GEBZE TECHNICAL UNIVERSITY

CSE 222
DATA STRUCTURES AND ALGORITHMS

HOMEWORK 6
REPORT

OGUZ MUTLU
1801042624

1. DETAILED SYSTEM REQUIREMENTS

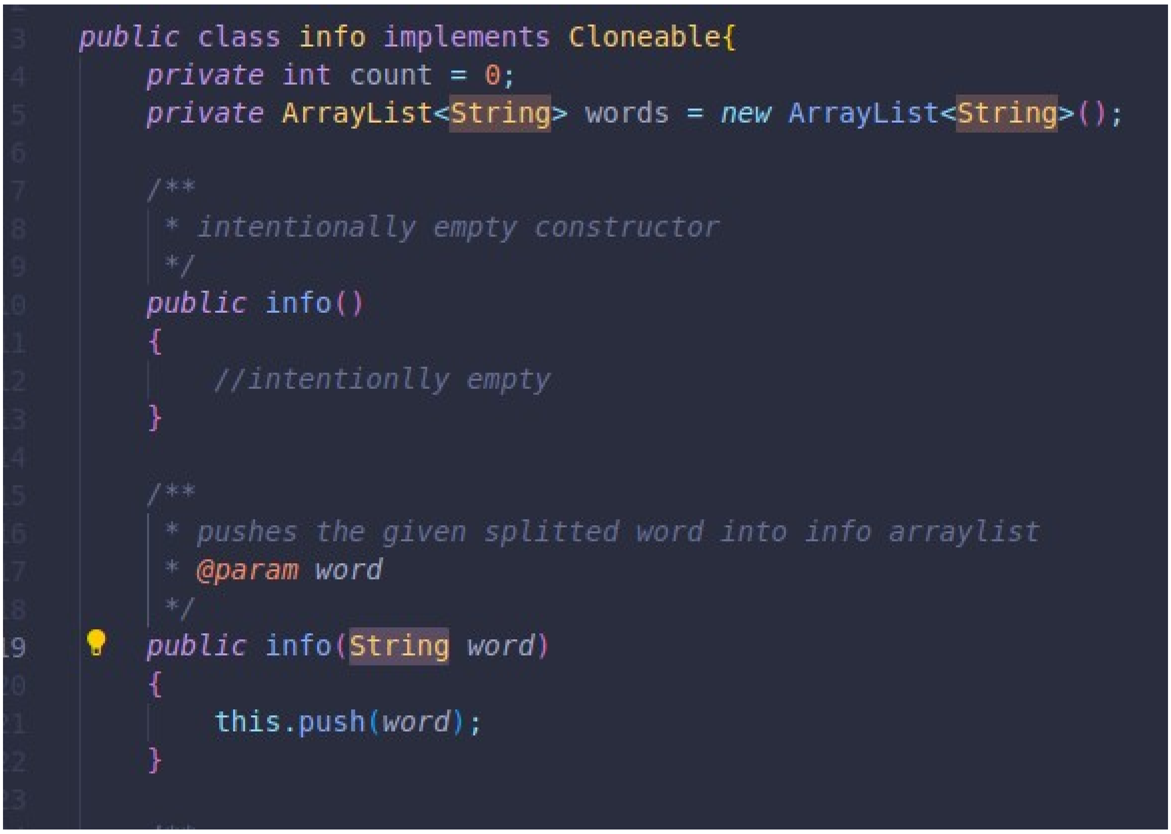
- Take an input string and preprocess that string, if there is capital letter, convert it to lower case letter, If there is a character which is not a letter or space, delete it. Spaces should be kept in order
to keep track of the words.

