ÇANKAYA UNIVERSITY
SOFTWARE ENGINEERING DEPARTMENT

Name Surname	**Zekiye İpek ÖZMEN**
Identity Number	**202028004**
Course	**Seng 383**
Experiment	**Simple Search Engine Using Linked Lists**
E-mail	**c2028004@student.cankaya.edu.tr**

Table of Contents

1.Questions

PROBLEM STATEMENT

- Describe the goals of the programming assignment
- What are the normal inputs to the program?
- What output should the program create?
- What error handling was required?

DESIGN

- Describe the design decisions you made.
- What data structures did you use?
- What algorithms did you use?
- What were pros/cons of choices above?

IMPLEMENTATION DETAILS

- Describe your implementation process. (Flowchart)
- What sample code did you start with?
- How did you extend or adapt this code?
- What was your development timeline?

TESTING NOTES

- Describe how you tested your program.
- What were the normal inputs you used?
- What were the special cases you tested?
- Did everything work as expected?
- Include sample input/output from your program.

COMMENTS

- Describe the overall result of the assignment.
- Was the programming project a success?
- What would you do same or differently next time?

2. Introduction

The purpose of this report is to provide an overview of the design considerations and architecture for the "Simple Search Engine Using Linked Lists" project. This project aims to develop a basic search engine that leverages linked list data structures and efficient algorithms to enable users to search, load, and remove documents based on query strings.

3.Answers

Problem Statement

- The primary goal of this programming assignment is to develop a simple search engine using a linked list data structure. The system should be able to load information from files, perform searches based on query strings, and remove unwanted documents.

- Normal inputs include command files specifying operations like clearing the list, loading information from a file, searching, and removing documents.

- The program should generate an output file containing search results based on user queries.
- Error handling is needed to ensure the program can handle cases where a document specified in a "remove" command is not found in the information file. Memory leaks should be prevented when clearing the list.

Design

- The design involves using a linked list data structure to store documents and their associated data. The linked list should be organized in alphabetical order, making it efficient for searching. Binary tree search algorithms could be used for faster lookups.

- For the primary data structure a linked list is going to be used. Also a binary search tree could be used to store the list of terms in alphabetical order. This tree can be used for efficient searching.

- Algorithms to use are simply,to clear data, simply using a loop. When loading data from a file, use file operations. For searching, split the query into wanted and unwanted terms, and maintain an index for quick retrieval. To remove data, traverse the list and adjust pointers. To handle sparse matrices, use a linked list of linked lists to save memory. Keep columns in alphabetical order by sorting when adding data. For efficient searching, using a binary search tree.

- Linked List pro’s are efficient for adding and removing documents, con’s are less efficient for searching documents based on the query. Sparse Matrix pro’s are efficient memory usage for storing document-term relationships and con’s are can be inefficient for searching. Binary Search Tree pro’s are efficient for searching terms in alphabetical order and con’s are additional memory usage for tree structure.

4. Problem Statement

The project aims to address the following challenges:

Implement a simple search engine using linked lists.
Develop features like clearing the list, loading information from files, searching documents, and removing documents.
Optimize search operations for acceptable execution time.
Maintain alphabetical order of columns for efficient searching.
Ensure an object-oriented design for modularity and maintainability.

5. Design Overview

5.1 Data Structures

The project will utilize the following data structures:

Linked List: To manage documents and data.
Linked List of Linked Lists: To represent a sparse matrix efficiently.
Binary Search Tree (BST): To maintain alphabetical order for terms.

5.2 Algorithms

The key algorithms used in the project:

Clear List: Iterative algorithm to clear entries and deallocate memory.
Load Info: Standard file I/O operations for reading and parsing documents.
Search: Algorithm for parsing and processing query strings.
Remove: Linked list traversal for removing unwanted documents.
Sparse Matrix: Algorithms for linked list management to optimize memory usage.
Alphabetical Order: Sorting algorithms to maintain alphabetical order for terms.
Binary Tree Search: BST for quick term retrieval.

5.3 Object-Oriented Design

The project will adhere to object-oriented design principles:

Implement classes for Documents, Linked Lists, Terms, and the Search Engine itself.
Utilize encapsulation, inheritance, and polymorphism for modular and maintainable code.

