DOCUMENT INDEXER

CS3381 OBJECT ORIENTED PROGRAMMING LABORATORY

A MINI PROJECT REPORT

Submitted by

KARTHIKA R (953622104045)

NANDHINI DEVI M (953622104064)

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAMCO INSTITUTE OF TECHNOLOGY

RAJAPALAYAM

JAN 2024

ABSTRACT

An index is a vital component of a book, serving as a comprehensive roadmap to the content within its pages. This abstract outlines the design and functionality of an application created to generate an index for a given document. The program reads input from a specified text file, meticulously parsing the content to compile an index. A concordance provides a detailed catalog of every word found within a document, accompanied by the line numbers where each word appears. This abstract describes the design and implementation of a program dedicated to creating a concordance for a given text file.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

An index serves as a systematic guide, offering readers a structured pathway to locate specific terms within a document efficiently. By compiling keywords and their corresponding page numbers, the index acts as a roadmap, facilitating quick access to relevant sections of the text. This section of the report explores the intricacies of creating an index, detailing the processes involved in identifying significant words, excluding common terms, and generating an organized reference system. The importance of an index for researchers, scholars, and general readers is underscored, highlighting its role in streamlining information retrieval and enhancing the overall user experience.

CHAPTER 2

PROJECT DESCRIPTION

2.1 Index for a book:

Keyword Extraction: The program reads a specified input text file and identifies significant words, excluding common terms and words with fewer than three characters.

Output to file: The generated index is written to an output file, offering readers a quick and structured reference to locate specific terms within the document.

2.2 Concordance for a book:

Exclusion of common terms: Similar to the index, common terms ae excluded and words with fewer than three characters are ignored.

Output to file: The resulting concordance is written to an output file, providing readers with a detailed catalog for every word and its respective line numbers, offering insights into the distribution and usage of terms.

2.3 MODULES IMPLEMENTED

2.3.1 Module for index generation('createCombinedIndex'):

Objective: Generates an index for a given document.

Functionality:

- Reads two input files.
- Processes each file, identifying significant words and associating them with page numbers.
- Writes the resulting index to an output file.

KeyComponents:

'processIndexFile': Common functionality for processing each input file. Utilizes a 'Hashmap' to store words and corresponding page numbers.

2.3.2 Module for concordance generation(‘createConcordance’):

Objective: Generates a concordance for a given document.

Functionality:

- Reads an input file.
- Processes the file, identifying significant words and associating them with line numbers.
- Excludes common words and those with less than three characters.
- Writes the resulting concordance to an output file.

KeyComponents:

- Utilizes a ‘Hashmap’ to store words and corresponding line numbers.
- ‘Append’ parameter controls whether the concordance is appended to an existing file.

CHAPTER 3

TECHNOLOGY STACK

3.1 Programming Language: Java

Java, renowned for its versatility and platform independence, serves as a robust foundation for developing a Document Indexer. This report provides an overview of how Java is employed in the implementation of a Document Indexer, a tool designed to organize and facilitate efficient retrieval of information within textual documents. A class in java is defined as template or blueprint of an object. Here we use the class called "**Application**". The method in Java is a collection of instructions that performs a specific task. It provides the reusability of code Here we use the method called "Inputfile", "Outputfile". An object in Java is an instance of a class. Object used here are grid, inputlabel, resultTextArea, browse button ,scene and tokenizer.

3.2 GUI Library: JAVAFX

The graphical user interface (GUI) of the application is built using JavaFX. JavaFX provides a set of APIs for creating rich and interactive user interfaces for desktop applications.

The following controls and layouts are used to implement Document Indexer.

Controls and Layout:

1.Grid Pane:

Grid pane is a layout in which it arranges its children in the grid. It develops to create complex layouts by arranging components in rows and columns.

Syntax:

GridPane grid =new GridPane();

2.Button:

Button is a graphical user interface component, it is used to trigger an action when we click the button. In the Document Indexer, buttons are employed to browse the input file.

Syntax:

Button browseButton=new Button(“Browse”);

3.Label:

Label is a non-editable text control for displaying information. It is used prompt the user for input or to display the select input file.

Syntax:

Label inputLabel=new Label(“select input file”);

4.TextField:

TextArea is a user interface control that provides a multi-line text input area. It is suitable for handling a larger amount of text compared to a single-line input control like TextField.

Syntax:

TextArea resultTextArea=new TextArea();

5.Event Handling:

An event is a specific occurrence or action that takes place during the execution of a program. It include user interactions like mouse clicks, keybroad input, button press, window resizing.