- By using the preprocessed string we have to build custom map structure, in this structure we will have to keep count of occurences of each letter withing the string along with the words that it was seen

- After custom my map build operation is done we have to sort this array by using count values of myMap class we have to do merge sort on that array

2. PROBLEM SOLUTION APPROACH

-info class

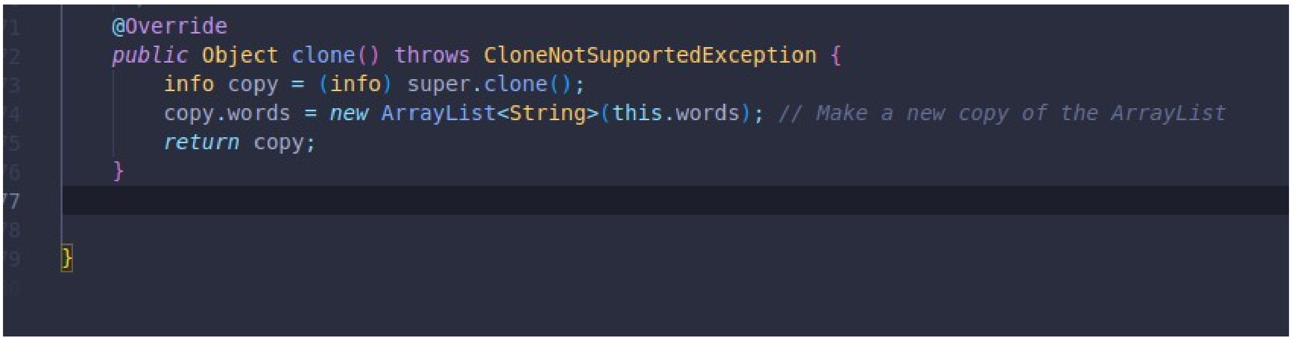


```java
public class info implements Cloneable{
    private int count = 0;
    private ArrayList<String> words = new ArrayList<String>();

    /**
     * intentionally empty constructor
     */
    public info()
    {
        //intentionlly empty
    }

    /**
     * pushes the given splitted word into info arraylist
     * @param word
     */
    public info(String word)
    {
        this.push(word);
    }
```

In the info class, in the myMap class, I put the words that I split with the split function into the words arraylist and keep their numbers and which words they are.

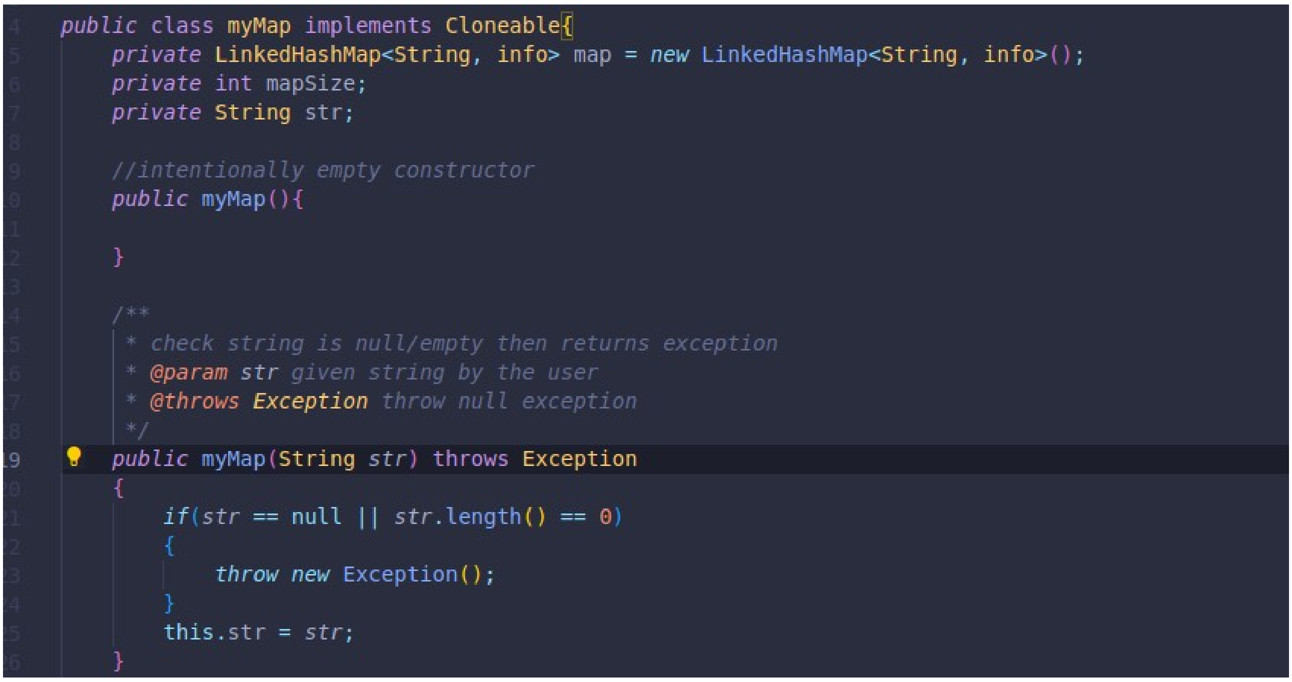


```
    @Override
    public Object clone() throws CloneNotSupportedException {
        info copy = (info) super.clone();
        copy.words = new ArrayList<String>(this.words); // Make a new copy of the ArrayList
        return copy;
    }


}
```

