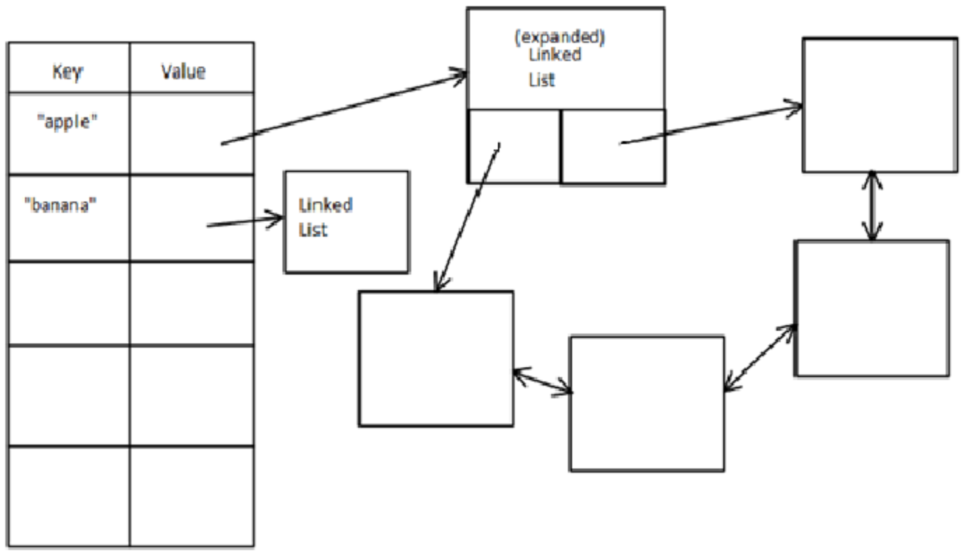
Data Structure Design

On evaluation of the data structures, it has been decided that a linked list in association with a hash table will be used. The linked list will be an original construction, in other words not imported from java.util. The reasoning behind this is that linked lists are not fixed length data structures. There is no way of determining beforehand how many instances of a word will be found in a book, therefore a variable sized data structure will be needed. Other variable length data structures such as queues and stacks were considered but they would not allow the flexibility that a linked list would since they are FIFO and FILO structures respectively.

There is a way of possibly using a primitive array such as an array of integers (int[]). This could only be implemented if the program first went through the book searching for each instance of a word and creating an array of that size. This would undoubtedly double the processing time.

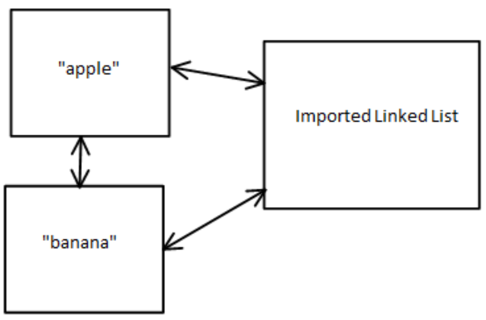
Creating a linked list class with nodes containing data relating to the task will allow accessing and processing data inside it much more efficient than it would be if an imported linked list was used. However, using a separate imported linked list for a different purpose in the program could prove useful and will be explained later in this document.

Hash tables have two fields: Keys and Values. It is obvious right from the beginning that each key in the hash table will consist of a word read from an index file of string type. This word will have an associated value containing a linked list of line numbers of where the word (key) appears in a book file.



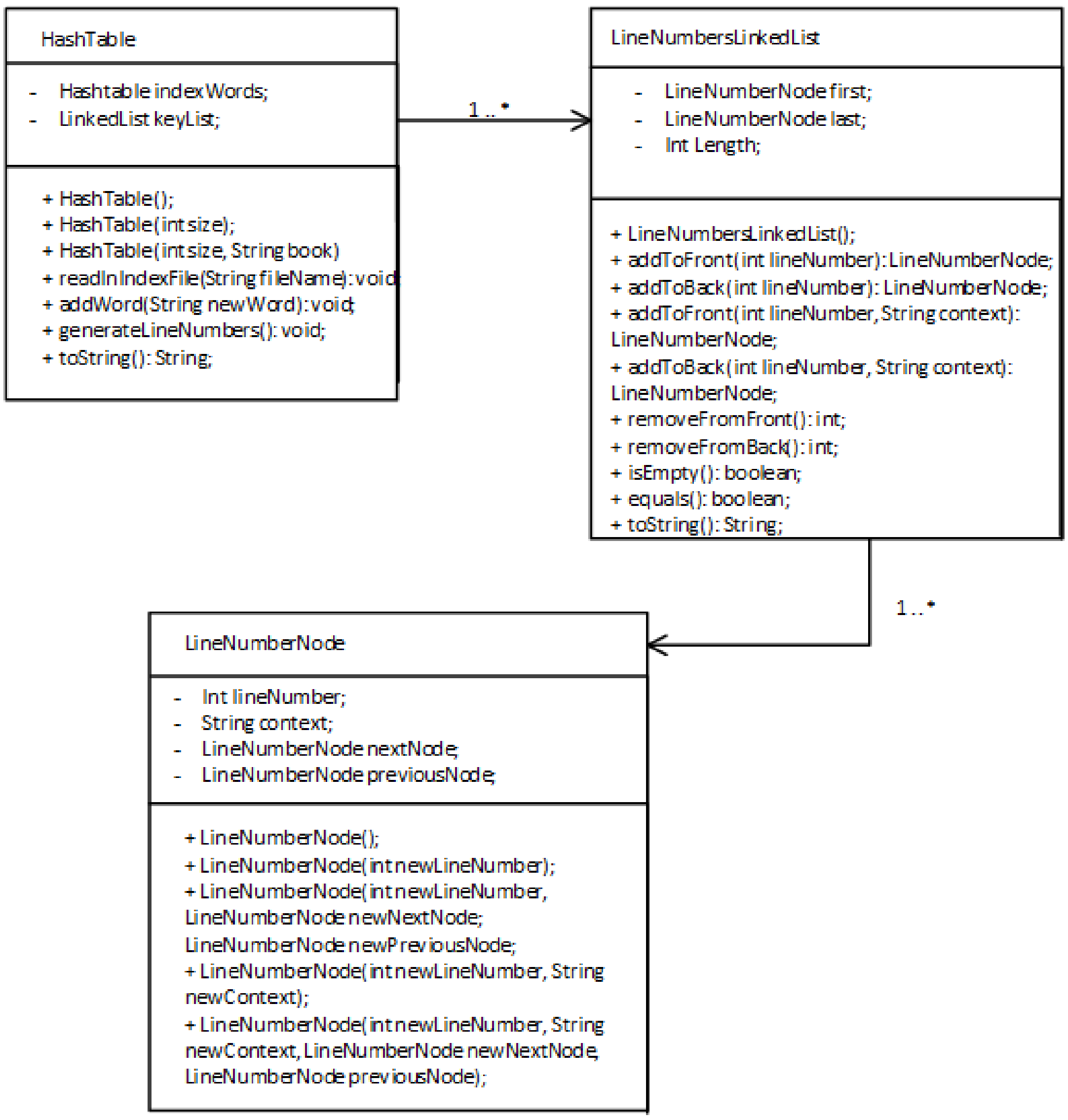
The above diagram describes the relationship between the hash table and the line numbers linked lists. The linked list will be doubly linked so that data can be extracted from the front and back without having to perform many looping operations. Each node in the linked list has 4 variables: Line number, context (The text in the line that has a match to a word in the hash table), a pointer to the next node and a pointer to the previous node.

