Stage 1

Description

Let's implement the simplest search engine possible ever. It should search for a specific word in a multi-word input line.

The input line contains several words separated by a space. The words are numbered according to their order, with the first word having index 1. Consider that all the words in the line are unique, so there can be no repetition.

Objectives

Write a simple program that reads two lines: a line of words and a line containing the search word. The program must search in the first line for a word specified in the second one. The program should output the index of the specified word. If there is no such word in the first line, the program should print Not Found. Please remember that indexes start from 1!

You should output exactly one line.

Examples

The lines that start with > represent the user input. Note that these symbols are not part of the input.

Example 1:

```
Unset
> first second third fourth

> third

3
```

Example 2:

```
Unset
> cat dog and mouse
```

> elephant

Not found

Solution

```
package search;

import java.util.Scanner;


public class Main {

   public static void main(String args[]){

        Scanner scan = new Scanner(System.in);

        String string = scan.nextLine();

        String string2 = scan.next();

        String[] strings =string.split(" ");

        int count=1;

        boolean flag=false;

        for(String tester: strings){


            if(string2.equals(tester)){

                System.out.println(count);

                flag = true;

                break;

            }

            count++;
```

```
        }
        if(flag==false){
            System.out.println("Not Found");
        }
    }


    }
```

Stage 2

Description

Now, let's make our search a little more complex. Let's write a program that performs multiple searches in multiple text lines.

Objectives

Write a program that reads text lines from the standard input and processes single-word queries. The program must output all lines that contain the string from the query. For this stage, this should include the case where the query string appears as a substring of one of the text lines. For example, the query "bc" should be found in a line containing "abcd".

You may choose what the text represents in your project. For example, each line may describe:

a person represented by the first name, the last name, and optionally an email;

an address of a building represented by the country, city, state, street, and zip code;

a book represented by its ISBN, title, author/authors, publisher, and so on.

You can use any of these options or come up with your own, because your search algorithm should work regardless of what the text actually represents.

Here is an example of a line. It contains three items: first name, last name, and this person's email.

```
Unset
Elsa Sanders elsa@gmail.com
```

In this example, all items are separated by spaces.

The search should ignore letter cases and all the extra spaces.

Firstly, the user should input a number N, which is a number of lines with data they are going to enter next. Then the user enters N lines with data. After that, the user enters a number M, which is a number of search queries. And after each query, the program should print the information it managed to find among the data. You can see this searching process in the example below.

Example

In the following example, we use different names and e-mails as an example of the dataset. The lines that start with > represent the user input. Note that these symbols are not part of the input.

```
Unset
Enter the number of people:

> 6

Enter all people:

> Dwight Joseph djo@gmail.com

> Rene Webb webb@gmail.com

> Katie Jacobs

> Erick Harrington harrington@gmail.com

> Myrtle Medina

> Erick Burgess
```

Enter the number of search queries:

> 3

Enter data to search people:

> ERICK

Found people:

Erick Harrington harrington@gmail.com

Erick Burgess

Enter data to search people:

> unknown

No matching people found.

Enter data to search people:

> WEBB@gmail.com

Found people:

Rene Webb webb@gmail.com

Solution:

```java
package search;

import java.util.Scanner;


public class Main {

   public static void main(String args[]) {


        Scanner scan = new Scanner(System.in);

        boolean flag = false;

        System.out.println("Enter number of people:");

        String hi = scan.nextLine();

        int num = Integer.parseInt(hi);

        //String s = scan.nextLine();

        //System.out.println(s);

        System.out.println("Enter all people:");

        String[] store = new String[num];

        for (int i = 0; i < num; i++) {


            store[i] = scan.nextLine();

            //System.out.println(store[i]);

            //System.out.println(i);

        }

        System.out.println("Enter the number of search queries");

        String hello = scan.nextLine();

        int mum = Integer.parseInt(hello);

        for (int j = 0; j < mum; j++) {

            System.out.println("Enter data to search people:");
```

```
            String input = scan.next();
            for (String vars : store) {
                if (vars.toLowerCase().contains(input.toLowerCase())) {
                    System.out.println(vars);
                    flag = true;
                }

            }
            if (!flag) {
                System.out.println("No matching people found.");
            }

        }
    }
}
```

Stage 3

Description

Let's modify the previously written search program an add a user menu for a better user experience.

Objectives

In this stage, you need to create a menu. The menu should display the following options:

```
Unset
1. Search information.

2. Print all data.
```

```
0. Exit.
```

The user must select a menu item and then enter data if necessary. Your program must not stop until the corresponding option (the exit option) is chosen.

Decompose the program into separate methods to make it easy to understand and to further develop or edit.

Example

In the example below, we use people as a dataset example. The lines that start with > represent the user input. Note that these symbols are not part of the input.

```
Unset
Enter the number of people:

> 6

Enter all people:

> Dwight Joseph djo@gmail.com

> Rene Webb webb@gmail.com

> Katie Jacobs

> Erick Harrington harrington@gmail.com

> Myrtle Medina

> Erick Burgess


=== Menu ===

1. Find a person

2. Print all people
```

0. Exit

> 3

Incorrect option! Try again.

=== Menu ===

1. Find a person

2. Print all people

0. Exit

> 1

Enter a name or email to search all suitable people.

> KATIE

Katie Jacobs

=== Menu ===

1. Find a person

2. Print all people

0. Exit

> 2

=== List of people ===

Dwight Joseph djo@gmail.com

Rene Webb webb@gmail.com

Katie Jacobs

Erick Harrington harrington@gmail.com

Myrtle Medina

Erick Burgess

=== Menu ===

1. Find a person

2. Print all people

0. Exit

> 0

Bye!

Solution:

```
package search;

import java.util.Scanner;

public class Main {

   private static String[] people;

   private static Scanner scanner = new Scanner(System.in);

   public static void main(String[] args) {
```

```
        initializeData();
        showMenu();
    }
    private static void initializeData() {
        System.out.println("Enter the number of people:");
        int n = Integer.parseInt(scanner.nextLine());
        //System.out.println(n);
        people = new String[n];
        System.out.println("Enter all people:");
        for (int i = 0; i < n; i++) {
            people[i] = scanner.nextLine().trim();
        }
    }
    private static void showMenu() {
        int option;
        do {
            System.out.println("\n=== Menu ===");
            System.out.println("1. Find a person");
            System.out.println("2. Print all people");
            System.out.println("0. Exit");
            System.out.print("> ");
            option = Integer.parseInt(scanner.nextLine());
            switch (option) {
                case 1:
                    findPerson();
                    break;
```

```
                case 2:
                    printAllPeople();
                    break;
                case 0:
                    System.out.println("Bye!");
                    break;
                default:
                    System.out.println("Incorrect option! Try again.");
            }
        } while (option != 0);
    }
    private static void findPerson() {
            System.out.println("Enter a name or email to search all suitable
people.");
        String searchWord = scanner.nextLine().trim().toLowerCase();
        //System.out.println(searchWord);
        boolean found = false;
        for (String person : people) {
            if (person.toLowerCase().contains(searchWord)) {
                System.out.println(person);
                found = true;
            }
        }
        if (!found) {
            System.out.println("No matching people found.");
        }
    }
```

```
    private static void printAllPeople() {
        System.out.println("\n=== List of people ===");
        for (String person : people) {
            System.out.println(person);
        }
    }
}
```

Stage 4

Description

Now, let's further modify the program and teach it to read the input data from a file.

Objectives

In this stage, your program should read data from a text file instead of the standard input. The file structure will depend on your text data type (personal information, address information, book information, and so on). See an example below:

```
Unset
Dwight Joseph djo@gmail.com

Rene Webb webb@gmail.com

Katie Jacobs

Erick Harrington harrington@gmail.com

Myrtle Medina

Erick Burgess
```

The program must process the command line argument --data followed by the name of the file with the data, for example, --data text.txt.

Note that the file should not include the total number of lines. All lines must be read only once, at the start of your program.

