing on the third try.)

```
Enter userID>> two words
UserID must be a single word with lowercase alphabets and numbers.
Enter userID>> Hel*o
UserID must be a single word with lowercase alphabets and numbers.
Enter userID>>
```

```
UserID must be a single word with lowercase alphabets and numbers.
Enter userID>> trump
Hello trump!
trump>>
```

※ From now on, each requirement is on implementing a command from the client prompt. In order to implement the command, you will need to implement functions at the client as well as the server. Especially, you will need to design the message sent and received between the server and the client. Note that all the following commands are available after the user successfully logins with a user ID.

※ When the user enters "add", the client asks the user for the title and the author using the prompts as in the example shown below. The blue characters are entered by the user.

```
trump>> add
add-title> Trump: The Art of the Deal
add-author> Donald J. Trump
A new book added to the list.
trump>> add
add-title> Trump: The Art of the Deal
The book already exists in the list.
trump>> add
add-title>
trump>> add
add-title> Becoming
add-author>
trump>>
```

R7. If the user enters both the title and the author, then the client should send a message to the server in order to add the book to the list. The server should check whether a book with the same name already exists in the list. If the book is not in the list, then the server adds the new book to the list. As default, the book is not borrowed by anyone. Then, the server sends a message to the client saying that the new book was successfully added to the list. The client prints "A new book added to the list.".

R8. If the book with the same title is already in the list, the server does not add the book to the list. The server sends a message to the client saying that a book with the same title already exists in the list. The client prints "The book already exists in the list.".

R9. When the prompt "add-title>" or "add-author>" is shown, if the user enters nothing, the command is cancelled, and the prompt goes back to the main prompt.

• For the books, the title and the author are case-insensitive. "A Promised Land" and "a promised land" can be regarded as the same book. Thus, adding a new book "a promised land" should fail if another book called "A Promised Land" already is in the book list.
※ When the user enters "borrow", the client asks the user for the title using the prompts as in the example shown below. The blue characters are entered by the user.

```
trump>> borrow
borrow-title> a promised land
You borrowed a book. - A Promised Land
```

```
trump>> borrow
borrow-title> a promised land
The book is not available.
trump>> borrow
borrow-title>
trump>>
```

R10. If the user enters the title, then the client should send a message to the server. The server should check whether a book with the same name already exists in the list. The title is case-insensitive. If the book exists and the book is not borrowed by anyone, then the server changes the borrower for the book and sends a message to the client saying the user borrowed the book. The client prints a message on the screen as shown in the example above.

R11. If the requested book is in the list but is not available (borrowed by someone), or is not in the list, then the server sends a message to the client saying that the book is not available. The client prints a message on the screen as shown in the example above.

R12. When the prompt "borrow-title>" is shown, if the user enters nothing, the command is cancelled, and the prompt goes back to the main prompt.

※ When the user enters "return", the client asks the user for the title using the prompts as in the example shown below. The blue characters are entered by the user.

```
trump>> return
return-title> a promised land
You returned a book. - A Promised Land
trump>> return
return-title> a promised land
You did not borrow the book.
trump>> return
borrow-title>
trump>>
```

R13. If the user enters the title, then the client should send a message to the server. The server should check whether the book exists in the list, and the book has been borrowed by the returning user. The title is case-insensitive. If the book has been borrowed by the user, the server changes the borrower to '-' and returns a message to the client saying that the book is returned successfully. The client prints the message as shown in the example above.

R14. If the returning book is in the list but was not borrowed by the user (either available or borrowed by someone else), or is not in the list, then the server sends a message to the client saying that the user did not borrow the book. The client prints the message as shown in the example above.

R15. When the prompt "return-title>" is shown, if the user enters nothing, the command is cancelled, and the prompt goes back to the main prompt.

R16. When the user enters "info", the list of books that the user is currently borrowing from the library is printed on the screen. This can be achieved by requesting the server to send the information. An example output would be as follows. The list of books should be sorted in case-insensitive alphabetical order of the titles.

```
trump>> info
You are currently borrowing 2 books:
1. A Promised Land, Barack Obama
2. Harry Potter and the Sorcerer's Stone, J.K. Rowling
trump>>
```

R17. When the user enters "search", the client asks the user for a string pattern. If the user inputs a string that is less than 3 characters, then the client prints "Search string must be longer than 2 characters" and show the prompt again. An example is shown below.

R18. If the user gives a string that is longer than 2 characters (that is, 3 characters or more), then the client sends a message to the server. The server will search for books where the title or the author includes the string pattern, case insensitive. The server returns all the books to the client, and the client prints the list on the screen. An example is shown below. If there are more than 1 book that matches the search string, then the books should be sorted in case-insensitive alphabetical order of the title.

R19. When the prompt "search-string>" is shown, if the user enters nothing, the command is cancelled, and the prompt goes back to the main prompt.

```
trump>> search
search-string> ab
Search string must be longer than 2 characters.
search-string> the
Your search matched 3 results.
1. Green Lights, Matthew McConaughey
2. Harry Potter and the Sorcerer's Stone, J.K. Rowling
3. I Love You to the Moon and Back, Amelia Hepworth
trump>> search
search-string>
trump>>
```

R20. When the user types an invalid command (anything other than "add", "borrow", "return", "info", or "search"), the program should show the command list. You can literally print the text on the screen as shown below.

```
trump>> abcde
[available commands]
add: add a new book to the list of books.
borrow: borrow a book from the library.
return: return a book to the library.
info: show list of books I am currently borrowing.
search: search for books.
trump>>
```

3. Other Requirements (Very Important!)

• Keep in mind that the server and the client program is meant to run on different machines. The file "books.txt" is only at the server. The client program MUST NOT directly access the books.txt file. If the client program accesses the books.txt file, then you will get no credit for this project.

• You must implement the server so that multiple clients can concurrently connect to the server and

send commands to the server. If your server can only support one client at the time, you will only get 50% of the credit you obtained from implementing the requirements.

4. Submission Instructions

※ Carefully read this part and follow the instructions. Not properly following the instructions here may result in point deduction.

(1) For this project, you are going to submit only the .java files. Specifically, you should submit **Server.java** and **Client.java**.

(2) Once you are ready to submit, you should make the .java files into a single zip file named cse3040_fp_20180001.zip. The numbers in the file name should be your student ID.

When you make the zip file, make sure that you are not using subfolders inside the zip file. When the zip file is extracted, the .java files should appear without having any subdirectory structures.

(3) Submit your zip file on the cyber campus.

5. Evaluation Criteria

- This project will count towards 30% of the final grade.

- The maximum point for this project is 100.

- You will get 5 points for successfully passing the test for each requirement items (R1 - R20).

- Additional point deductions may occur if you fail to follow the "other requirements" as well as the "submission instructions".

6. Academic Integrity

- You should write your own code. You can discuss ideas with other students but must not copy their work. You can also get help from the Internet, but you must not copy the source code from the Internet either. We have a duplicate check program which tests whether your source code is similar to other students' code as well as codes that are on the Internet.

- Duplicate work will receive zero credit.

7. Late Policy

- 10% of the score is deducted for each day, up to three days. Submissions are accepted up to three days after the deadline.