The second linked list (which is imported) is used as a reference tool when loading in and finding index words. When the index file is read in, for each word a new instance in the hash table is created with a corresponding line number linked list. At the same time, this word is saved in the imported linked list. This linked list would then be used as a reference when comparing the book files' words to what the user would like to index. If there is a match in the linked list to a word in the book, the item in the hash table with that key (word in the linked list/found in the book) will be extracted, a line number would be added to its line number linked list and then the word item with its value (the new line numbers linked list) will be fed back into the hash table.



UML

Below is a UML diagram of how the words are connected via a hash table to their line numbers. For each item in the hash table there is an associated LineNumbersLinkedList and for each LineNumbersLinkedList there is LineNumberNodes each containing a line number and the context of that line.


HashTable
- Hashtable indexWords;
- LinkedList keyList;
+ HashTable();
+ HashTable(int size);
+ HashTable(int size, String book)
+ readInIndexFile(String fileName):void
+ addWord(String newWord):void
+ generateLineNumbers(): void;
+ toString(): String;
1 .. *
LineNumbersLinkedList
- LineNumberNode first;
- LineNumberNode last;
- Int Length;
+ LineNumbersLinkedList();
+ addToFront(int lineNumber): LineNumberNode;
+ addToBack(int lineNumber): LineNumberNode;
+ addToFront(int lineNumber, String context): LineNumberNode;
+ addToBack(int lineNumber, String context): LineNumberNode;
+ removeFromFront(): int;
+ removeFromBack(): int;
+ isEmpty(): boolean;
+ equals(): boolean;
+ toString(): String;
1 .. *
LineNumberNode
- Int lineNumber;
- String context;
- LineNumberNode nextNode;
- LineNumberNode previousNode;
+ LineNumberNode();
+ LineNumberNode(int newLineNumber);
+ LineNumberNode(int newLineNumber, LineNumberNode newNextNode; LineNumberNode newPreviousNode;
+ LineNumberNode(int newLineNumber, String newContext);
+ LineNumberNode(int newLineNumber, String newContext, LineNumberNode newNextNode, LineNumberNode previousNode);

METHODS

HashTable

- readInIndexFile(String fileName); This method reads each line in the index file and adds the word to the hash table by calling addWord(String newWord) inserts the word and a LineNumbersLinkedList into the hash table. It also saves each index word in an imported linked list to save as a reference to the hash table;
- generateLineNumbers(); This method goes through the book and checks each word against the items in the imported linked list. If it finds matches it adds the line number and the line context to its LineNumberLinkedList;
- toString(); Prints out all the LineNumberLinkedLists in all of the words in the hash table.

LineNumbersLinkedList

- addToFront(); and addToBack(); Both self explanitory with varying degrees of injected parameters;
- removeFromFront(); and removeFromBack(); Throw a CannotRemoveException if there are no items there.

Algorithm Descriptions

The first thing that the user is prompted with is to choose an index file consisting of one word per line to load into the hashtable.

Once the index file has been read in and the book has been set the user will have the option to load in a book file. Only after both files have been set can the user select to generate line numbers and context. This algorithm goes through the book line by line, splits the line into separate words, then compares each word to the words in the reference linked list. Here is the pseudocode:

```
Load in book
    While the book as another line
        Read line
        Convert line to all lower case
        Split the line into array of Strings
        For each word in the array of words
            Compare word with linked list of index words
            If there's a match
                Extract that word item from the hash table
                Add the line number and context (line text) to its linked list
                Feed the word item with edited linked list back in hash table
```

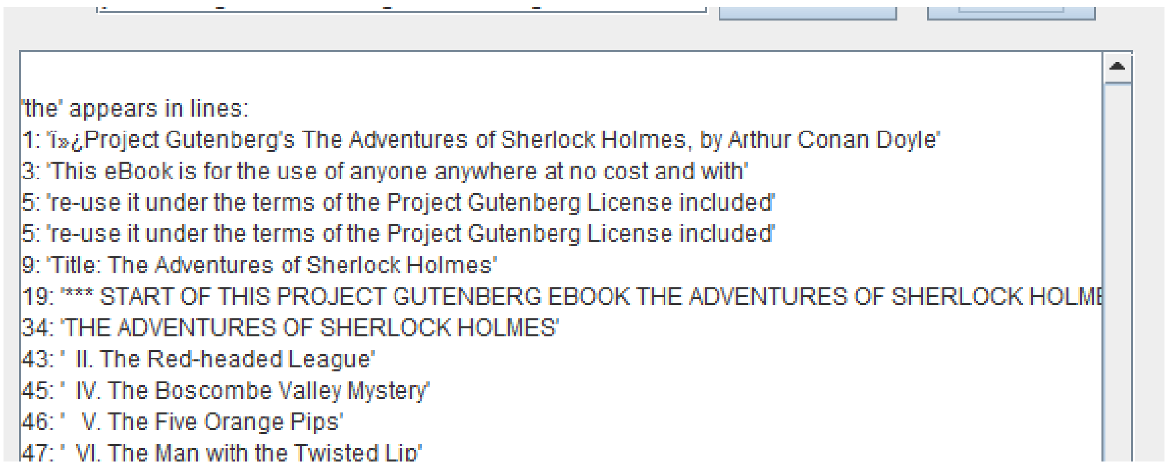
There are two main approaches that can be adopted at this stage, one being the example above and the other was to search through the book for each index word. This approach produces exactly the same results but clearly takes much longer for larger book files as the program would have to go through the book for each word instead of just the once.

When splitting the words on each line of the book line a piece of regular expression which splits the word according to blank space and punctuation characters is utilised. Here is the regex:

“([.,!?:;’\”-]|\\s)+”

Because of the way the line number generation method works, not only does it find words which have punctuation either side (like full stop and question marks) but it also recognises words which have capital letters at the beginning or throughout. This is due to conversion of the whole line being read in to lower case. However, a ‘true’ representation of the line (with capital letters and punctuation etc.) is kept and stored in the context if the line is chosen to contain a word so that the output the user gets is the true text and not a lower case transformation.

The following image is an example of how the program finds words of both lower and upper cases even if the word appears only as lower case in the index file. As you can see, the program has read in the word “the” and has recognised the word in the form “The” in line 1 and in the form “THE” which appears in line 19 of the book.

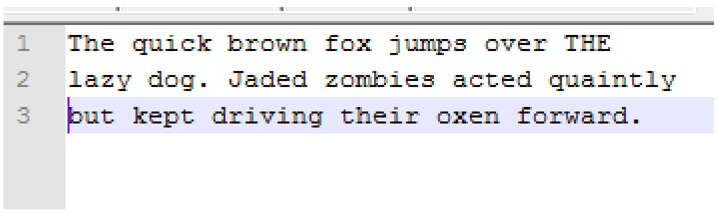


```
'the' appears in lines:
1: 'ï»¿Project Gutenberg's The Adventures of Sherlock Holmes, by Arthur Conan Doyle'
3: 'This eBook is for the use of anyone anywhere at no cost and with'
5: 're-use it under the terms of the Project Gutenberg License included'
5: 're-use it under the terms of the Project Gutenberg License included'
9: 'Title: The Adventures of Sherlock Holmes'
19: '*** START OF THIS PROJECT GUTENBERG EBOOK THE ADVENTURES OF SHERLOCK HOLME
34: 'THE ADVENTURES OF SHERLOCK HOLMES'
43: '  II. The Red-headed League'
45: '  IV. The Boscombe Valley Mystery'
46: '   V. The Five Orange Pips'
47: '  VI. The Man with the Twisted Lip'
```

Testing

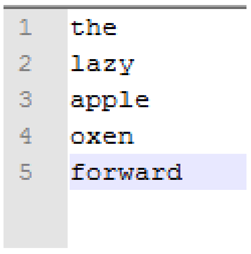
To test the program two files will be created, one representing a book and another representing an index file. Index file must be one word per line. Here are screenshots of the two files:

Book test file



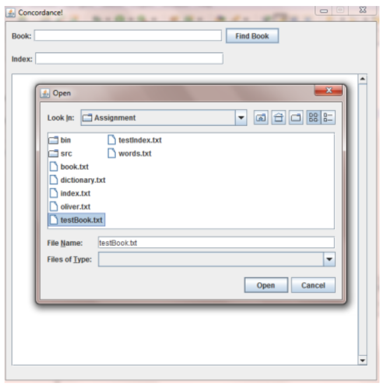
```
The quick brown fox jumps over THE
lazy dog. Jaded zombies acted quaintly
but kept driving their oxen forward.
```

