GTU DEPARTMENT of COMPUTER ENGINEERING

CSE222/505 – SPRING 2023

HOMEWORK 6 REPORT

ÖMER FARUK ÇOLAKEL

200104004043

1. SYSTEM REQUIREMENTS

Main:

```
public static String processString(String input)
```

This method requires a string as input. It removes non-letter characters, converts all uppercase letters to lower case letters and leaves spaces as is. Returns the processed string.

Info:

```
public info(String word)
```

This constructor sets count and words variables. Then calls push method to add the word to words arraylist.

```
public void push(String word)
```

This method requires a word to add. After addition, calls increment method to increment count variable.

myMap:

```
public myMap(String newStr)
```

This constructor requires a string. Sets map, mapSize and str. Then calls add method to map the str variable.

```
public void put(String key, info value)
```

This method adds a new key (String) and value (info) to map. Only sortedMap located in mergeSort objects use this method.

```
public info get(String key)
```

This method requires a key as string. It returns the info corresponding to given key. Only originalMap located in mergeSort objects use this method.

mergeSort:

```
public mergeSort(myMap map)
```

This constructor sets originalMap to map, sets sortedMap and aux string array. Then calls sort method and puts keys and info to sortedMap according to aux array.

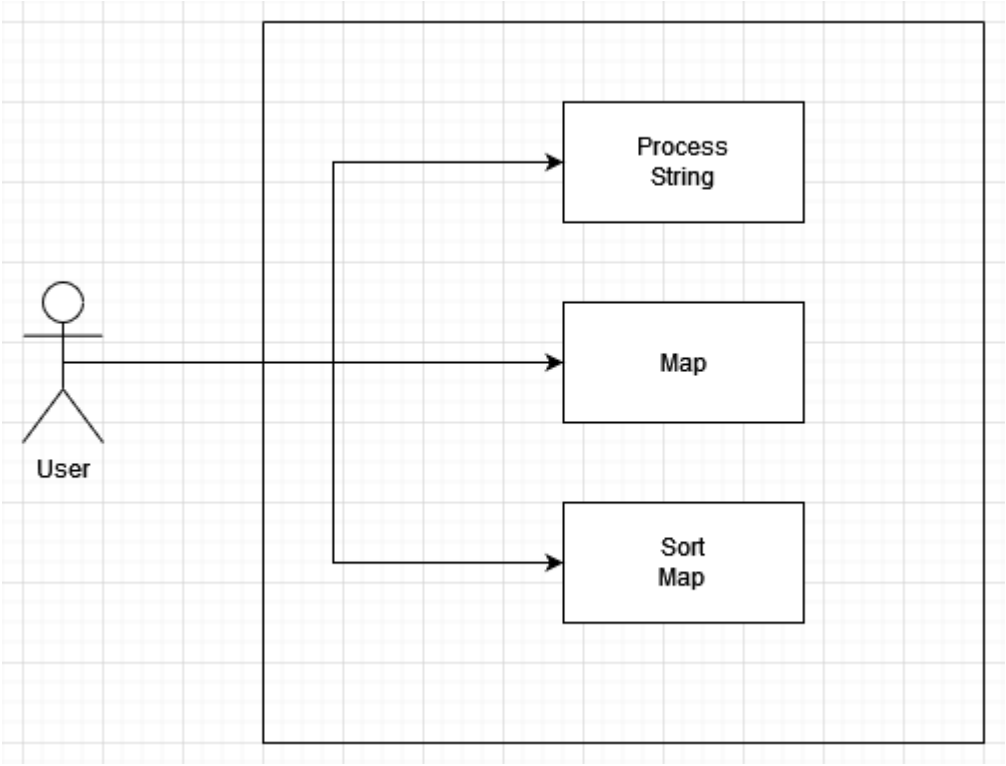
```
private void sort(int low, int high)
```

This method requires lowest and highest bound of aux array. Sorts and merges small parts of aux array then bigger parts.

```
private void merge(int low, int mid, int high)
```

This method requires lower, higher bounds and middle of aux array. Merges the subarray located between those bounds according to their count value at info and writes it to aux.

2. USE CASE DIAGRAM



3. PROBLEM SOLUTION APPROACH

To process the input string, a string builder is instantiated and traversed through the input. It appends lower-case letters as they are, upper case letters append it as lower-case letters, spaces append it as space and others are ignored. Then returned the string builder as string. Consecutive spaces are being added once to new string.

To map the processed string, str is divided into words and assigned to words string array. If str is empty or words array doesn't have any letters an exception is thrown. If there is a new letter, it's added to map as a key with info object. Also, mapSize is increased. If map contains a letter as key, count value in that key's info object is increased and added to words arraylist. mapSize doesn't increase. If the next character is a space, indexOfWord is increased. This variable holds the index of the word that is being mapped.

We used merge sort algorithm in this project. It uses the divide and conquer strategy. We first sort the array in small pieces and then merge to get a bigger piece. First, sort method splits array until it gets smallest array possible. Left side of array is sorted before the right side. With those indexes we call merge method. Left and right side in given indexes are copied to left and right arrays and we compare the values and change the values in aux array. At the we get a sorted array. Lastly, we get info according to keys in aux array from originalMap and put them in sortedMap.

4. TEST CASES AND RESULTS

Test 1 and Result

```java
public static void Test1()
{
    System.out.println("Test 1:");
    // Preprocess the string
    String input = "java";
    System.out.println("Original String: " + input);
    input = processString(input);
    System.out.println("Preprocessed String: " + input + "\n\n");

    // Create a map
    System.out.println("The original (unsorted) map:");
    myMap myMap = new myMap(input);
    System.out.println(myMap);
}
```

```
Test 1:
Original String: java
Preprocessed String: java


The original (unsorted) map:
Letter j - Count: 1 Words: [java]
Letter a - Count: 2 Words: [java, java]
Letter v - Count: 1 Words: [java]
```

Test 2 and Result

```java
public static void Test2()
{
    System.out.println("\nTest 2:");
    // Preprocess the string
    String input = "abc aba";
    System.out.println("Original String: " + input);
    input = processString(input);
    System.out.println("Preprocessed String: " + input + "\n\n");

    // Create a map
    System.out.println("The original (unsorted) map:");
    myMap myMap = new myMap(input);
    System.out.println(myMap);
}
```

```
Test 2:
Original String: abc aba
Preprocessed String: abc aba


The original (unsorted) map:
Letter a - Count: 3 Words: [abc, aba, aba]
Letter b - Count: 2 Words: [abc, aba]
Letter c - Count: 1 Words: [abc]
```

Test 3 and Result

```java
public static void Test3()
{
    System.out.println("\nTest 3:");
    // Preprocess the string
    String input = "buzzing bees buzz.";
    System.out.println("Original String: " + input);
    input = processString(input);
    System.out.println("Preprocessed String: " + input + "\n\n");

    // Create a map
    System.out.println("The original (unsorted) map:");
    myMap myMap = new myMap(input);
    System.out.println(myMap);

    // Sort the map
    System.out.println("\n\nThe sorted map:");
    mergeSort mergeSort = new mergeSort(myMap);
    System.out.println(mergeSort);
}
```

```
Test 3:
Original String: buzzing bees buzz.
Preprocessed String: buzzing bees buzz


The original (unsorted) map:
Letter b - Count: 3 Words: [buzzing, bees, buzz]
Letter u - Count: 2 Words: [buzzing, buzz]
Letter z - Count: 4 Words: [buzzing, buzzing, buzz, buzz]
Letter i - Count: 1 Words: [buzzing]
Letter n - Count: 1 Words: [buzzing]
Letter g - Count: 1 Words: [buzzing]
Letter e - Count: 2 Words: [bees, bees]
Letter s - Count: 1 Words: [bees]



The sorted map:
Letter i - Count: 1 Words: [buzzing]
Letter n - Count: 1 Words: [buzzing]
Letter g - Count: 1 Words: [buzzing]
Letter s - Count: 1 Words: [bees]
Letter u - Count: 2 Words: [buzzing, buzz]
Letter e - Count: 2 Words: [bees, bees]
Letter b - Count: 3 Words: [buzzing, bees, buzz]
Letter z - Count: 4 Words: [buzzing, buzzing, buzz, buzz]
```

Test 4 and Result

```java
public static void Test4()
{
    System.out.println("\nTest 4:");
    // Preprocess the string
    String input = "'Hush, hush!' whispered the rushing wind.";
    System.out.println("Original String: " + input);
    input = processString(input);
    System.out.println("Preprocessed String: " + input + "\n\n");

    // Create a map
    System.out.println("The original (unsorted) map:");
    myMap myMap = new myMap(input);
    System.out.println(myMap);

    // Sort the map
    System.out.println("\n\nThe sorted map:");
    mergeSort mergeSort = new mergeSort(myMap);
    System.out.println(mergeSort);
}
```

```
Test 4:
Original String: 'Hush, hush!' whispered the rushing wind.
Preprocessed String: hush hush whispered the rushing wind


The original (unsorted) map:
Letter h - Count: 7 Words: [hush, hush, hush, hush, whispered, the, rushing]
Letter u - Count: 3 Words: [hush, hush, rushing]
Letter s - Count: 4 Words: [hush, hush, whispered, rushing]
Letter w - Count: 2 Words: [whispered, wind]
Letter i - Count: 3 Words: [whispered, rushing, wind]
Letter p - Count: 1 Words: [whispered]
Letter e - Count: 3 Words: [whispered, whispered, the]
Letter r - Count: 2 Words: [whispered, rushing]
Letter d - Count: 2 Words: [whispered, wind]
Letter t - Count: 1 Words: [the]
Letter n - Count: 2 Words: [rushing, wind]
Letter g - Count: 1 Words: [rushing]


The sorted map:
Letter p - Count: 1 Words: [whispered]
Letter t - Count: 1 Words: [the]
Letter g - Count: 1 Words: [rushing]
Letter w - Count: 2 Words: [whispered, wind]
Letter r - Count: 2 Words: [whispered, rushing]
Letter d - Count: 2 Words: [whispered, wind]
Letter n - Count: 2 Words: [rushing, wind]
Letter u - Count: 3 Words: [hush, hush, rushing]
Letter i - Count: 3 Words: [whispered, rushing, wind]
Letter e - Count: 3 Words: [whispered, whispered, the]
Letter s - Count: 4 Words: [hush, hush, whispered, rushing]
Letter h - Count: 7 Words: [hush, hush, hush, hush, whispered, the, rushing]
```

Test 5 and Result

```
private static void Test5() {
    System.out.println("\nTest 5:");
    String input = "";
    input = processString(input);
    System.out.println("Preprocessed String: " + input);
    myMap myMap1 = new myMap(input);
    System.out.println("The original (unsorted) map:");
    System.out.println(myMap1);
}
```

```
Test 5:
Preprocessed String:
String cannot be empty

Process finished with exit code 0
```

Test 6 and Result

```
private static void Test6() {
    System.out.println("\nTest 6:");
    // Preprocess the string
    String input = " ";
    input = processString(input);
    System.out.println("Preprocessed String: " + input);
    myMap myMap = new myMap(input);
    System.out.println("The original (unsorted) map:");
    System.out.println(myMap);
}
```

```
Test 6:
Preprocessed String:
String cannot be empty

Process finished with exit code 0
```

Test 7 and Result

```
private static void Test7() {
    System.out.println("\nTest 7:");
    // Preprocess the string
    String input = "a   b ";
    input = processString(input);
    System.out.println("Preprocessed String: " + input);
    myMap myMap = new myMap(input);
    System.out.println("The original (unsorted) map:");
    System.out.println(myMap);
}
```

```
Test 7:
Preprocessed String: a b
The original (unsorted) map:
Letter a - Count: 1 Words: [a]
Letter b - Count: 1 Words: [b]
```

5. CLASS DIAGRAM