findNeedles() API Reference Document

Overview

This document provides an overview of the API function `findNeedles()` written in Java. The function prints the number of times each element of the string array `needles` appears in the string `haystack`.

Audience analysis

This document is intended for developers familiar with Java programming who want to gain an understanding of the `findNeedles()` function.

Before you begin

To use this code, declare class `NeedlesInHaystack` and use function `findNeedles()` as a part of this class.

For example:

```
public class NeedlesInHaystack {
    public static void main(String[] args) {
        // Sample call
        findNeedles("Mode of transport are car bus train", new
String[]{"car", "bus", "bike", "car", "plane"});
    }
}
```

Calling the API method

This method uses two inputs: string `haystack` and a string array `needles`

The parameters used in the `findNeedles()` are tabulated below:

Parameters	Datatype	Description
`haystack`	`String`	A sentence that acts as the text to search for `needles`.
`needles`	`String []`	An array of words to search for within the `haystack`.

```
String haystack = "Mode of transport are car bus train and
bike";
String[] needles = {"car", "bus", "bike", "train"};
```

Decoding the code

Line No	Code	Remarks
1.	`public static void findNeedles(String haystack, String[] needles)`	Declares the parameters `haystack (string)` and `needles [string array]` in the `findNeedles()` function.
2.	`if (needles.length > 5)`	Verifies that the length of the `needles` array does not exceed five.
3.	`System.err.println("Too many words!");`	Sends an error message if the length of the `needles` array exceeds five.
4.	`} else {`	If there are 5 or fewer needles, the code proceeds to the next block.
5.	`int[] countArray = new int[needles.length];`	Declares a new variable `countArray` to store the count of occurrences for each needle in the haystack.
6.	`for (int i = 0; i < needles.length; i++) {`	Loops through each element in the `needles` array.
7.	`String[] words = haystack.split("[ \"\'\t\n\b\f\r]", 0);`	Splits the `haystack` string into words using the following delimiters: " "– space `\'` – apostrophe `\t` – tab `\n` – newline `\b` – backspace `\f` – form feed `\r` – carriage return
8.	`for (int j = 0; j < words.length; j++) {`	Loops through each word in the `words` array.
9.	`if (words[j].compareTo(needles[i]) == 0) {`	Compares the current word in the `haystack` string with the current needle. If a match is found, returns 0.
10.	`countArray[i]++;`	If found, increments the count for the specific word being searched for in the `haystack` string.

Line No	Code	Remarks
11.	`}`	Closes the `if` statement that checks for a match between the current word and needle.
12.	`}`	Closes the inner `for` loop that iterates through each word in the `haystack` string.
13.	`}`	Closes the outer `for` loop that iterates through the words in `needles` array.
14.	`for (int j = 0; j < needles.length; j++) {`	Loops through the `needles` array again to count the occurrences of each needle.
15.	`System.out.println(needles[j] + ": " + countArray[j]);`	Prints the current needle along with its corresponding count from `countArray`.
16.	`}`	Terminates the `for` loop that prints the results for each needle.
17.	`}`	Closes the `else` block that handles the case when the length of the `needles` array length is five or fewer.
18.	`}`	Terminates the `findNeedles()` method.

Use Cases

Case 1:

```
String haystack = "Mode of transport are car bus train and
bike";
String[] needles = {"car", "bus", "bike", "train"};
```

Output:

```
car: 1
bus: 1
bike: 1
train: 1
```

Case 2: (Without String Normalization)

```
String haystack = "Mode of transport are car, bus, train and
bike";
String[] needles = {"car", "bus", "bike", "train"};
```

Output:

```
car: 0
bus: 0
bike: 1
train: 1
```

Case 3: With String Normalization

```
String haystack = "Mode of transport are car, bus, train and
bike";
String[] needles = {"car", "bus", "bike", "Train"};
```

Output:

```
car: 1
bus: 1
train: 1
bike: 1
```

Questions for the programmer

1. Why did you restrict the number of needles to five?
2. Is punctuation in the `haystack` affects the needle matching?
   For example:

   ```
   haystack = "A quick brown fox jumped over the lazy dog.";
   needles = {"quick", "brown", "dog"};
   ```

3. What was the reasoning behind using the `spilt` function with `“[ \"\'\t\n\b\f\r]"`?
4. `Why` use `System.err,printIn()` instead of `System.out.println`() for error message?

Improvements and Suggestions

- **Interactive Output**

Prompt the user to input `haystack` and `needles` at runtime. Ensure the user cannot proceed until the data is inputted.

```
Scanner scanner = new Scanner(System.in);

System.out.print("Enter the haystack: ");
String haystack = scanner.nextLine().trim();
while (haystack.isEmpty()) {
    System.out.println("Haystack cannot be empty. Please enter
again.");
    haystack = scanner.nextLine().trim();
}

System.out.print("Enter the number of needles: ");
int numNeedles = Integer.parseInt(scanner.nextLine());

String[] needles = new String[numNeedles];
for (int i = 0; i < numNeedles; i++) {
    System.out.print("Enter needle " + (i+1) + ": ");
    needles[i] = scanner.nextLine().trim();
}

findNeedles(haystack, needles);
scanner.close();
```

After collecting all the inputs, close the `scanner` object to free the resources.

```
     scanner.close();
```

- **Invalid Input**

In case the user fails to enter any string, re-prompt them for the input.

```
if (haystack.isEmpty() || needles.length == 0) {
    System.out.println("String cannot be empty.");
    return;
}
```

- **Optimizing Performance**

If the number of elements in `needles` exceeds five, the program should exit immediately and print a message such as "`Limit the number of needles to five or fewer.`"

```
 if (needles.length > 5) {

    System.out.println("Limit the number of needles elements
to five or fewer.");

    System.exit(0);

}
```

- **String Normalization**

To ensure case-insensitive comparison, normalize both the `haystack` and `needles` to lowercase.

```
haystack = haystack.toLowerCase().replaceAll("[^a-zA-Z0-9\\s]", "");

String[] needles =
Arrays.stream(needles).map(String::toLowerCase).toArray(String[]::ne
w);
```

- **Rewriting the error message**

The message “`too many words`” can be rephrased to be more constructive.

For example:

```
System.out.println("Limit the number of needles to five or fewer.");
```

- **Code Improvement (Performance Optimization)**

The current code loops through each word in the `haystack` multiple times for each `needle`, which affects the code performance. To improve performance, use a `HashMap` to store the occurrences of words in the haystack.

```
Map<String, Integer> wordCountMap = new HashMap<>();
String[] words = haystack.split("\\s+");

for (String word : words) {
    wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1);
}

for (String needle : needles) {
    int count = wordCountMap.getOrDefault(needle, 0);
    System.out.println(needle + ": " + count);
}}
```

```
for (int i = 0; i < needles.length; i++) {
    int count = wordCountMap.getOrDefault(needles[i], 0);
    System.out.println(needles[i] + ": " + count);
}
```

- **Re-arranging and splitting the string**

To increase efficiency, replace the split `"[\"'\\t\\n\\b\\f\\r]"` with `"\\s+"` to split by one or more whitespace characters. Execute the spilt function before the first `for` loop.

```
        int[] countArray = new int[needles.length];

        String[] words = haystack.split("[\\s+]", 0);

        for (int i = 0; i < needles.length; i++) {

            for (int j = 0; j < words.length; j++) {

                if (words[j].compareTo(needles[i]) == 0) {

                    countArray[i]++;

                }

            }

        }
```