Syntax:

```
Buttonname.setOnAction(eve->{
        // statements;
});
```

In this program it is used to choose the input file for creating index.

CHAPTER 4

RESULTS

The home page of the Documentation index system contains “Load Document”, ”Create index” button as shown in figure 4.1

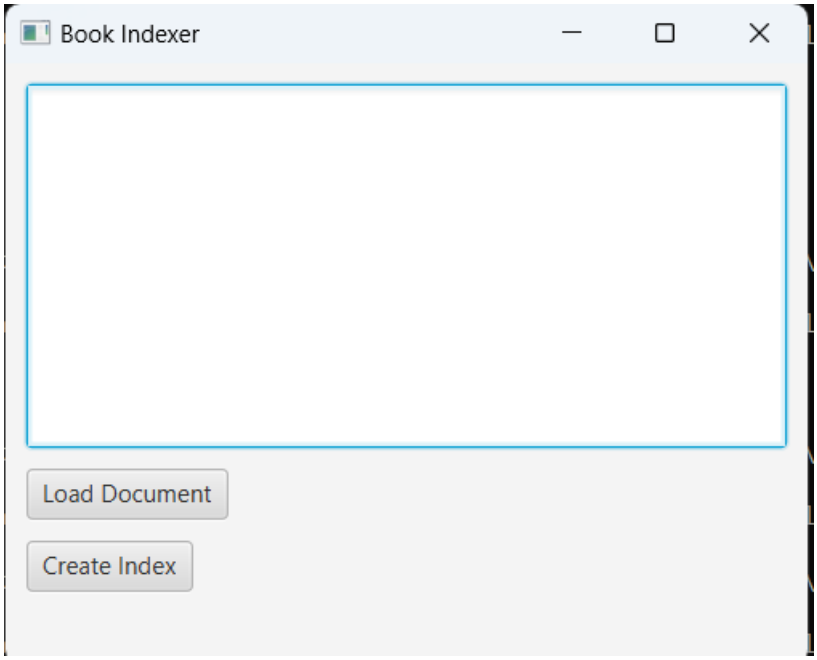

Fig 4.1: Home page

The Load Document button prompts the inputs from the user such as select the input file on the top of index as shown in figure 4.2. When the input file **cs.txt** is selected its generate to select a program. If the program is selected automatically its generated it displays the index generated successfully.

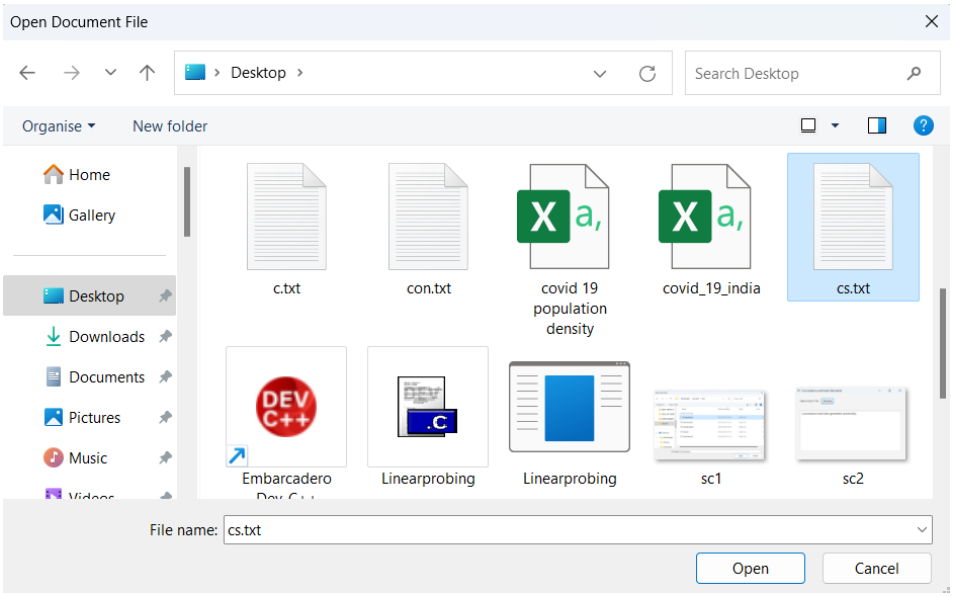

Fig 4.2 Select the input file

After selecting the input file , input was given as **"hi, hello, welcome"** as shown in figure 4.3.

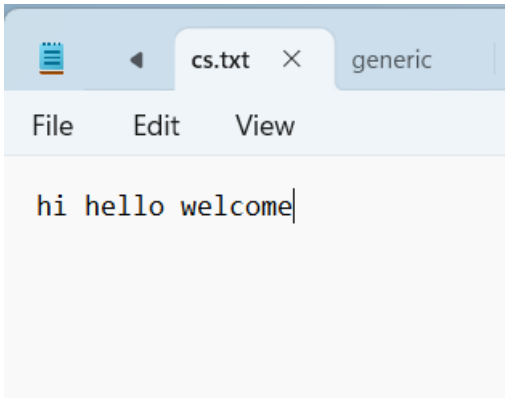

Fig 4.3: Content Added

Here index was created successfully as shown in figure 4.4.

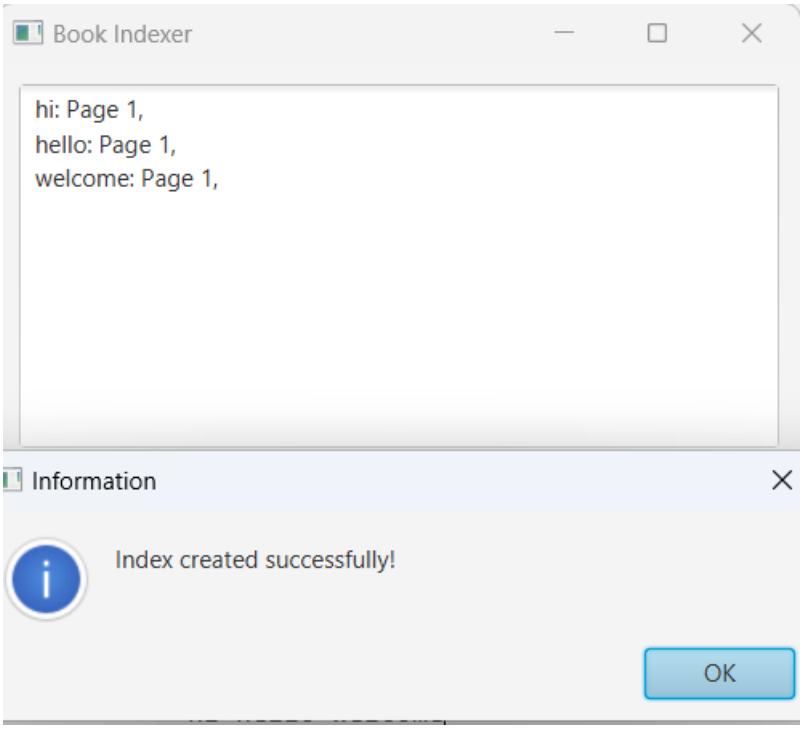

Fig 4.4: Index created

The home page of the Documentation index system contains “Load Document”, ”Count Occurences” button as shown in figure 4.5

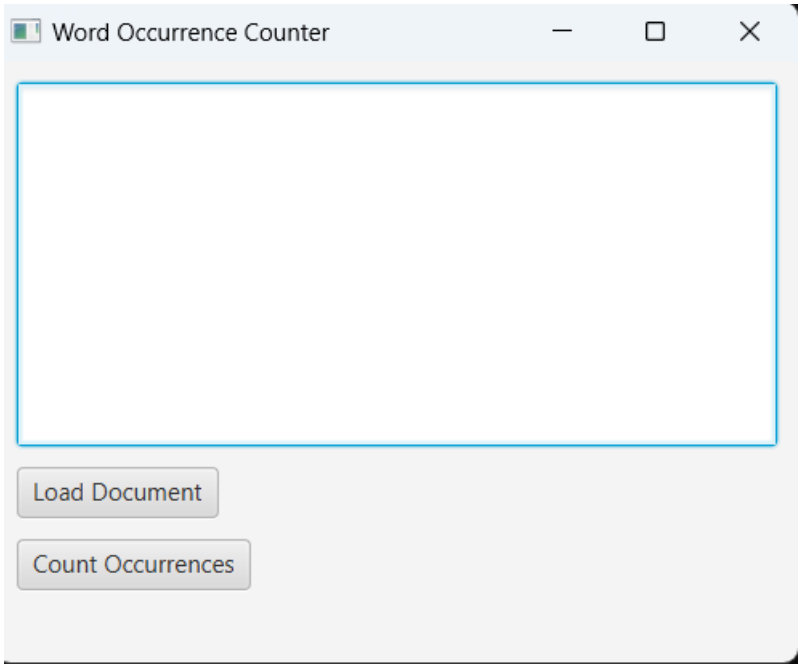

Fig 4.5: Word Occurrence Counter

The Load Document button prompts the inputs from the user such as select the input file on the top of index as shown in figure 4.6. When the input file **c.txt** is selected its generate to select a program. If the program is selected automatically its generated it displays the concordance generated successfully.

