Find needles in a haystack

API reference document

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

public class NeedlesInHaystack

Method: `findNeedles`

- **Type**: `public static`
- **Parameters and datatypes**:
  - `String haystack`
  - `String[] needles`
- **Returns**: Nothing
- **Usage**: Prints the number of times each element of the string array `needles` is present in the string `haystack`.
- **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);
  ```

  Output:

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

Ideas about the code sample

After evaluating the code sample, I have the following questions and ideas:

- Why should we restrict the length of the `needles` array to five? If this is a strict requirement, consider modifying the message in the print statement within the `if` block to "Use a maximum of five words!". The modification eliminates the ambiguity in "Too many…".
- Assign `needles.length` to a variable, as it eliminates the necessity to evaluate the expression thrice in the code. Also, referencing to a variable is more memory efficient.
- To increase efficiency, take the following statement out of the first `for` loop: `String[] words = haystack.split("[ \"\'\t\n\b\f\r]", 0);`. The `words` array does not change with every iteration.
- To enhance readability, consider using `k` as the iterator in the third `for` loop, as `i` and `j` are already in use at close proximity.
- The central idea of the method is to act as a frequency counter and print key-value pairs. For such situations, a hashmap is a better data structure. Using a hashmap is also convenient when we want to return an object with key-value pairs.

Based on the discussion above, I propose the following revised code (not introducing hashmap to avoid too many changes, but adding *some padding for completeness*):

Revised code sample:

```java
public class NeedlesInHaystack {
  public static void findNeedles(String haystack, String[] needles) {
    int needlesLength = needles.length;
    String[] words = haystack.split("[ \"\'\t\n\b\f\r]", 0);
    int[] countArray = new int[needlesLength];
    for (int i = 0; i < needlesLength; i++) {
      for (int j = 0; j < words.length; j++) {
        if (words[j].compareTo(needles[i]) == 0) {
          countArray[i]++;
        }
      }
    }
    for (int k = 0; k < needlesLength; k++) {
      System.out.println(needles[k] + ": " + countArray[k]);
    }
  }
  public static void main(String[] args) {
    /* Hard-coded values used for demonstration only.
    Ideally, receive values 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);
  }
}
```

Sample output:

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