Output example

The lines that start with > represent the user input. Note that these symbols are not part of the input.

```
Unset
=== Menu ===

1. Find a person

2. Print all people

0. Exit

> 1


Enter a name or email to search all suitable people.

> WEBB@gmail.com

Rene Webb webb@gmail.com


=== Menu ===

1. Find a person

2. Print all people

0. Exit

> 0


Bye!
```

Solution:

```
package search;

import java.io.FileReader;

import java.io.IOException;

import java.util.Scanner;

public class Main {

   private static String[] people;

   private static int count = 0;

   private static Scanner scanner = new Scanner(System.in);

   public static void main(String[] args) {

       if (args.length < 2 || !args[0].equals("--data")) {

           //System.out.println(args[0]);

           System.out.println("Usage: --data filename");

           return;

       }

       String filename = args[1];

       initializeData(filename);

       showMenu();

   }

   private static void initializeData(String filename) {

       people = new String[100];

       try (FileReader fr = new FileReader(filename);

            Scanner fileScanner = new Scanner(fr)) {

           while (fileScanner.hasNextLine()) {

               if (count >= people.length) {

                   String[] newArray = new String[people.length * 2];
```

```
                for (int i = 0; i < people.length; i++) {
                    newArray[i] = people[i];
                    //i++;
                }
                people = newArray;
            }
            people[count++] = fileScanner.nextLine().trim();
        }
    } catch (IOException e) {
        System.out.println("Error reading the file: " + e.getMessage());
    }
}
private static void showMenu() {
    int option;
    //while(option != 0);
    do {
        System.out.println("\n=== Menu ===");
        System.out.println("1. Find a person");
        System.out.println("2. Print all people");
        System.out.println("0. Exit");
        System.out.print("> ");
        option = Integer.parseInt(scanner.nextLine());
        switch (option) {
            case 1:
                findPerson();
                break;
```

```
                case 2:
                    printAllPeople();
                    break;
                case 0:
                    System.out.println("Bye!");
                    break;
                default:
                    System.out.println("Incorrect option! Try again.");
            }
        } while (option != 0);
    }
    private static void findPerson() {
            System.out.println("Enter a name or email to search all suitable
people.");
        String searchWord = scanner.nextLine().trim().toLowerCase();
        boolean found = false;
        for (int i = 0; i < count; i++) {
            if (people[i].toLowerCase().contains(searchWord)) {
                System.out.println(people[i]);
                found = true;
            }
        }
        if (!found) {
            System.out.println("No matching people found.");
        }
    }
    private static void printAllPeople() {
```

```
        System.out.println("\n=== List of people ===");
        for (int i = 0; i < count; i++) {
            System.out.println(people[i]);
        }
    }
}
```

Stage 5

Description

Now your program can successfully search for all matching lines, and the search is case- and space-insensitive. There is one problem though: you need to check each line to find out whether it contains the query string.

To optimize your program, you can use a data structure called an Inverted Index. It maps each word to all positions/lines/documents in which the word occurs. As a result, when we receive a query, we can immediately find the answer without any comparisons.

Objectives

In this stage, build an inverted index at the start of the program and then use the index for searching operations. You can implement it using the Map class. It connects an item with a list (or set) of indexes belonging to the lines that contain the item.

Suppose you have the following list of lines:

```
Unset
0: Katie Jacobs

1: Erick Harrington harrington@gmail.com

2: Myrtle Medina

3: Erick Burgess
```

For these lines, the inverted index will look like this:

```
Unset
Katie -> [0]

Jacobs -> [0]

Erick -> [1, 3]

Harrington -> [1]

harrington@gmail.com -> [1]

Myrtle -> [2]

Medina -> [2]

Burgess -> [3]
```

The order of pairs is not important. If you are searching for Erick, you can immediately get the target fields using this mapping.

Note that the Inverted Index is not intended to search for parts of a word, it is only used to search for full words.

Example

The lines that start with > represent the user input. Note that these symbols are not part of the input.

```
Unset
=== Menu ===

1. Find a person

2. Print all people

0. Exit

> 1
```

Enter a name or email to search all suitable people.