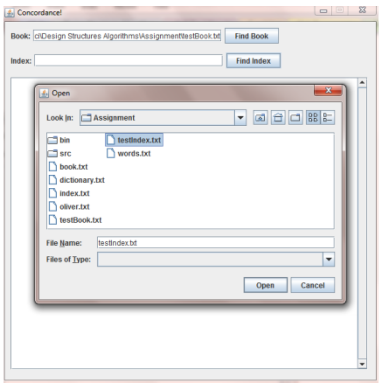
Index test file

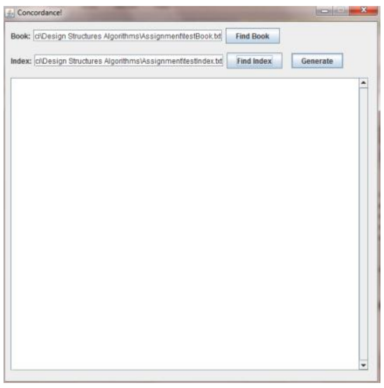


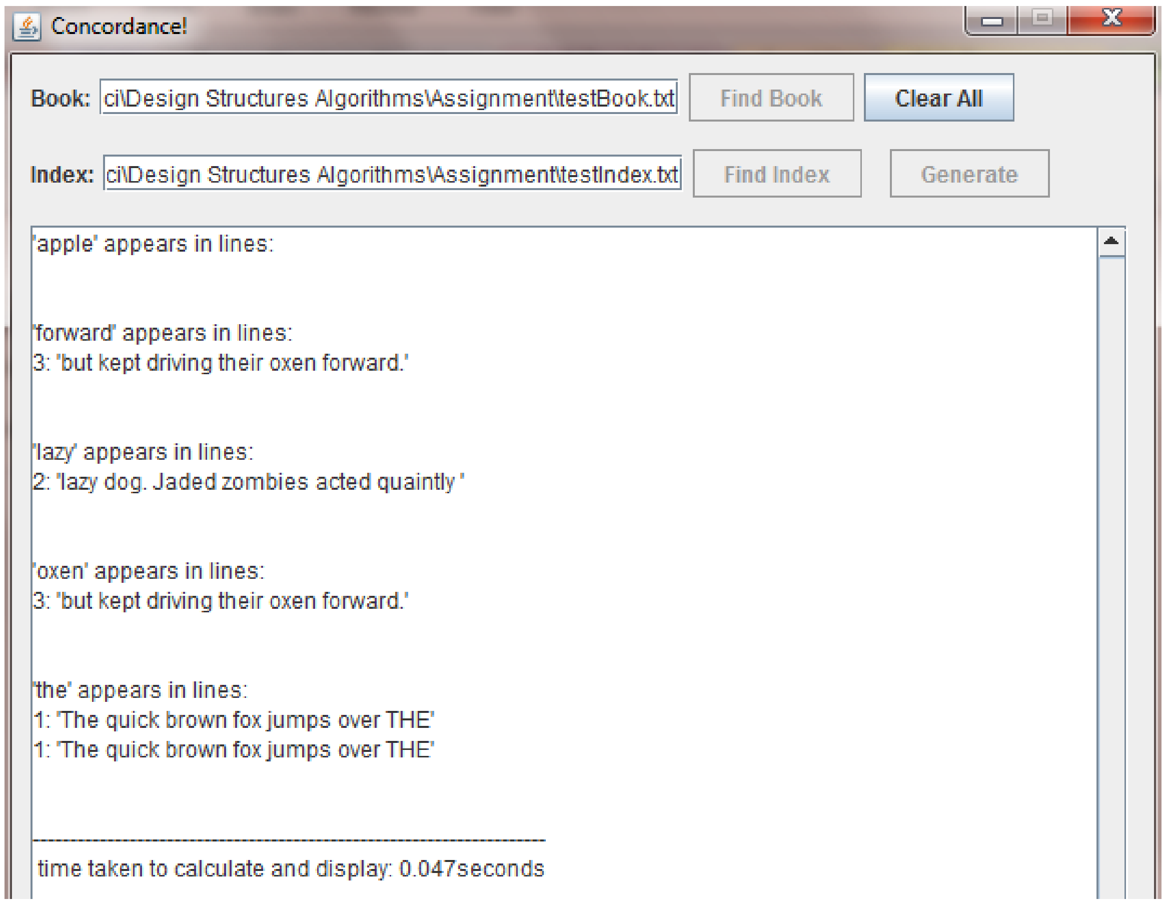
```
the
lazy
apple
oxen
forward
```

What should happen is that the word ‘the’ should appear twice, both in line 1; the word ‘lazy’ should appear once in line 2; ‘apple’ should never appear; ‘oxen’ should appear once in line 3 and ‘forward’ should appear once, also in line 3. The obstacles are that the word ‘the’ does appear twice but in different forms than is specified in the index file. Also, ‘forward’ appears with a full stop at the end. This is what happens:






Concordance!
Book: c:\Design Structures Algorithms\Assignment\testBook.txt
Find Book
Index: c:\Design Structures Algorithms\Assignment\testIndex.txt
Find Index
Generate