This class is implemented from cloneable because there is a sortedMap fields that will require deep copy in the myMap class, which will be used later. Equalizes all fields in the Info class with the new object.

-myMap class



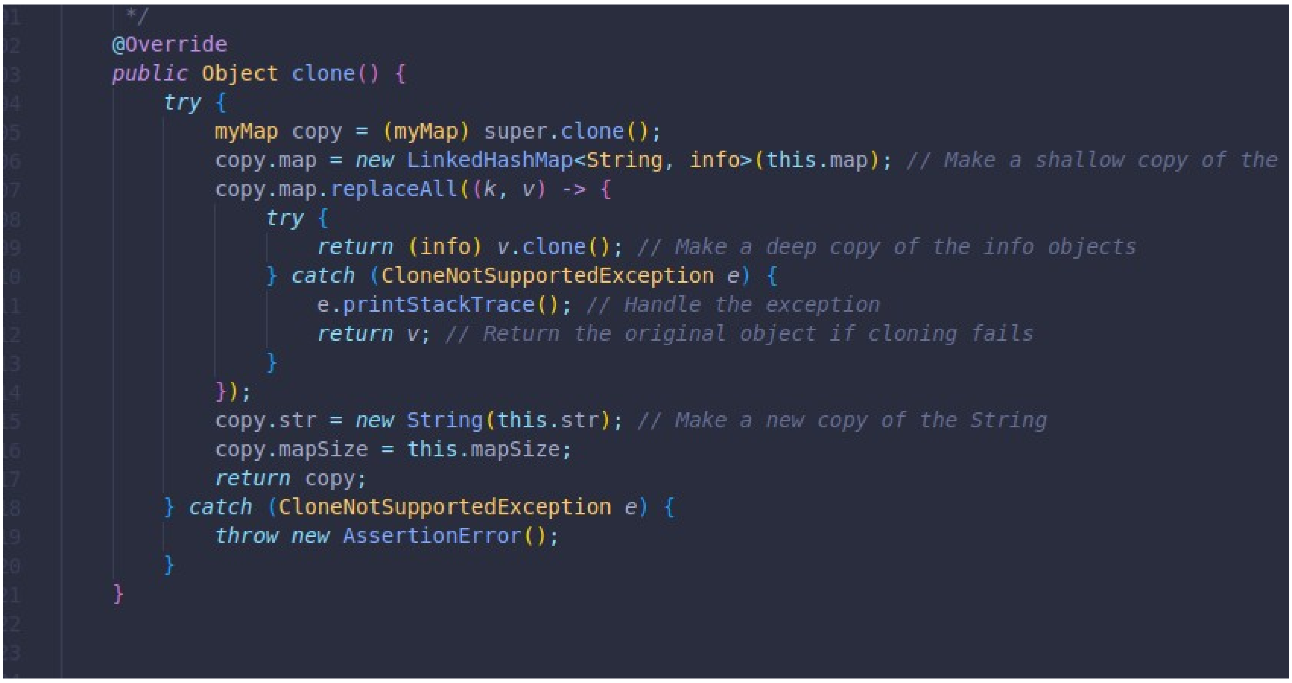
```
public class myMap implements Cloneable{
    private LinkedHashMap<String, info> map = new LinkedHashMap<String, info>();
    private int mapSize;
    private String str;

    //intentionally empty constructor
    public myMap(){

    }

    /**
     * check string is null/empty then returns exception
     * @param str given string by the user
     * @throws Exception throw null exception
     */
    public myMap(String str) throws Exception
    {
        if(str == null || str.length() == 0)
        {
            throw new Exception();
        }
        this.str = str;
    }
```

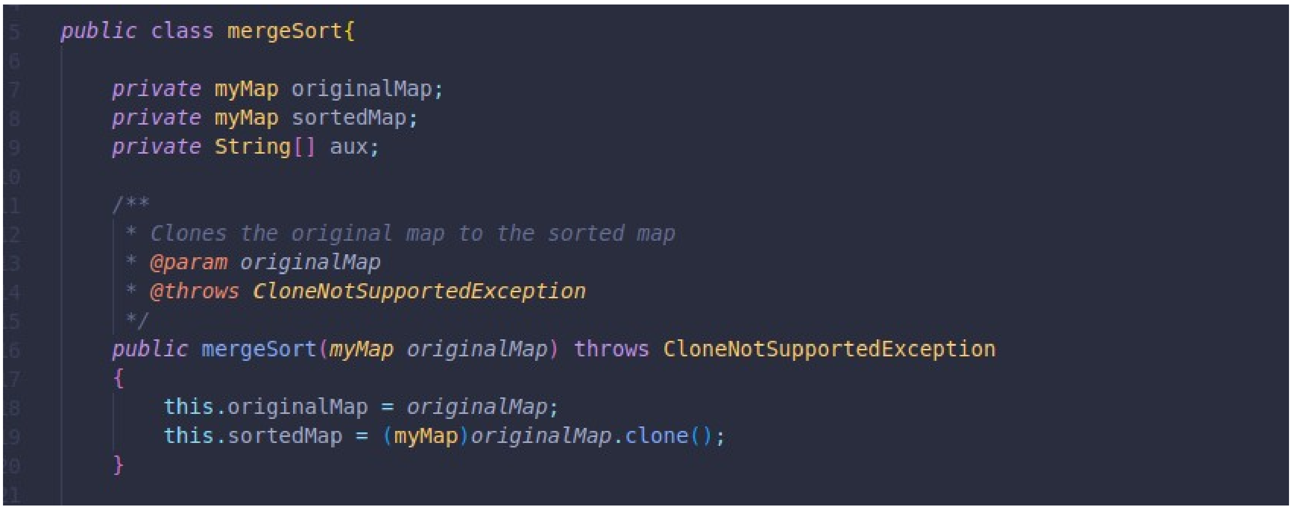
In this class, firstly, the preprocessed string is defined to the variable str, then these string values are extracted according to the space character, and parsed into characters. Then, each character is added to the LinkedHashMap structure one by one, if there are two of the same key value, the push function in the info class is called and the words are added to the info class one by one.In this way, the addition structure with push given in the content of the assignment is complied with.



```java
     */
    @Override
    public Object clone() {
        try {
            myMap copy = (myMap) super.clone();
            copy.map = new LinkedHashMap<String, info>(this.map); // Make a shallow copy of the
            copy.map.replaceAll((k, v) -> {
                try {
                    return (info) v.clone(); // Make a deep copy of the info objects
                } catch (CloneNotSupportedException e) {
                    e.printStackTrace(); // Handle the exception
                    return v; // Return the original object if cloning fails
                }
            });
            copy.str = new String(this.str); // Make a new copy of the String
            copy.mapSize = this.mapSize;
            return copy;
        } catch (CloneNotSupportedException e) {
            throw new AssertionError();
        }
    }
```

This class is implemented from cloneable because there is a sortedMap fields that will require deep copy in the myMap classes sorted map and original map, which will be used later. Equalizes all fields in the Info class with the new object

-mergeSort class

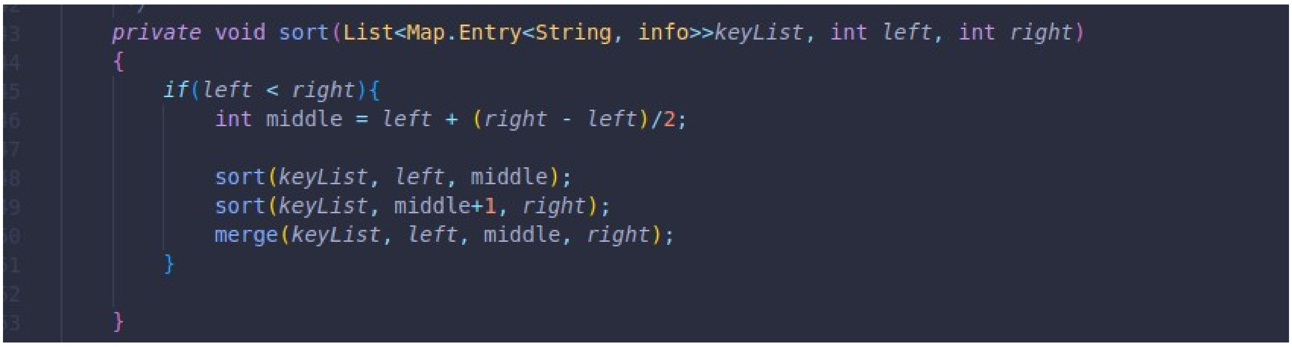


```java
public class mergeSort{

    private myMap originalMap;
    private myMap sortedMap;
    private String[] aux;

    /**
     * Clones the original map to the sorted map
     * @param originalMap
     * @throws CloneNotSupportedException
     */
    public mergeSort(myMap originalMap) throws CloneNotSupportedException
    {
        this.originalMap = originalMap;
        this.sortedMap = (myMap)originalMap.clone();
    }
```

This class implements the merge sort algorithm that class constructor deep copies the original map to the sorted mapIf the originalMap value created is a valid structure, this value is sent to the MergeSort() method for sorting.

