**VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY**

**INTERNATIONAL UNIVERSITY**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**ALGORITHMS AND**

**DATA STRUCTURES**

**IT013IU**

**FINAL REPORT**

**Course by Prof. Vi Chi Thanh**

**TOPIC: NOTEPAD APPLICATION**

**USING DATA STRUCTURES AND ALGORITHMS**

**Student: Nguyen Minh Thuan – ITCSIU22269**

**TABLE OF CONTENTS**

Contents

[**I. INTRODUCTION** 4](#_Toc169868393)

[**1. Abstract** 4](#_Toc169868394)

[**2. Used Techniques, Apps & Tools** 4](#_Toc169868395)

[**3. Attached Materials** 4](#_Toc169868396)

[**II. METHODOLOGY** 5](#_Toc169868397)

[**1. Requirements Analysis** 5](#_Toc169868398)

[**2. System Analysis** 5](#_Toc169868399)

[**2.1. UML Class Diagram** 5](#_Toc169868400)

[**2.2. Used Data Structures/Algorithms** 6](#_Toc169868401)

[**III. CONCLUSION** 9](#_Toc169868402)

[**1. Completed Objectives** 9](#_Toc169868403)

[**2. Future Orientation** 9](#_Toc169868404)

[**IV. REFERENCES** 10](#_Toc169868405)

**LIST OF FIGURES**

Figure 1. UML Class Diagram 5

Figure 2. Search and Highlight function #1 7

Figure 3. Search and Highlight function #2 8

# **I. INTRODUCTION**

## **1. Abstract**

In this Algorithms and Data Structures course project, I have developed a simple text editing application inspired by the Notepad application on Windows that uses data structures and algorithms to improve the application’s functionality and utilization. In this project, my intention is try to implement some data structures and algorithms (DSA) concepts into a actual software development procedure to create a functional application.

With that objective, I have outlined some potential features for my application such as text editing with undo/redo actions, text search and highlight, font adjustments, zoom in/out, and color changes. These features will greatly improve the interaction of the user with the application, making it more intuitive and versatile.

## **2. Used Techniques, Apps & Tools**

* Java IDE: Eclipse & Visual Studio Code
* GitHub’s Version Control System

## **3. Attached Materials**

* Presentation Video: <https://shorturl.at/iiFmz>
* Github Repository: <https://github.com/Celsius06/NotepadJava>. I have utilized Git as the version control system to keep track with the changes to the source code, manage code versions, and make it easy to figure out potential problems for the code if exists.

# **II. METHODOLOGY**

## **1. Requirements Analysis**

During the research phase for the project, I have analyzed the expectations of users and the capabilities of the system for a simple Windows application. As for the primary users of the application, it is necessary to identify them as people who need a versatile text editor. The actions that should be effortlessly executed by the user including create a new document, open the existing document, and saving changes on a document. Additionally, features allowing users to customize font, font size, and text color were considered crucial for enhancing the usability of the application.

To cover the demands of ordinary users, the application refrained from implementing complex features. Instead, emphasis was placed on creating a user-friendly interface with intuitive navigation. Common operations like undo/redo actions, word search, and basic formatting were prioritized as they address the fundamental needs of most users.

## **2. System Analysis**

### **2.1. UML Class Diagram**

A screenshot of a computer

Description automatically generated

Figure 1. UML Class Diagram

### **2.2. Used Data Structures/Algorithms**

**- Queue**

Particularly, in the searchAndHighlight()method of my Function\_View class, the Queue data structure is used to store the positions (indices) within the text where the searched word is found. When a word is found in the text, its position is added to the queue, allowing the program to process and highlight these positions in the order they were found. The code for this method is as follow:

|  |
| --- |
| /\* Name: Nguyen Minh Thuan – ITCSIU22269 Purpose: This method illustrates the usage of Queue by the search and highlight function. \*/    public void **searchAndHighlight**(String word) {          Queue<Integer> positions = new **LinkedList**<>();          String content = gui.textArea.**getText**();          int index = content.**indexOf**(word);          int count = 0; // Counter for occurrences          while (index >= 0) {              positions.**add**(index);              index = content.**indexOf**(word, index + 1);              count++;          }          Highlighter highlighter = gui.textArea.**getHighlighter**();          Highlighter.HighlightPainter painter = new DefaultHighlighter.**DefaultHighlightPainter**(Color.YELLOW);          highlighter.**removeAllHighlights**();          while (!positions.**isEmpty**()) {              int pos = positions.**poll**();              try {                  // Highlight the found word                  highlighter.**addHighlight**(pos, pos + word.**length**(), painter);              } catch (BadLocationException e) {                  e.**printStackTrace**();              }          }          if (count > 0) {              JOptionPane.**showMessageDialog**(null, "Total occurrences of '" + word + "': " + count, "Word Search Result", JOptionPane.INFORMATION\_MESSAGE);          } else {              JOptionPane.**showMessageDialog**(null, "Word not found.", "Word Search Result", JOptionPane.INFORMATION\_MESSAGE);          }      } |

This is a detailed explanation of how the Queue operated in my method above.

**Initialization**:

* At the beginning of the searchAndHighlight()method, a Queue<Integer> called positions is initialized as a LinkedList. This creates an empty queue to store the positions of the searched word.

**Storing Positions**:

* As the program searches for the word within the text, it finds the index (position) of each occurrence of the word using the indexOf method.
* If the word is found (index >= 0), its position is added to the positions queue using the add method. This process continues until all occurrences of the word are found.

**Highlighting Process**:

* After storing all the positions of the searched word, the program proceeds to highlight each occurrence.
* It dequeues (removes) positions from the positions queue using the poll method, which retrieves and removes the head of the queue.
* For each position retrieved, the program uses the highlighter object to add a highlight to the corresponding word in the text area.

**Displaying Search Result**:

* + Finally, the program displays a message dialog indicating the total number of occurrences of the searched word.
  + If no occurrences are found (count == 0), it informs the user that the word was not found in the text.

Why do I use Queue for this feature of my application instead of other Data Structures?

- In my opinion, Queue operates on the FIFO principle, which means elements are processed in the order they are added. This is beneficial for both searching and highlighting, as Queue will process them in the order they appear in the text ensures a logical flow and prevents issues that might arise from processing out-of-order indices, which might be caused from other data structures like Array.

- Furthermore, for my situation here, A Queue provides a straightforward and efficient way to handle the positions, and the time complexity for the operations are also O(1), which will reduce the complexity of the code execution.

Below are two screenshots from my application that illustrate the operation of the searchAndHighlight()method:

A screenshot of a computer

Description automatically generated

Figure 2. Search and Highlight function #1

A screenshot of a computer

Description automatically generated

Figure 3. Search and Highlight function #2

# **III. CONCLUSION**

## **1. Completed Objectives**

In conclusion, I have designed a simple Notepad-inspired text editing application which fulfills the fundamental requirements of the proposed system. The application provides the fundamental abilities to create, open, search, highlight, and save files, which are basic operations related to text editing. Moreover, it has simple styling features such as font type and size, as well as word wrap feature, which improves the overall appearance of the text. The use of undo and redo features also increases the usability of the application since the users are able to reverse actions made. In summary, the completed objectives are the prerequisite for a workable and intuitive text editing environment.

## **2. Future Orientation**

In the future planning for my application, I will try to make more enhancements and features to further improve its functionality. Here are some potential improvements that I envision for future development:

* **Advanced Formatting Options**: Integrate additional formatting options such as text alignment, line spacing, and paragraph indentation to provide users with more control over the appearance of their documents.
* **Customization and Personalization**: Allow users to customize the interface by choosing themes, customizing toolbar layouts, and setting preferences to tailor the application to their individual preferences and workflows.
* **Collaboration and Sharing**: Introduce collaboration features that enable multiple users to collaborate on documents in real-time, as well as sharing options that facilitate easy sharing of documents via email, cloud storage services, or collaboration platforms.
* **Integration with Cloud Services**: Integrate with popular cloud storage services such as Google Drive, Dropbox, or OneDrive to enable seamless synchronization of documents across devices and easy access to files from anywhere.

# **IV. REFERENCES**

[1]. Java GUI: Full Course (2021). Retrieved from <https://www.youtube.com/watch?v=Kmgo00avvEw>

[2]. Data Structures and Algorithms with Visualizations – Full Course (Java). Retrieved from <https://www.youtube.com/watch?v=2ZLl8GAk1X4&t=114919s>