CSC 223 Project 3: Preliminary Version of Line Editor

The objective of this project is to implement a line editor application with selected data structures and test in JUnit framework to verify your implementation.

# Line Editor

In computing, a line editor is a basic type of computer-based text editor whereby one line of a file can be edited at a time. Unlike most modern editors, typing, editing, and document display do not occur simultaneously. Typically, typing does not enter text directly into the document. Instead, users modify the document text by entering commands at the command line. For this project, you will develop a preliminary version of a line editor where all manipulations are performed by entering commands through the command line. The manipulation commands include load file (either start a new file or append lines to the loaded file), display all lines, display single line, count number of lines, count number of words in the document, delete a line, insert a line, delete all lines in the loaded document, replace a word with another one, and save all lines to a file.

A sample command menu is shown below:

Menu: m

Load File: load fileName appendOption (1-append, 0-new list)

Print: p

Display number of lines: lines

Display line: line lineNumber

Count words: words

Delete line: del lineNumberToDelete

Append line: a

Insert line: i lineNumberToInseart

Clear document: cls

Replace words: rep originalWord newWord

Save to a file: s fileName

Quiz: quit

For example, a session where a user enters five lines of text, save them to a new file and manipulate lines with other commands may appear as follows:

C:\Users\jsmith2020>java LineEditor

Menu: m

Load File: load fileName appendOption(1-append, 0-new list)

Print: p

Display number of lines: lines

Display line: line lineNumber

Count words: words

Delete line: del lineNumberToDelete

Append line: a

Insert line: i lineNumberToInseart

Clear document:cls

Replace words: rep originalWord newWord

Save to a file: s fileName

Quiz: quit

-->a

Type a line:line 1

-->a

Type a line:line 2

-->a

Type a line:line 3

-->a

Type a line:line 4

-->a

Type a line:line 5

-->p

1. line 1

2. line 2

3. line 3

4. line 4

5. line 5

-->s data.txt

-->cls

-->p

-->load data.txt

java.lang.ArrayIndexOutOfBoundsException: 2

-->load data.txt 0

-->p

1. line 1

2. line 2

3. line 3

4. line 4

5. line 5

-->rep l L

-->p

1. Line 1

2. Line 2

3. Line 3

4. Line 4

5. Line 5

-->del 5

-->p

1. Line 1

2. Line 2

3. Line 3

4. Line 4

-->lines

There are 4 lines.

-->line 2

Line 2

-->quit

# Line Editor Commands

Like the one as shown above, if you run the program with no command line arguments it will begin with an empty document. If you give the name of an existing file as a command line argument it will open and load the contents of that file.

## Displaying the menu

The menu is displayed when the program starts. It can be displayed with the command **m**.

## Loading and saving files

The *load file* command **load fileName loadOption** will read the file contents into the editor and close the file. The *loadOption* specifies how the contents will be loaded to the editor; loading file with ***true*** option will append contents to existing editor; otherwise, it will empty the current editor and load its contents. The *save file* command **s fileName** will write the contents of the editor to a file.

## Displaying text

Text can be displayed with two commands: *display all(***p***)*, and *show a single line* (**line number**). The *display all(***p***)* command will display text with line numbers.

## Entering new lines of text

New lines can be added in two ways: append to the end of document or insert to the specified line. Command **a** will append the new line to the end of the document, it will prompt user to type a line. Command **i n** will insert a new line to the nth line.

Examples:

->a

Type a line:

Hello line editor

->p

1. line 1

2. Hello line editor

->i 2

Type a line:

Line 2

->p

1. line 1

2. line 2

3. Hello line editor

->

## Displaying number of lines and counting number of words

Your line editor allows users to check numbers of lines and words in the document. Command **lines** display the number of lines of the document; The command **words** displays the number of words in the document. All words in the document are separated with delimiters including: \s(white spaces) \t(tab) , . ; ’ ? \* ! ” @ - : .

## Deleting a line or all lines in the editor

To delete a single line, use command **del n**, it will delete nth line from the editor. You may delete all lines in the editor with the command **cls**.

## Replacing words

To replace a word with another one, use command **rep originalWord newWord**. This command will replace all occurrences of the original word with the new word.

# Part 1 – LineList and unit testing:

You have already learned several different data structures at this point. You may utilize any of them to work on this project. However, both stack and queue are structures that provide restricted access to their elements; a list may be a better option for this project. For this project, you are required to use a Node to store each line in a document and all functions of the line editor are encapsulated in the class named LineList.java.

Download the following files:

* Node.java
* LineList.txt
* LLTest.java

LineList.txt is the template for the LineList class. Rename it to LineList.java. Your task in this part is to implement all required methods as specified in the template of LineList class.

Once you completed your implement of LineList class, run LLTest.java in JUnit framework to test your LineList class. The LLTest.java test file includes sample test cases. Your goal is to pass all test cases in LLTest.java. We may modify the test case while grading the project.

After passing test cases, take a screenshot of the running output of unit testing. Then move to the part 2 of this project.

You may use the following file to do additional testing. (Optional)

* Data1.txt

# Part 2 - Line Editor Application

Write a line editor application LineEditor.java. If you run the program with no command line arguments, it will begin with an empty document. If you give the name of an existing file as a command line argument, it will open and load the contents of that file and store all lines in LineList object.

The following is a sample menu when the program starts.

Menu: m

Load File: load fileName appendOption(1-append, 0-new list)

Print: p

Display number of lines: lines

Display line: line lineNumber

Count words: words

Delete line: del lineNumberToDelete

Append line: a

Insert line: i lineNumberToInseart

Clear document: cls

Replace words: rep originalWord newWord

Save to a file: s fileName

Quiz: quit

Each command in the menu is associated with a method you implemented in the LineList.java. Your menu must include all commands defined in LineList class. The following table shows the menu command and its corresponding method call in the LineList class.

|  |  |  |
| --- | --- | --- |
|  | Menu command | Method in LineList class |
| 1 | load | load() |
| 2 | s | save() |
| 3 | a | addLine() |
| 4 | i | addLine() |
| 5 | words | words() |
| 6 | lines | lines() |
| 7 | del | delete() |
| 8 | p | print() |
| 9 | rep | replace() |
| 10 | line | line() |
| 11 | cls | empty() |

Run LineEditor.class, test all commands in the menu, take at least two screenshots of running results from your LineEditor application.

1. run the program with no command line arguments and all commands
2. run the program with an existing file as a command line argument and all commands

You may use the following file to do additional testing. (Optional)

* Data1.txt

Text

Description automatically generated