> ERICK

2 persons found:

Erick Harrington harrington@gmail.com

Erick Burgess

=== Menu ===

1. Find a person

2. Print all people

0. Exit

> 1

Enter a name or email to search all suitable people.

> ROY@gmail.com

1 persons found:

Richard Roy roy@gmail.com

=== Menu ===

1. Find a person

2. Print all people

0. Exit

> 1

Enter a name or email to search all suitable people.

> john

No matching people found.

=== Menu ===

1. Find a person

2. Print all people

0. Exit

> 0

Bye!

Solution:

```
import java.io.FileReader;
import java.io.IOException;
import java.util.*;
public class Main {
   private static List<String> people;
   private static Map<String, Set<Integer>> invertedIndex;
   private static Scanner scanner = new Scanner(System.in);
```

```
public static void main(String[] args) {
    if (args.length < 2 || !args[0].equals("--data")) {
        System.out.println("Usage: --data filename");
        return;
    }
    String filename = args[1];
    //System.out.println(filename);
    initializeData(filename);
    buildInvertedIndex();
    showMenu();
}
private static void initializeData(String filename) {
    people = new ArrayList<>();
    try (FileReader fr = new FileReader(filename);
         Scanner fileScanner = new Scanner(fr)) {
        while (fileScanner.hasNextLine()) {
            //System.out.println(fileScanner.nextLine());
            people.add(fileScanner.nextLine().trim());
        }
    } catch (IOException e) {
        System.out.println("Error reading the file: " + e.getMessage());
    }
}
private static void buildInvertedIndex() {
    invertedIndex = new HashMap<>();
    for (int i = 0; i < people.size(); i++) {
```

```
            String[] words = people.get(i).split("\\s+");
            for (String word : words) {
                String lowerCaseWord = word.toLowerCase();
                invertedIndex.putIfAbsent(lowerCaseWord, new HashSet<>());
                invertedIndex.get(lowerCaseWord).add(i);
            }
        }
    }
    private static void showMenu() {
        int option;
        do {
            System.out.println("\n=== Menu ===");
            System.out.println("1. Find a person");
            System.out.println("2. Print all people");
            System.out.println("0. Exit");
            System.out.print("> ");
            option = Integer.parseInt(scanner.nextLine());
            switch (option) {
                case 1:
                    findPerson();
                    break;
                case 2:
                    printAllPeople();
                    break;
                case 0:
                    System.out.println("Bye!");
```

```
                    break;
                default:
                    System.out.println("Incorrect option! Try again.");
            }
        } while (option != 0);
    }
    private static void findPerson() {
            System.out.println("Enter a name or email to search all suitable
people.");
        String searchWord = scanner.nextLine().trim().toLowerCase();
        Set<Integer> foundIndexes = invertedIndex.get(searchWord);
        if (foundIndexes != null && !foundIndexes.isEmpty()) {
            System.out.println(foundIndexes.size() + " persons found:");
            for (int index : foundIndexes) {
                System.out.println(people.get(index));
            }
        } else {
            System.out.println("No matching people found.");
        }
    }
    private static void printAllPeople() {
        System.out.println("\n=== List of people ===");
        for (String person : people) {
            System.out.println(person);
        }
    }
}
```

Stage 6

Description

Now let's Improve your search engine to make it support complex queries containing word sequences and use several strategies that determine how to match data.

Objectives

In this stage, your program should be able to use such searching strategies as ALL, ANY, and NONE.

Take, for example, these six sample lines:

```
Unset
Dwight Joseph djo@gmail.com

Rene Webb webb@gmail.com

Katie Jacobs

Erick Harrington harrington@gmail.com

Myrtle Medina

Erick Burgess
```

If the strategy is ALL, the program should print lines containing all the words from the query.

Query:

```
Unset
Harrington Erick
```

Result:

```
Unset
Erick Harrington harrington@gmail.com
```

If the strategy is ANY, the program should print the lines containing at least one word from the query.