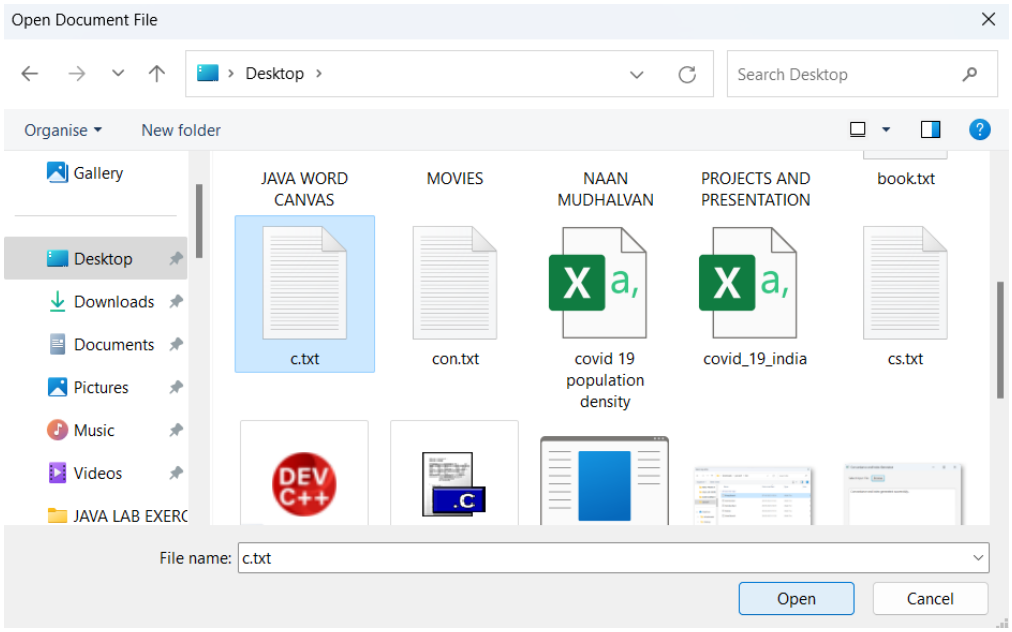

Fig 4.6: Select the Input file

After selecting the input file , input was given as **"nandhini, karthika"** as shown in figure 4.7.

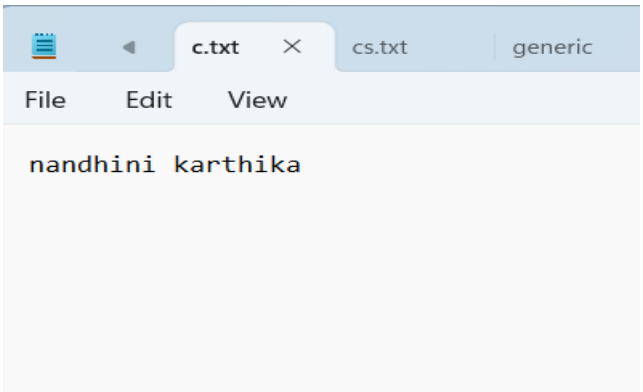

Fig 4.7: Content Added

Here concordance was created successfully as shown in figure 4.8,figure 4.9.

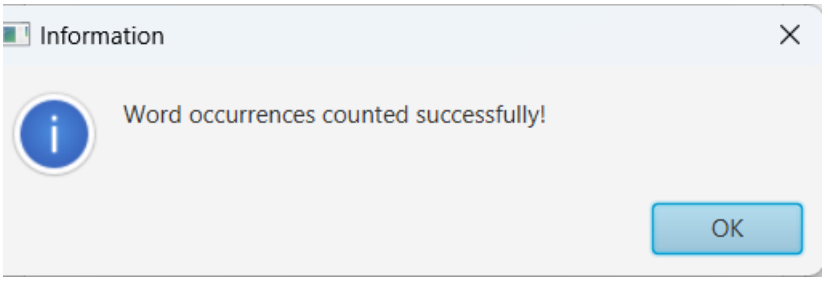

Fig 4.8: Concordance created

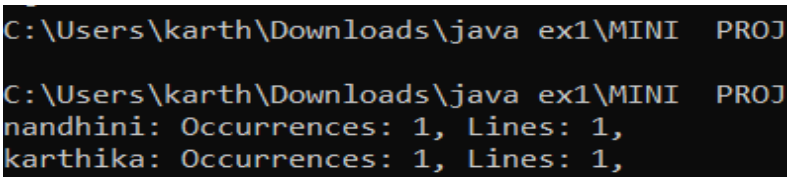

Fig 4.9: Occurrence and lines counted

CHAPTER 5

CONCLUSION

In conclusion, the design of the application to create an index and concordance involves reading a document from an input file and generating structured input files based on specific criteria. The user should be able to select the input and output files when running the program, providing flexibility in processing different documents. To achieve this, the program needs to handle file input and output operations efficiently.

The provided JavaFX application, named "Concordance and Index Generator," offers a user-friendly interface for generating concordance and index data from a selected input file. The program utilizes JavaFX components to create a simple GUI that allows users to choose an input file through a file browsing mechanism.

Upon selecting an input file, the application processes the file's content, generating both a concordance and an index. The generated concordance and index are then written to separate output files, namely "concordance.txt" and "index.txt," respectively. This ensures that the results are saved for later reference or analysis.

The application demonstrates effective use of Java I/O operations for reading and writing files, as well as proper handling of exceptions. The graphical user interface provides a seamless user experience, allowing users to interact with the program easily.

To further improve the application, additional features could be considered, such as the ability to customize output file names, display progress indicators during processing, or handle different file formats. Overall, the current implementation provides a solid foundation for a Concordance and Index Generator application in a JavaFX environment.

Appendix

CODING

```
package com.mycompany.mavenproject4;
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextArea;
import javafx.scene.layout.GridPane;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import java.io.*;
import java.util.HashMap;
import java.util.Map;
import java.util.StringTokenizer;

public class App extends Application

{

   private File inputFile;

   private File outputFile;

   public void start(Stage primaryStage)

{

      primaryStage.setTitle("Concordance and Index Generator");

      GridPane grid = new GridPane();

      grid.setPadding(new Insets(20, 20, 20, 20));

      grid.setVgap(10);
```

```
grid.setHgap(10);

Label inputLabel = new Label("Select Input File:");

grid.add(inputLabel, 0, 0);

TextArea resultTextArea = new TextArea();

resultTextArea.setEditable(false);

resultTextArea.setWrapText(true);
grid.add(resultTextArea, 0, 3, 2, 1);
Button browseButton = new Button("Browse");
grid.add(browseButton, 1, 0);
browseButton.setOnAction(e -> {
FileChooser fileChooser = new FileChooser();
fileChooser.setTitle("Select Input File");
inputFile = fileChooser.showOpenDialog(primaryStage);
 if (inputFile != null)
{
            generateConcordanceAndIndex(inputFile);
            resultTextArea.setText("Concordance and Index generated
            successfully.");

 }

 else

   {

      resultTextArea.setText("Please select a valid input file.");

   }

});

   Scene scene = new Scene(grid, 500, 300);
```

```
        primaryStage.setScene(scene);

        primaryStage.show();

}

    private void generateConcordanceAndIndex(File inputFile) {

   Map<String, String> concordance = new HashMap<>();

   Map<String, String> index = new HashMap<>();

    try (BufferedReader reader = new BufferedReader(new FileReader(inputFile)))

      {
              String line;
              int lineNumber = 1;
         while ((line = reader.readLine()) != null)
        {
                StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens())

            {
                        String word = tokenizer.nextToken().toLowerCase();

                        if (!word.equals("the") && word.length() >= 3)
                       {
                                concordance.merge(word,String.valueOf(line
                               Number),(prev, current) -> prev + ", " + current);

                                index.putIfAbsent(word, String.valueOf(lineNumber));

                        }

                }

            lineNumber++;
```

```
        }
        try (PrintWriter concordanceWriter = new PrintWriter(new
        FileWriter("concordance.txt")))
        {
                for (Map.Entry<String, String> entry : concordance.entrySet())
                {
                        concordanceWriter.println(entry.getKey() + ": " + entry.getValue());
                }
        }
        try (PrintWriter indexWriter = new PrintWriter(new        FileWriter("index.txt")))
        {
           for (Map.Entry<String, String> entry : index.entrySet())
            {
                        indexWriter.println(entry.getKey() + ": " + entry.getValue());
            }
        }
        catch (IOException e)
        {
                e.printStackTrace();
      }
   }
        public static void main(String[] args) {
      launch(args);
   }
}
```

REFERENCES

1. **www.stackoverflow.com**

2. **https://www.github.com**

3. **Java: A Beginner's Guide**