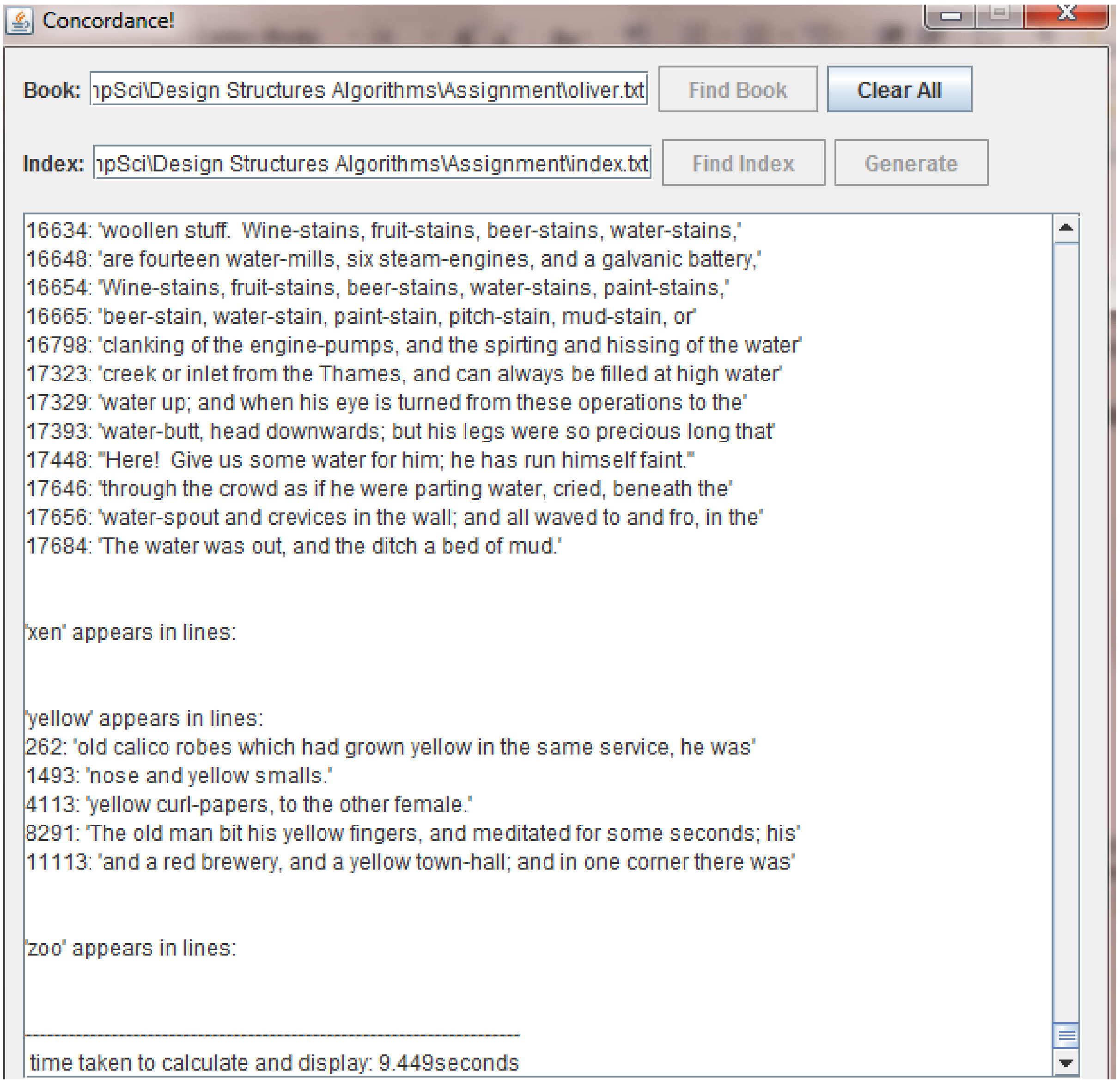

Concordance!
Book: ci\Design Structures Algorithms\Assignment\testBook.txt
Find Book
Clear All
Index: ci\Design Structures Algorithms\Assignment\testIndex.txt
Find Index
Generate

```
'apple' appears in lines:


'forward' appears in lines:
3: 'but kept driving their oxen forward.'


'lazy' appears in lines:
2: 'lazy dog. Jaded zombies acted quaintly '


'oxen' appears in lines:
3: 'but kept driving their oxen forward.'


'the' appears in lines:
1: 'The quick brown fox jumps over THE'
1: 'The quick brown fox jumps over THE'


----------------------------------------------------------------------
 time taken to calculate and display: 0.047seconds
```

It is evident that ‘apple’ is never found; ‘forward’ is found once on the second line even though the full stop follows it directly; ‘lazy’ is found once on the second line; ‘oxen’ is found once on the third line and ‘the’ appears twice on the same line (one with a capital ‘T’ and one with all capitals), thus printing the same context out twice. All results are printed out in alphabetical order.

Once the output has been generated, the only option available to the user is to ‘clear all’. This resets the program by restarting the GUI interface and erasing all data inside the data structures.

The following example is of how the program deals with reading through Oliver Twist with an index file of very common words. Here is a list of the words that appear in the index file: “and, bend, can, dog, ever, fantastic, going, happy, in, just, kill, line, man, loop, option, pen, query, run, sit, the, under, vent, water, xen, yellow, zoo”.



```
Book: ıpSci\Design Structures Algorithms\Assignment\oliver.txt
Index: ıpSci\Design Structures Algorithms\Assignment\index.txt

16634: 'woollen stuff.  Wine-stains, fruit-stains, beer-stains, water-stains,'
16648: 'are fourteen water-mills, six steam-engines, and a galvanic battery,'
16654: 'Wine-stains, fruit-stains, beer-stains, water-stains, paint-stains,'
16665: 'beer-stain, water-stain, paint-stain, pitch-stain, mud-stain, or'
16798: 'clanking of the engine-pumps, and the spirting and hissing of the water'
17323: 'creek or inlet from the Thames, and can always be filled at high water'
17329: 'water up; and when his eye is turned from these operations to the'
17393: 'water-butt, head downwards; but his legs were so precious long that'
17448: "Here!  Give us some water for him; he has run himself faint."
17646: 'through the crowd as if he were parting water, cried, beneath the'
17656: 'water-spout and crevices in the wall; and all waved to and fro, in the'
17684: 'The water was out, and the ditch a bed of mud.'


'xen' appears in lines:


'yellow' appears in lines:
262: 'old calico robes which had grown yellow in the same service, he was'
1493: 'nose and yellow smalls.'
4113: 'yellow curl-papers, to the other female.'
8291: 'The old man bit his yellow fingers, and meditated for some seconds; his'
11113: 'and a red brewery, and a yellow town-hall; and in one corner there was'


'zoo' appears in lines:


----------------------------------------------------------------------
time taken to calculate and display: 9.449seconds
```

The time it took to process the request is obviously far higher than the test files because of the sheer amount of times each word appears.

