Simple Search Engine (Java). Stage 1/6

Project description ↓

String theory

Report a typo

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.

Simple Search Engine (Java). Stage 2/6

Project description ↓

Expand the search

Report a typo

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.

```
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

Simple Search Engine (Java). Stage 3/6

Project description ↓

User menu

Report a typo

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:

```
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.

Simple Search Engine (Java). Stage 4/6

Project description ↓

X-files

Report a typo

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:

```
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.

Simple Search Engine (Java). Stage 5/6

Project description ↓

Inverted Index search

Report a typo

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.

Simple Search Engine (Java). Stage 6/6

Project description ↓

Search strategies

Report a typo

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:

```
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:

```
Harrington Erick
```

Result:

```
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:

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

Result:

```
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: