CSE 222
Data Structures
Report

Yavuz Selim İkizler
1901042617

Code Design

General Class Structures

```
public static String preprocess(String str)
```

I created a string in the Main class and tested it. First, I created a static method called 'preprocess' in the Main class to apply preprocess to the string.

The method first converts the input string to lowercase using the toLowerCase() method. This is done to make sure that all characters in the string are in the same case.

The method then uses regular expressions to remove any characters that are not letters or spaces from the input string. This is done using the Pattern class and the Matcher class. Specifically, the regular expression pattern [^a-z\\s] matches any character that is not a lowercase letter or a space. The Matcher class then finds all occurrences of this pattern in the input string and replaces them with an empty string using the appendReplacement() method. The appendTail() method is then called to append the remaining part of the input string to the StringBuffer. Finally, the toString() method is called on the StringBuffer to get the preprocessed string.

The preprocessed string is returned as the result of the method.

Afterwards, I sent this newly created string as a parameter to the constructor of myMap object.

```
public class myMap
```

Class myMap:

This class contains three different constructors and some helper methods. An integer variable named mapSize is used as a private variable to store the size of the LinkedHashMap. A string variable named str is used to store the string that will be processed, and a LinkedHashMap<String, info> named map is used to store the created map.

Constructures:

public myMap(String str):

The myMap class has a constructor that takes a string as input. This string is split into individual words using the split method, and then each word is added to an ArrayList of words. A LinkedHashMap named map is initialized to store the mapping of characters to words.

Next, a loop is run over all the words in the ArrayList. For each word, another loop runs over all the characters in the word. If the character is not already present in the char_list ArrayList, a new info object is created and the word is added to it using the push method. The character and info object are then added to the map using the put method, and the character is added to the char_list.

If the character is already present in the char_list, then a loop runs over all the entries in the map to find the entry corresponding to the character. The push method is called on the info object corresponding to the character, and the word is added to it.

Finally, the size of the char_list ArrayList is stored in the mapSize variable, and the input string is stored in the str variable.

Through this constructure, the string variable obtained from the parameter can be parsed and converted to a map variable.

In addition to this, there is a default constructor that initializes the map as empty. There is also a constructor that takes a LinkedHashMap variable as a parameter and assigns the map to this variable.

As helper methods, there is a method that checks whether a character is in the list called contains_letter. There is also a put method that adds new key and value pairs to the map, and a to_String method that prints the map in the desired format.

Class info:

The info class is used to store the value of the map variable. An int variable count is used as a private variable to keep track of the word count, and a String array words is used to store the words. It has a default constructor and a constructor that takes a string array as a parameter. The latter constructor assigns the string to the words string array.

There is a void method called push that adds the parameter string to the words array.

Getter methods include get_count which returns the value of the count variable, and get_words which concatenates the strings in the words array into a single string and returns it. There is also a get_words2 method which simply returns the words array.

Class mergeSort:

The mergeSort class is the class where the sorting algorithm is implemented. It has private variables such as myMap, which is the original map before sorting, myMap_sorted, which is the sorted map, and String array list aux, which is used as a helper variable for the sorting algorithm.

The mergeSort class has one constructor that takes a myMap variable. Private variables are filled based on this class. The originalMap is directly assigned with the parameter passed to the class. An aux ArrayList is created, and within a for loop, the key-value pairs of the original map are added to the aux list using the add method. Then, the mergeSort method is called to sort the aux list. Afterwards, a while loop runs until there are no key-value pairs left in the aux list. At each step, the key-value pair from the aux list is used to retrieve the corresponding info object from the original map, which is then added to the sorted map using the put method.

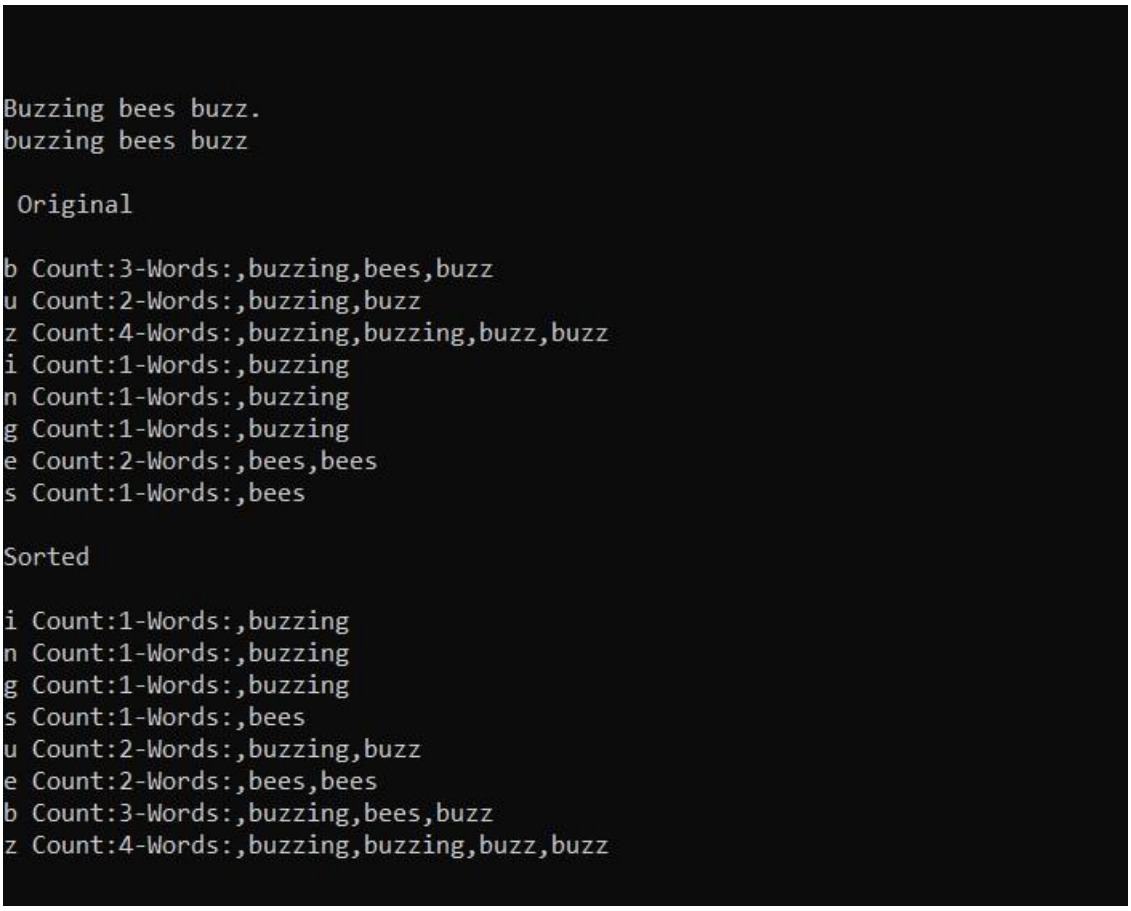
In addition to that, the mergeSort class has helper methods for the mergeSort algorithm and a method called printMaps. This method calls the toString methods of the original map and sorted map.

Method mergeSort:

The mergeSort method takes the input ArrayList arr and determines its size (n).It enters a loop that iterates over subarray sizes starting from 1 and doubling at each iteration, until the size is less than n-1.Within the outer loop, another loop runs to merge subarrays of the current size.For each subarray, it calculates the mid index based on the left start index and the subarray size. It also calculates the rightEnd index.It then calls the merge method to merge the two subarrays.The merge method takes the input ArrayList arr and the indices of the left, middle, and right elements.It calculates the sizes of the left and right subarrays.Two temporary ArrayLists (leftArr and rightArr) are created to hold the elements of the left and right subarrays, respectively.The data from arr is copied to the temporary ArrayLists using appropriate indices.The method enters a while loop that compares elements from the left and right subarrays and merges them into arr in a sorted manner.The comparison is performed based on the count property of the corresponding info objects obtained from the originalMap.If the count of the element in the left subarray is less than or equal to the count of the element in the right subarray, it is placed in arr at index k and i is incremented.If the count of the element in the right subarray is smaller, it is placed in arr at index k and j is incremented.The index k is always incremented.After the while loop, any remaining elements in the left or right subarrays are copied into arr.The mergeSort method completes when all subarrays are merged and sorted.

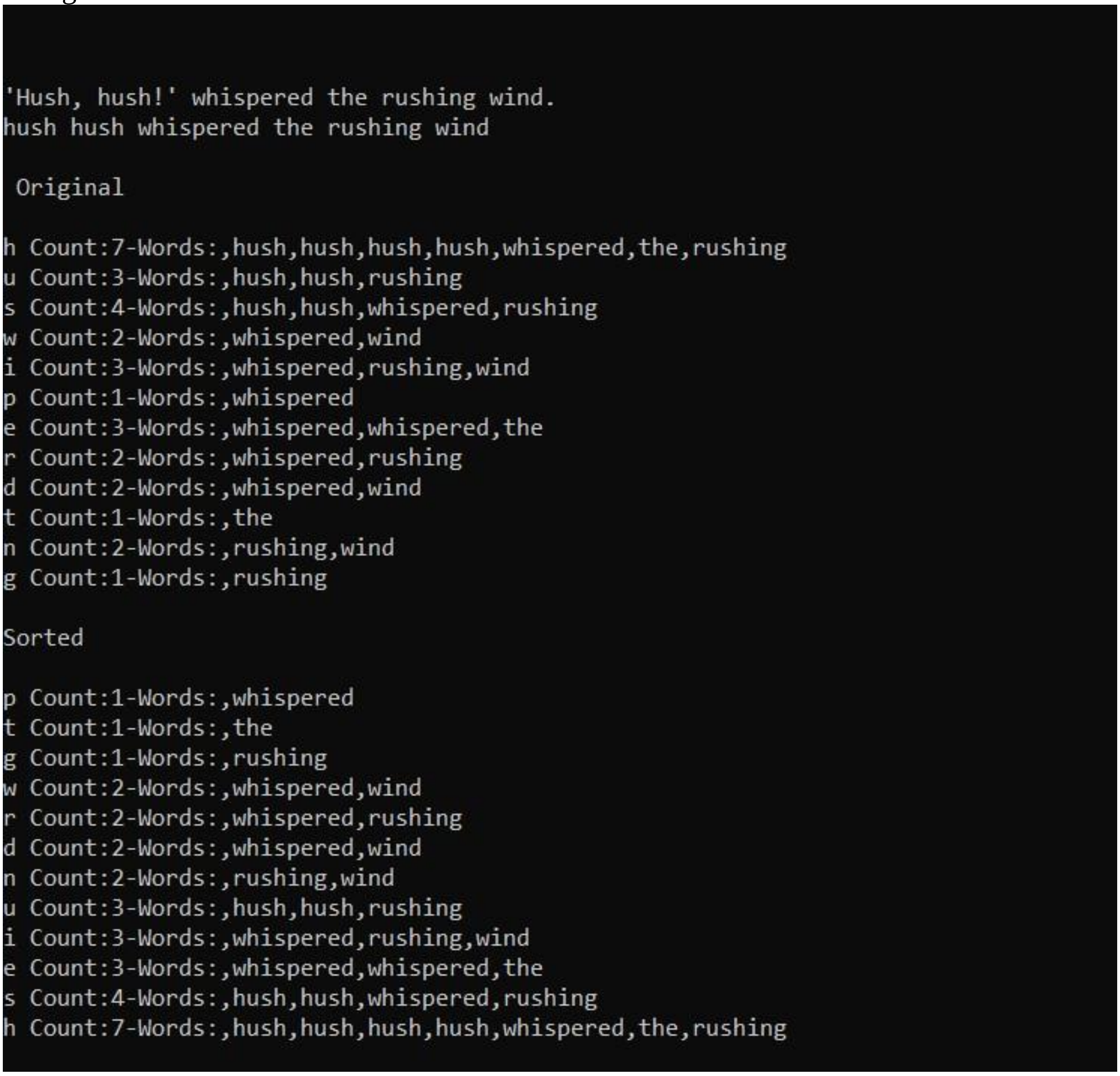
Test Outputs

String1:



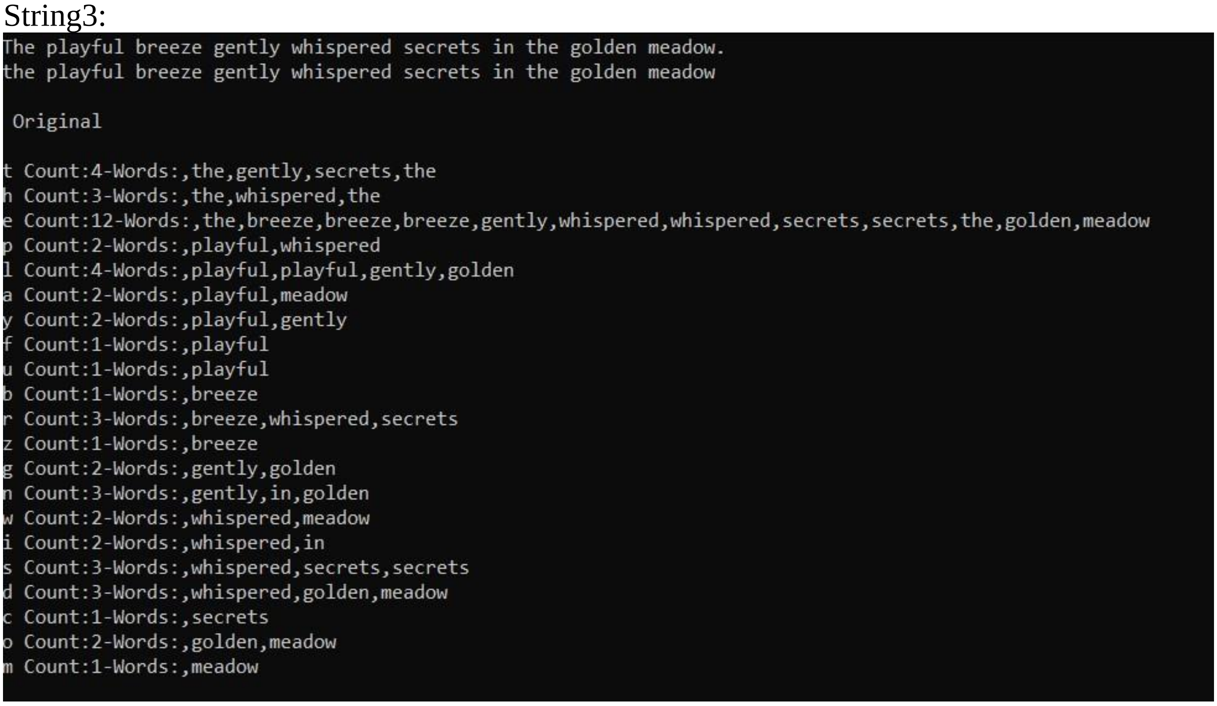
```
Buzzing bees buzz.
buzzing bees buzz

 Original

b Count:3-Words:,buzzing,bees,buzz
u Count:2-Words:,buzzing,buzz
z Count:4-Words:,buzzing,buzzing,buzz,buzz
i Count:1-Words:,buzzing
n Count:1-Words:,buzzing
g Count:1-Words:,buzzing
e Count:2-Words:,bees,bees
s Count:1-Words:,bees

Sorted

i Count:1-Words:,buzzing
n Count:1-Words:,buzzing
g Count:1-Words:,buzzing
s Count:1-Words:,bees
u Count:2-Words:,buzzing,buzz
e Count:2-Words:,bees,bees
b Count:3-Words:,buzzing,bees,buzz
z Count:4-Words:,buzzing,buzzing,buzz,buzz
```

String2:

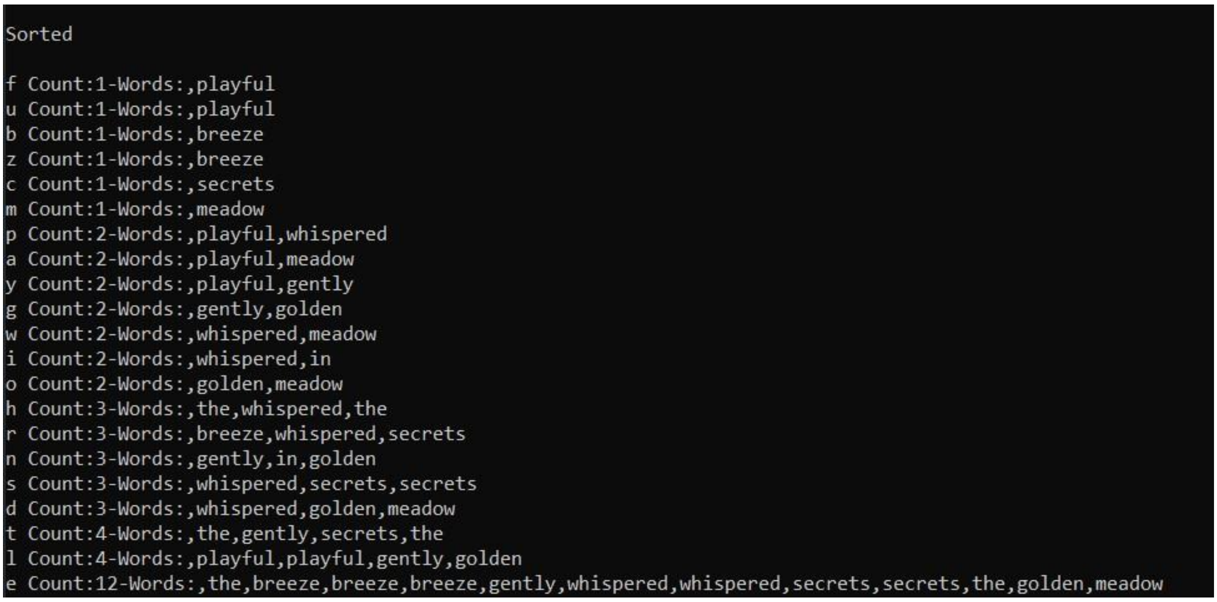


```
'Hush, hush!' whispered the rushing wind.
hush hush whispered the rushing wind

 Original

h Count:7-Words:,hush,hush,hush,hush,whispered,the,rushing
u Count:3-Words:,hush,hush,rushing
s Count:4-Words:,hush,hush,whispered,rushing
w Count:2-Words:,whispered,wind
i Count:3-Words:,whispered,rushing,wind
p Count:1-Words:,whispered
e Count:3-Words:,whispered,whispered,the
r Count:2-Words:,whispered,rushing
d Count:2-Words:,whispered,wind
t Count:1-Words:,the
n Count:2-Words:,rushing,wind
g Count:1-Words:,rushing

Sorted

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

String3:



```
The playful breeze gently whispered secrets in the golden meadow.
the playful breeze gently whispered secrets in the golden meadow

 Original

t Count:4-Words:,the,gently,secrets,the
h Count:3-Words:,the,whispered,the
e Count:12-Words:,the,breeze,breeze,breeze,gently,whispered,whispered,secrets,secrets,the,golden,meadow
p Count:2-Words:,playful,whispered
l Count:4-Words:,playful,playful,gently,golden
a Count:2-Words:,playful,meadow
y Count:2-Words:,playful,gently
f Count:1-Words:,playful
u Count:1-Words:,playful
b Count:1-Words:,breeze
r Count:3-Words:,breeze,whispered,secrets
z Count:1-Words:,breeze
g Count:2-Words:,gently,golden
n Count:3-Words:,gently,in,golden
w Count:2-Words:,whispered,meadow
i Count:2-Words:,whispered,in
s Count:3-Words:,whispered,secrets,secrets
d Count:3-Words:,whispered,golden,meadow
c Count:1-Words:,secrets
o Count:2-Words:,golden,meadow
m Count:1-Words:,meadow
```



```
Sorted

f Count:1-Words:,playful
u Count:1-Words:,playful
b Count:1-Words:,breeze
z Count:1-Words:,breeze
c Count:1-Words:,secrets
m Count:1-Words:,meadow
p Count:2-Words:,playful,whispered
a Count:2-Words:,playful,meadow
y Count:2-Words:,playful,gently
g Count:2-Words:,gently,golden
w Count:2-Words:,whispered,meadow
i Count:2-Words:,whispered,in
o Count:2-Words:,golden,meadow
h Count:3-Words:,the,whispered,the
r Count:3-Words:,breeze,whispered,secrets
n Count:3-Words:,gently,in,golden
s Count:3-Words:,whispered,secrets,secrets
d Count:3-Words:,whispered,golden,meadow
t Count:4-Words:,the,gently,secrets,the
l Count:4-Words:,playful,playful,gently,golden
e Count:12-Words:,the,breeze,breeze,breeze,gently,whispered,whispered,secrets,secrets,the,golden,meadow
```