Reviewing a Java code snippet

Souvik Sarkar · 2020-11-09

1 API reference

For this section, I assumed that the method in the code sample is part of a class `NeedlesInHaystack`.

1.1 Class: **`NeedlesInHaystack`**

This class provides functionality to search for multiple needle strings within a single haystack string and outputs the frequency of each needle's occurrence.

*1.1.1 Method: **`findNeedles`***

- Type: `public static`
- Parameters:
  - `String haystack`: The text within which to search for the needles.
  - `String[] needles`: An array of strings representing the words (needles) to be counted in the haystack.
- Returns: `void` (prints results to the console)
- Usage: This method prints the number of times each string in the `needles` array appears in the `haystack` string.

1.1.2 Example

```
String haystack = "Google Cloud provides APIs to use Google's ML/AI capabilities.";
String[] needles = {"Google", "API", "documentation", "AWS", "ML/AI"};
new NeedlesInHaystack().findNeedles(haystack, needles);
```

1.1.3 Output

```
Google: 2
API: 0
documentation: 0
AWS: 0
ML/AI: 1
```

2 Suggestions for code improvement

After reviewing the code sample, I have the following suggestions for improvement:

- Limit the number of **`needles`**: The current code restricts the length of the `needles` array to five. If this is a strict requirement, modify the message in the print statement within the `if` block to "Use a maximum of five words!". This provides clearer guidance than "Too many...".
- Assign **`needles.length`** to a variable: Store `needles.length` in a variable to eliminate repeated evaluation of the same expression. This also improves memory efficiency.

- Optimize **`haystack.split()`**: Move the following statement outside of the loop: `String[] words = haystack.split("[ "'\t\n\b\f\r]", 0);`. The `words` array does not change with each iteration, so splitting the `haystack` only once is more efficient.
- Improve readability: In the third for loop, use `k` as the iterator variable, as `i` and `j` are already in use in nearby loops.
- Consider using a **`HashMap`**: To store the frequency of each needle, consider using a `HashMap`. This provides a more flexible and efficient way to handle the counting, especially if you want to return key-value pairs or add further functionality later.

2.1 Suggested code

Here is the revised code, which incorporates the suggested improvements for efficiency and clarity:

```
public class NeedlesInHaystack {
    public static void findNeedles(String haystack, String[] needles) {
        // Store the length of the needles array in a variable
        int needlesLength = needles.length;

        // Split the haystack string once
        String[] words = haystack.split("[ \"'\t\n\b\f\r]", 0);

        // Create an array to store the frequency counts
        int[] countArray = new int[needlesLength];

        // Iterate through the needles and count their occurrences
        for (int i = 0; i < needlesLength; i++) {
            for (int j = 0; j < words.length; j++) {
                if (words[j].compareTo(needles[i]) == 0) {
                    countArray[i]++;
                }
            }
        }

        // Print the results
        for (int k = 0; k < needlesLength; k++) {
            System.out.println(needles[k] + ": " + countArray[k]);
        }
    }

    public static void main(String[] args) {
        // Hard-coded values for demonstration
        // Ideally, values should be received from standard input
        String haystack = "Google Cloud provides APIs to use Google's ML/AI
    capabilities.";
        String[] needles = {"Google", "API", "documentation", "AWS", "ML/AI"};
        findNeedles(haystack, needles);
    }
}
```

2.2 Sample output

```
Google: 2
```

```
API: 0
documentation: 0
AWS: 0
ML/AI: 1
```