Query:

```
Unset
Erick Dwight webb@gmail.com
```

Result:

```
Unset
Erick Harrington harrington@gmail.com

Erick Burgess

Dwight Joseph djo@gmail.com

Rene Webb webb@gmail.com
```

If the strategy is NONE, the program should print lines that do not contain words from the query at all:

Query:

```
Unset
djo@gmail.com ERICK
```

Result:

```
Unset
Katie Jacobs
```

Myrtle Medina

Rene Webb webb@gmail.com

All listed operations are implemented in the inverted index. The results should not contain duplicates.

Do not forget to use methods to decompose your program.

Example

The lines that start with > represent the user input. Note that these symbols are not part of the input.

```
=== Menu ===

1. Find a person

2. Print all persons

0. Exit

> 1


Select a matching strategy: ALL, ANY, NONE

> ANY


Enter a name or email to search all suitable people.

> Katie Erick QQQ


3 persons found:
```

Katie Jacobs

Erick Harrington harrington@gmail.com

Erick Burgess

Solution:

```
package search;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.*;


public class Main {

   private static List<String> people = new ArrayList<>();

   private static Map<String, Set<Integer>> invertedIndex = new HashMap<>();

   private static Scanner scanner = new Scanner(System.in);


   public static void main(String[] args) {

       if (args.length < 2 || !args[0].equals("--data")) {

           System.out.println("Usage: --data filename");

           return;

       }

       initializeData(args[1]);

       showMenu();

   }


   private static void initializeData(String filename) {
```

```
    try (Scanner fileScanner = new Scanner(new File(filename))) {
        int lineIndex = 0;
        while (fileScanner.hasNextLine()) {
            String line = fileScanner.nextLine().trim();
            if (!line.isEmpty()) {
                people.add(line);
                for (String word : line.toLowerCase().split("\\s+")) {
                    invertedIndex.putIfAbsent(word, new HashSet<>());
                    invertedIndex.get(word).add(lineIndex);
                }
                lineIndex++;
            }
        }
    } catch (FileNotFoundException e) {
        System.out.println("Error reading the file: " + e.getMessage());
    }
}

private static void showMenu() {
    while (true) {
        System.out.println("\n=== Menu ===");
        System.out.println("1. Find a person");
        System.out.println("2. Print all people");
        System.out.println("0. Exit");
        System.out.print("> ");
        int option = Integer.parseInt(scanner.nextLine());
```

```
            switch (option) {
                case 1: findPerson(); break;
                case 2: printAllPeople(); break;
                case 0: System.out.println("Bye!"); return;
                default: System.out.println("Incorrect option! Try again.");
            }
        }
    }

    private static void findPerson() {
        System.out.print("Select a matching strategy: ALL, ANY, NONE\n> ");
        String strategyInput = scanner.nextLine().toUpperCase();
            System.out.print("Enter a name or email to search all suitable
people.\n> ");
        String query = scanner.nextLine().toLowerCase();

        Set<Integer> results = new HashSet<>();
        String[] words = query.split("\\s+");

        if ("ALL".equals(strategyInput)) {
                            results.addAll(invertedIndex.getOrDefault(words[0],
Collections.emptySet()));
            for (String word : words) {
                             results.retainAll(invertedIndex.getOrDefault(word,
Collections.emptySet()));
            }
        } else if ("ANY".equals(strategyInput)) {
```

```
            for (String word : words) {
                                    results.addAll(invertedIndex.getOrDefault(word,
Collections.emptySet()));
            }
        } else if ("NONE".equals(strategyInput)) {
            Set<Integer> excluded = new HashSet<>();
            for (String word : words) {
                                   excluded.addAll(invertedIndex.getOrDefault(word,
Collections.emptySet()));
            }
            for (int i = 0; i < people.size(); i++) {
                if (!excluded.contains(i)) results.add(i);
            }
        } else {
              System.out.println("Invalid strategy! Please choose ALL, ANY, or
NONE.");
            return;
        }

        System.out.println(results.size() + " person(s) found:");
        for (Integer index : results) {
            System.out.println(people.get(index));
        }
    }

    private static void printAllPeople() {
        System.out.println("\n=== List of people ===");
```

```
        for (String person : people) {
            System.out.println(person);
        }
    }
}
```