Source Code

```
//-----MAIN.JAVA-----//
/**
 *
 * @author alr16
 *
 */
```

```
public class Main {
      /**
       * @param args  String
       * Initializes the GUI
       */
      public static void main(String[] args) {
            @SuppressWarnings("unused")
            UserInterface gui = new UserInterface();
      }
}


//-----USERINTERFACE.JAVA-----//
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import javax.swing.*;

/**
 *
 * @author alr16
 * User interface enabling human interaction with concordance
application
 */

@SuppressWarnings("serial")
public class UserInterface extends JFrame implements ActionListener{

      JFileChooser chooser;
      HashTable controller;

      JPanel row1 = new JPanel();
      JLabel bookLabel = new JLabel("Book:", JLabel.LEFT);
      JTextField enterBook = new JTextField(28);
      JButton confirmBook = new JButton("Find Book");
      JButton clearAll = new JButton("Clear All");

      JPanel row2 = new JPanel();
      JLabel indexLabel = new JLabel("Index:", JLabel.LEFT);
      JTextField enterIndex = new JTextField(28);
      JButton confirmIndex = new JButton("Find Index");
      JButton generateLineNumbers = new JButton("Generate");

      JPanel row4 = new JPanel();
      JTextArea output = new JTextArea(30,52);
      JScrollPane scroll = new JScrollPane(output,
JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);


      /**
       * Creates a new user interface window and instanciates a new
HashTable
       */
      public UserInterface(){
            super("Concordance!");
            this.setSize(620, 620);
            this.setResizable(false);
            this.setLocation(50, 50);

            this.setDefaultCloseOperation(EXIT_ON_CLOSE);
            FlowLayout layout = new FlowLayout(FlowLayout.LEFT);
            this.setLayout(layout);

            controller = new HashTable();


            confirmBook.addActionListener(this);
            clearAll.addActionListener(this);
            row1.add(bookLabel);
            row1.add(enterBook);
            row1.add(confirmBook);
            row1.add(clearAll);
            clearAll.setVisible(false);
            this.add(row1);

            confirmIndex.addActionListener(this);
            row2.add(indexLabel);
            row2.add(enterIndex);
            row2.add(confirmIndex);
            confirmIndex.setVisible(false);
            row2.add(generateLineNumbers);
            generateLineNumbers.addActionListener(this);
            generateLineNumbers.setVisible(false);
            this.add(row2);



            row4.add(scroll);
            this.add(row4);

            this.setVisible(true);
      }


      /**
       * recognises when buttons are clicked and reponds in appropriate
way
       */
      @Override
      public void actionPerformed(ActionEvent e) {

            chooser = new JFileChooser();
            chooser.setCurrentDirectory(new java.io.File("."));

      chooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
            chooser.setAcceptAllFileFilterUsed(false);
            System.out.println(e.getActionCommand());
            if
(e.getActionCommand().equals(this.generateLineNumbers.getActionCommand(
))){
                  long init = System.currentTimeMillis();
                  controller.readInIndexFile(this.enterIndex.getText());
                  controller.setBook(this.enterBook.getText());
                  controller.generateLineNumbers();
```

```
                this.output.setText(controller.toString());
                long ending = System.currentTimeMillis();
                float timeTaken = (float)(ending - init)/1000;
                this.output.setText(this.output.getText() +
                            "-----------------------------------------
--------------------------" +
                            "\n time taken to calculate and display: "
+ timeTaken + "seconds");
                this.clearAll.setVisible(true);
                this.confirmBook.setEnabled(false);
                this.confirmIndex.setEnabled(false);
                this.generateLineNumbers.setEnabled(false);
            }
            else if
(e.getActionCommand().equals(this.clearAll.getActionCommand())){
                controller.clearAll();
                this.enterBook.setText("");
                this.enterIndex.setText("");
                this.output.setText("");
                this.confirmIndex.setVisible(false);
                this.generateLineNumbers.setVisible(false);
                this.clearAll.setVisible(false);
                this.confirmBook.setEnabled(true);
                this.confirmIndex.setEnabled(true);
                this.generateLineNumbers.setEnabled(true);
            }
            else if (chooser.showOpenDialog(this) ==
JFileChooser.APPROVE_OPTION){
                File file = chooser.getSelectedFile();
                String fileName = file.getAbsolutePath();
                if
(e.getActionCommand().equals(this.confirmBook.getActionCommand())){
                    this.enterBook.setText(fileName);
                    this.confirmIndex.setVisible(true);
                }
                else if
(e.getActionCommand().equals(this.confirmIndex.getActionCommand())){
                    this.enterIndex.setText(fileName);
                    this.generateLineNumbers.setVisible(true);
                }
            }
        }
}


//-----HASHTABLE.JAVA-----//
import java.util.Collections;
import java.util.Hashtable;
import java.util.LinkedList;
import java.util.Scanner;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;

/**
 * HashTable class holds a LinkedList of words from an index file
 * and a Hastable in which the keys are index words and values are
 * LineNumberLinkedLists of line numbers and content
 *
 * @author alr16
 *
 */
@SuppressWarnings("rawtypes")
public class HashTable {

      private Hashtable<String, LineNumbersLinkedList> indexWords;
      private FileReader fR;
      private BufferedReader bR;
      private String book;
      private File f;
      private Scanner input;
      private LinkedList keyList;


      //CONSTRUCTORS
      /**
       * Blank constructor creates new instances of Hashtable and
       * LinkedList
       */
      public HashTable(){
            this.indexWords = new Hashtable<String,
LineNumbersLinkedList>();
            keyList = new LinkedList();
      }

      /**
       * Takes an int size to set the size of the Hashtable
       * @param size  int
       */
      public HashTable(int size){
            this.indexWords = new Hashtable<String,
LineNumbersLinkedList>(size);
            keyList = new LinkedList();
      }

      /**
       * Takes an int to set size of Hashtable and String to set Book
       * file path
       * @param size  int
       * @param book  String
       */