6. Functional Requirements

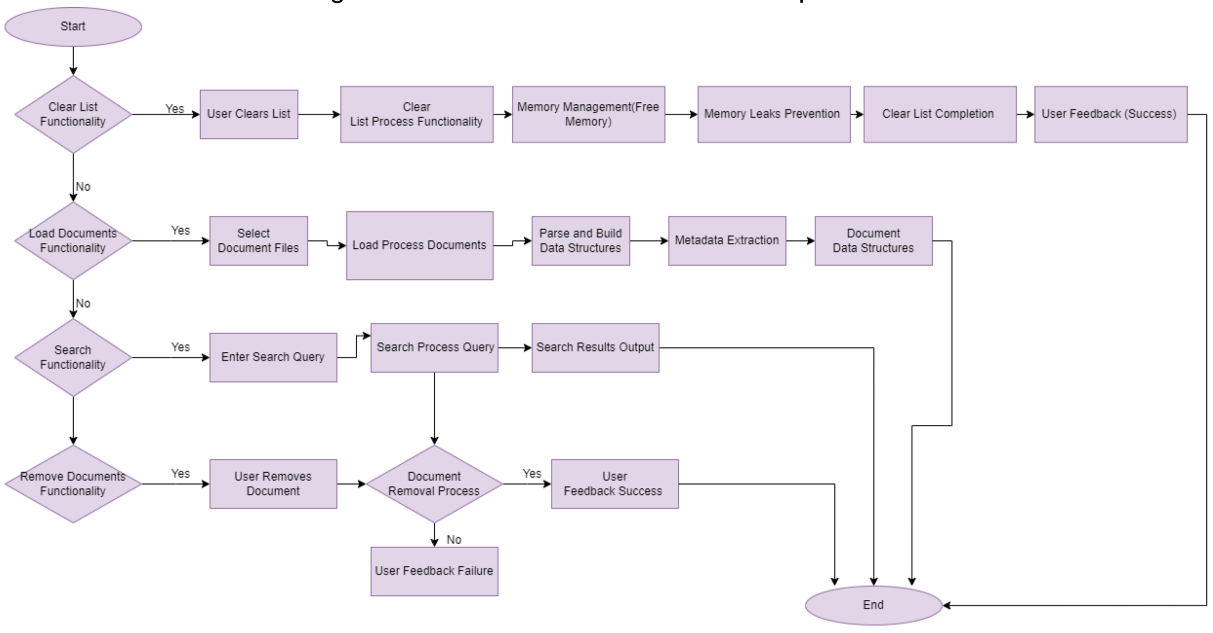
1. Clear List Functionality: The system should allow users to clear all entries in the search engine. Clearing the list should also free up memory and prevent memory leaks.

2. Load Documents: Users should be able to load documents from files into the search engine. The system must parse the input files and build the necessary data structures. Documents should include metadata (e.g., title, author, date).

3. Search Functionality: Users should be able to perform searches using query strings. Query strings should consist of desired and undesired words. The system should list documents that match the query. The logical operator "and" should be implicit; users don't need to specify it explicitly. The results should be appended to an output file.

4. Remove Documents: The system should allow users to remove unwanted documents. Users can specify the document to be removed. If the document does not exist in the information file, the system may skip the command.

Diagram 1: Flowchart of the Functional Requirements



7. Implementation Details

- I began with a basic Java code structure that included classes for a LinkedList. The code is designed to build a simple search engine that can load, search, remove, and clear documents from a linked list structure. It reads commands from a text file, processes them, and stores the results in another text file.

- The SearchEngine class is the entry point of the application. It reads a text file containing commands (e.g., load, search, clear, remove) and processes them one by one. It maintains a linked list of documents and a map for quick access to documents by their title. Search results are stored in a list, and execution time is recorded. The Document class represents a document with attributes like title, author, and date. This class is used to create and manage document objects. The LinkedList class represents a linked list data structure that holds document objects. It supports operations like inserting, removing, searching, clearing, and returning the size of the list. The LinkedListNode class is a simple node for the linked list, holding a document and a reference to the next node.

- I extended and adapted the code to read commands from a text file (commands.txt) and execute corresponding actions. Load documents from another text file specified in the "load" command and store them in the linked list and document map. Search for documents based on a query. Remove documents based on their titles. Measure and print the execution time. Save search results to a text file ("results.txt").
- I started by creating a document for the project, designing it and creating a flowchart for it. After that I started implementing the project and created test cases.

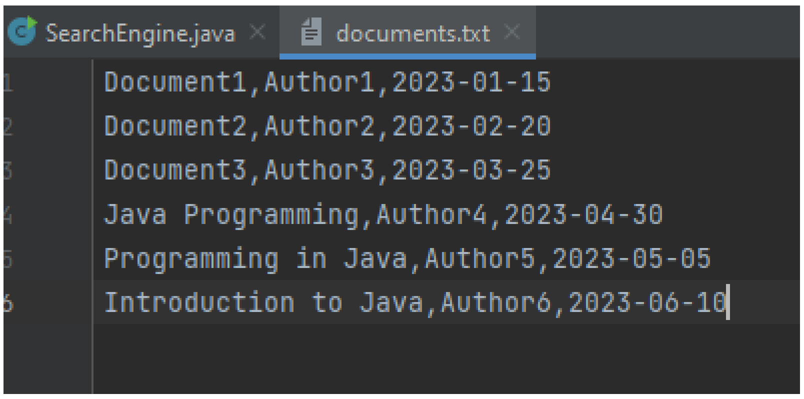
8. Testing Notes

- I use JUnit to create unit tests for the SearchEngine class, ensuring that different functionalities were tested independently. Each test case is designed to test a specific feature or command, such as loading documents, searching, removing documents, and clearing the list. For each test case, I created a temporary test input file with specific content and executed the corresponding commands to test the functionality.

- Normal inputs include typical use cases for the search engine, such as loading documents, searching for documents, and removing documents. These tests check if the basic functionality of the search engine works as expected.

- For example, in the testSearchDocuments test case, I tested searching for documents with a common prefix ("doc") to see if the search function can return multiple results. In the testRemoveDocument test case, I tested removing a document to check if the removal functionality works as expected.

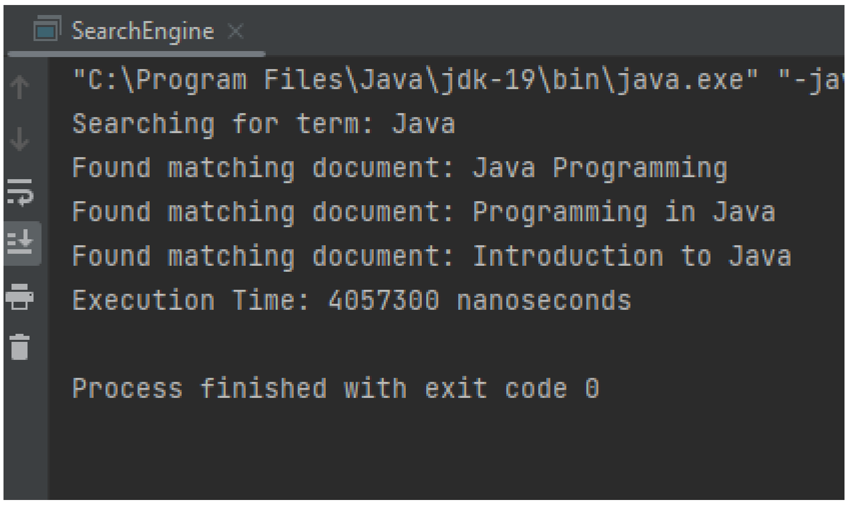
- Sample Input:



```
Document1,Author1,2023-01-15
Document2,Author2,2023-02-20
Document3,Author3,2023-03-25
Java Programming,Author4,2023-04-30
Programming in Java,Author5,2023-05-05
Introduction to Java,Author6,2023-06-10
```

Picture 1 Sample Input from document.txt

- Output:



```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-ja
Searching for term: Java
Found matching document: Java Programming
Found matching document: Programming in Java
Found matching document: Introduction to Java
Execution Time: 4057300 nanoseconds

Process finished with exit code 0
```

Picture 2 Output

9. Conclusion

This design report outlines the architecture, design principles, implementation details and testing notes for the "Simple Search Engine Using Linked Lists" project. It focuses on efficient data structures, algorithms, object-oriented design, user interface, testing, and project management. The successful implementation of this design will result in a functional and user-friendly search engine that meets the project's objectives.