      public HashTable(int size, String book){
            this.indexWords = new Hashtable<String,
LineNumbersLinkedList>(size);
            this.book = book;
            keyList = new LinkedList();
      }


      //METHODS
      /**
       * Reads in an index file full of words that will be searched for
and
       * indexed in the specified book
```

```
	 * @param fileName  String
	 */
	public void readInIndexFile(String fileName){
		String line = "";
		try{
			fR = new FileReader(fileName);
			bR = new BufferedReader(fR);
			File f = new File(fileName);
		   Scanner input = new Scanner(f);
			while(input.hasNextLine()){
				line = bR.readLine();
				line = line.toLowerCase();
				this.addWord(line);
				line = input.nextLine();
			}
			fR.close();
		}
		catch(Exception e){
			e.printStackTrace();
		}

	}

	/**
	 * Adds the word to the LinkedList of keys as a reference to
	 * the keys in the Hashtable. Then creates a new
LineNumbersLinkedList
	 * and uses it to create a new entry in the Hashtable with the
new word
	 * @param newWord  String
	 */
	@SuppressWarnings("unchecked")
	public void addWord(String newWord){
		this.keyList.add(newWord);
		LineNumbersLinkedList arrayOfLineNumbers = new
LineNumbersLinkedList();
		this.indexWords.put(newWord, arrayOfLineNumbers);
	}

	/**
	 * For each line in the book, this alorithm splits the line and
compares
	 * each word to every word in the LinkedList of keys. If a match
is found,
	 * the Hashtable entry is extracted, it's LineNumbersLinkedList
is edited
	 * to include the new line number and context and is fed back
into the
	 * Hashtable.
	 */
	public void generateLineNumbers(){
		try {
			f = new File(this.book);
			input = new Scanner(f);
			String lineContent;
			String lineContentLowerCase;
			String[] splitLine;
			int lineNumber = 0;
```

```
                LineNumbersLinkedList list;
                while (input.hasNextLine()){
                lineNumber++;
                      lineContent = input.nextLine();
                      lineContentLowerCase =
lineContent.toLowerCase();
                      splitLine =
lineContentLowerCase.split("([.,!?:;'\"-]|\\s)+");
                      for (int wordInLine = 0;
wordInLine<splitLine.length; wordInLine++){
                            if
(this.keyList.contains(splitLine[wordInLine])){
                                  list =
this.indexWords.get(splitLine[wordInLine]);
                                  list.addToBack(lineNumber,
lineContent);

      this.indexWords.put(splitLine[wordInLine], list);
                            }
                      }
                }

          } catch (FileNotFoundException e) {
                e.printStackTrace();
          }
    }

    //GETTERS AND SETTERS
    /**
     * Returns the entire Hashtable
     * @return  Hashtable
     */
    public Hashtable<String, LineNumbersLinkedList> getIndexWords() {
          return indexWords;
    }

    /**
     * Sets the entire Hashtable
     * @param indexWords  Hashtable
     */
    public void setIndexWords(Hashtable<String,
LineNumbersLinkedList> indexWords) {
          this.indexWords = indexWords;
    }

    /**
     * Gets the current book path name
     * @return  book String
     */
    public String getBook() {
          return book;
    }

    /**
     * Sets the book path name
     * @param fileName  String
     */
    public void setBook(String book) {
            this.book = book;
      }

      /**
       * Returns every item in the Hashtable
       * @return  String
       */
      @SuppressWarnings("unchecked")
      public String toString(){
            Collections.sort(this.keyList);
            String returnable = "";
            for (int item = 0; item < this.keyList.size(); item++){
                  LineNumbersLinkedList list =
(LineNumbersLinkedList)this.indexWords.get(this.keyList.get(item));
                  returnable = returnable + "'" + this.keyList.get(item)
+ "' appears in lines:\n" + list.toString() + "\n\n";
            }
            return returnable;
      }

      public void clearAll(){
            indexWords.clear();
            keyList.clear();
      }
}


//-----LINENUMBERSLINKEDLIST.JAVA-----//
/**
 *
 * @author alr16
 * Linked list of LineNunberNodes
 */
public class LineNumbersLinkedList {

      private LineNumberNode first;
      private LineNumberNode last;
      private int length;

      //CONSTRUCTORS
      /**
       * Blank constructor which initiates the linked list by creating
a first
       * and last node
       */
      public LineNumbersLinkedList(){
            this.first = new LineNumberNode();
            this.last = new LineNumberNode();
            this.first.setNextNode(this.last);
            this.last.setPreviousNode(this.first);
            this.length = 0;
      }


      //METHODS
      /**
       * Adds a new node to the front of the linked list with its line
number
```

```
	 * @param lineNumber  int
	 * @return newNode  LineNumberNode
	 */
	public LineNumberNode addToFront(int lineNumber){
		LineNumberNode newNode = new LineNumberNode(lineNumber);
		newNode.setNextNode(this.first.getNextNode());
		newNode.getNextNode().setPreviousNode(newNode);
		this.first.setNextNode(newNode);
		newNode.setPreviousNode(this.first);
		this.length++;
		return this.first.getNextNode();
	}


	/**
	 * Adds a new node to the front of the linked list with its line
number
	 * @param lineNumber  int
	 * @return newNode  LineNumberNode
	 */
	public LineNumberNode addToBack(int lineNumber){
		LineNumberNode newNode = new LineNumberNode(lineNumber);
		newNode.setPreviousNode(this.last.getPreviousNode());
		newNode.getPreviousNode().setNextNode(newNode);
		newNode.setNextNode(this.last);
		this.last.setPreviousNode(newNode);
		this.length++;
		return this.last.getPreviousNode();
	}


	/**
	 * Adds a new node to the front of the linked list with its line
number
	 * and context
	 * @param lineNumber  int
	 * @param context  context
	 * @return newNode  LineNumberNode
	 */
	public LineNumberNode addToFront(int lineNumber, String context){
		LineNumberNode newNode = new LineNumberNode(lineNumber,
context);
		newNode.setNextNode(this.first.getNextNode());
		newNode.getNextNode().setPreviousNode(newNode);
		this.first.setNextNode(newNode);
		newNode.setPreviousNode(this.first);
		this.length++;
		return this.first.getNextNode();
	}


	/**
	 * Adds a new node to the back of the linked list with its line
number
	 * and context
	 * @param lineNumber  int
	 * @param context  String
	 * @return newNode LineNumberNode
```

```
         */
        public LineNumberNode addToBack(int lineNumber, String context){
              LineNumberNode newNode = new LineNumberNode(lineNumber,
context);
              newNode.setPreviousNode(this.last.getPreviousNode());
              newNode.getPreviousNode().setNextNode(newNode);
              newNode.setNextNode(this.last);
              this.last.setPreviousNode(newNode);
              this.length++;
              return this.last.getPreviousNode();
        }


        /**
         * Removes an item from the front of the linked list
         * @return lineNumber  int
         * @throws CannotRemoveException
         */
        public int removeFromFront() throws CannotRemoveException{
              LineNumberNode currentNode;
              int removedNumber;
              if (this.isEmpty())
                    throw new CannotRemoveException("Cannot remove from
front. List is empty");
              else{
                    currentNode = this.first.getNextNode();
                    this.first.setNextNode(currentNode.getNextNode());
                    currentNode.getNextNode().setPreviousNode(this.first);
                    currentNode.setPreviousNode(null);
                    currentNode.setNextNode(null);
                    removedNumber = currentNode.getLineNumber();
                    this.length--;
              }
              return removedNumber;
        }


        /**
         * Removes an item from the back of the linked list
         * @return lineNumber  int
         * @throws CannotRemoveException
         */
        public int removeFromBack() throws CannotRemoveException{
              LineNumberNode currentNode;
              int removedNumber;
              if (this.isEmpty())
                    throw new CannotRemoveException("Cannot remove from
back. List is empty");
              else{
                    currentNode = this.last.getPreviousNode();

        this.last.setPreviousNode(currentNode.getPreviousNode());
                    currentNode.getPreviousNode().setNextNode(this.last);
                    currentNode.setNextNode(null);
                    currentNode.setPreviousNode(null);
                    removedNumber = currentNode.getLineNumber();
                    this.length--;
              }
```

```
      return removedNumber;
}


/**
 * Returns true if the linked list was empty
 * @return empty  boolean
 */
public boolean isEmpty(){
      boolean empty = false;
      if (this.first.getNextNode() == this.last){
            empty = true;
      }
      return empty;
}


/**
 * Returns true if this LineNumbersLinkedList is equal to
 * the inserted parameter
 * @param otherList  LineNumbersLinkedList
 * @return equals  bolean
 */
public boolean equals(LineNumbersLinkedList otherList){
      return (this == otherList);
}

//GETTERS AND SETTERS
/**
 * Gets the first node from the LineNumbersLinkedList
 * @return frontNode  LineNumberNode
 */
public LineNumberNode getFront(){
      return this.first.getNextNode();
}


/**
 * Gets the last node from the LineNumbersLinkedList
 * @return backNode  LineNumberNode
 */
public LineNumberNode getBack(){
      return this.last.getPreviousNode();
}


/**
 * Returns the length of the LineNumbersLinkedList
 * @return length  int
 */
public int getLength(){
      return this.length;
}


/**
 * Returns each nodes' lineNumber and context
 * @return returnable  String
```

```
	 */
	public String toString(){
		String returnable = "";
		LineNumberNode currentNode = this.first.getNextNode();
		for (int i=0; i<this.length; i++){
			returnable = returnable + currentNode.getLineNumber()
+ ": '" + currentNode.getContext() + "'\n";
			currentNode = currentNode.getNextNode();
		}
		return returnable;
	}
}


//-----LINENUMBERNODE-----//
/**
 *
 * @author alr16
 * Class containing a line number and context in a LineNumberLinkedList
 */
public class LineNumberNode {

	private int lineNumber;
	private String context;
	private LineNumberNode nextNode;
	private LineNumberNode previousNode;

	//CONSTRUCTORS
	/**
	 * Blank construcotr
	 */
	public LineNumberNode(){
	}

	/**
	 * Creates an instance of the lineNumber contained within this
node
	 * @param newLineNumber int
	 */
	public LineNumberNode(int newLineNumber){
		this.lineNumber = newLineNumber;
	}

	/**
	 * Creates an instance of the lineNumber contained within this
node and
	 * the pointers to the next and previous nodes
	 * @param newLineNumber  int
	 * @param newNextNode  LineNumberNode
	 * @param newPreviousNode  LineNumberNode
	 */
	public LineNumberNode(int newLineNumber,LineNumberNode
newNextNode, LineNumberNode newPreviousNode){
		this.lineNumber = newLineNumber;
		this.nextNode = newNextNode;
		this.previousNode = newPreviousNode;
	}
	/**
	 * Creates a new instance of this lineNumber and context
	 * @param newLineNumber  int
	 * @param newContext  String
	 */
	public LineNumberNode(int newLineNumber, String newContext){
		this.lineNumber = newLineNumber;
		this.context = newContext;
	}

	/**
	 * Creates a new instance of this lineNumber, context, nextNode
and previousNode
	 * @param newLineNumber  int
	 * @param newContext  String
	 * @param newNextNode  LineNumberNode
	 * @param newPreviousNode  LineNumberNode
	 */
	public LineNumberNode(int newLineNumber, String newContext,
LineNumberNode newNextNode, LineNumberNode newPreviousNode){
		this.lineNumber = newLineNumber;
		this.context = newContext;
		this.nextNode = newNextNode;
		this.previousNode = newPreviousNode;
	}


	//METHODS
	//GETTERS AND SETTERS
	/**
	 * returns this lineNumber
	 * @return lineNumber  int
	 */
	public int getLineNumber() {
		return lineNumber;
	}

	/**
	 * sets this lineNumber
	 * @param lineNumber  int
	 */
	public void setLineNumber(int lineNumber) {
		this.lineNumber = lineNumber;
	}

	/**
	 * gets the next node
	 * @return nextNode  LineNumberNode
	 */
	public LineNumberNode getNextNode() {
		return nextNode;
	}

	/**
	 * sets the next node
	 * @param nextNode  LineNumberNode
	 */
	public void setNextNode(LineNumberNode nextNode) {
		this.nextNode = nextNode;
	}

	/**
	 * gets the previous node
	 * @return previousNode  LineNumberNode
	 */
	public LineNumberNode getPreviousNode() {
		return previousNode;
	}

	/**
	 * sets the previous node
	 * @param previousNode  LineNumberNode
	 */
	public void setPreviousNode(LineNumberNode previousNode) {
		this.previousNode = previousNode;
	}

	/**
	 * gets this context
	 * @return context  String
	 */
	public String getContext() {
		return context;
	}

	/**
	 * sets this context
	 * @param context  String
	 */
	public void setContext(String context) {
		this.context = context;
	}
}


//-----CANNOTREMOVEEXCEPTION.JAVA-----//
/**
 *
 * @author alr16
 *
 */
@SuppressWarnings("serial")
public class CannotRemoveException extends Exception{

	/**
	 * This error occurs when an item cannot be removed from the
	 * LineNumbersLinkedList
	 * @param errorMessage  String
	 */
	public CannotRemoveException(String errorMessage){
		System.err.println(errorMessage);
	}
}
```