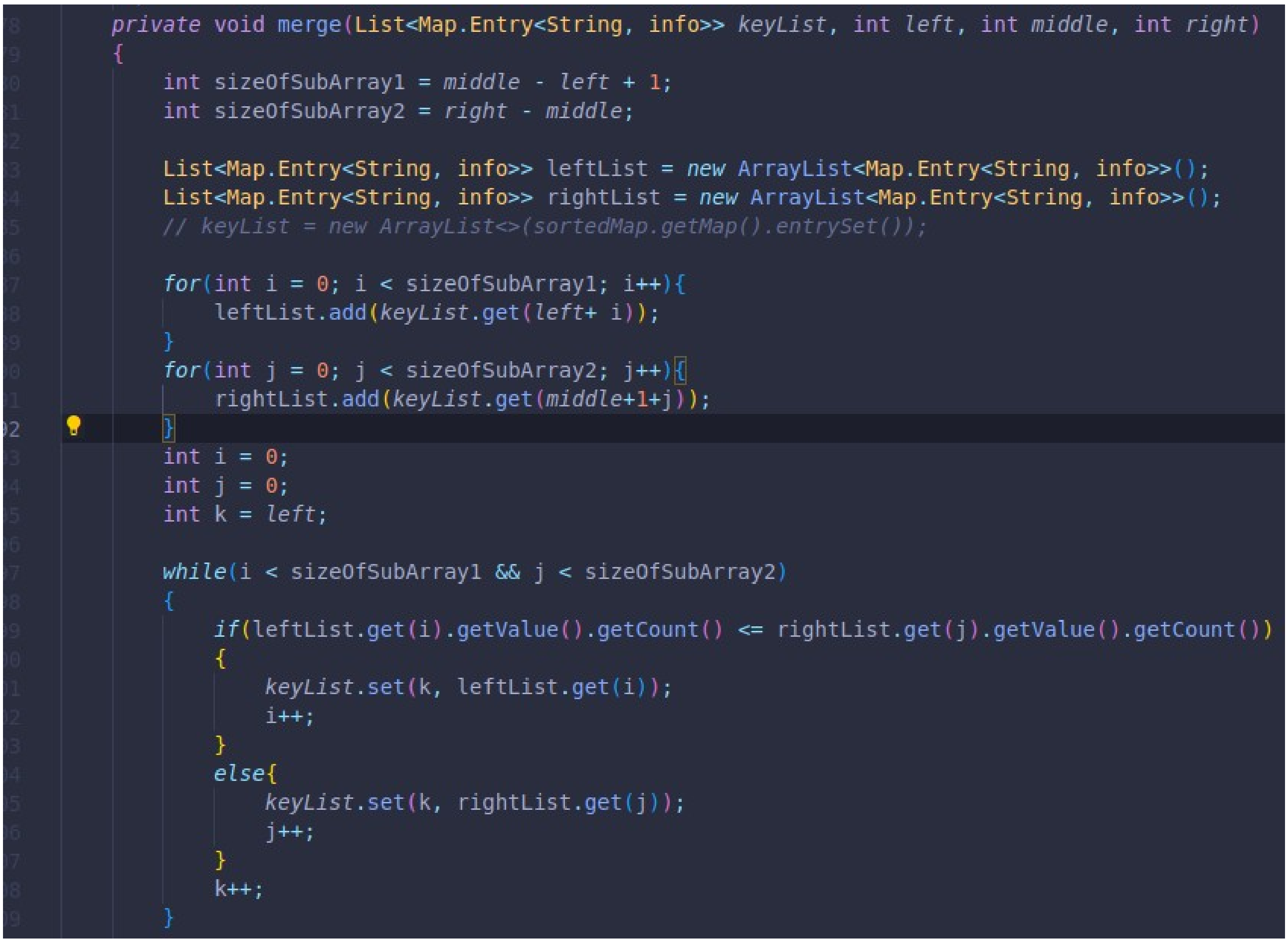


```
private void sort(List<Map.Entry<String, info>>keyList, int left, int right)
{
    if(left < right){
        int middle = left + (right - left)/2;

        sort(keyList, left, middle);
        sort(keyList, middle+1, right);
        merge(keyList, left, middle, right);
    }

}
```

In the MergeSort method, a variable named listkeypair is defined and the key and value values in the map are assigned to this variable, then the sort operation will be applied from these key and value values.
At first check if the left index of array is less than the right index if yes calculate its mid point. This sort process divides map into two halves until we reach atomic values. So we iterate it until atomic values are found. Then we compare this indexes one by one based on comparison of size of elements, first we compare the element for each part and then combine them into another new variable. After final merge operation done our collection must be sorted.



```
private void merge(List<Map.Entry<String, info>> keyList, int left, int middle, int right)
{
    int sizeOfSubArray1 = middle - left + 1;
    int sizeOfSubArray2 = right - middle;

    List<Map.Entry<String, info>> leftList = new ArrayList<Map.Entry<String, info>>();
    List<Map.Entry<String, info>> rightList = new ArrayList<Map.Entry<String, info>>();
    // keyList = new ArrayList<>(sortedMap.getMap().entrySet());

    for(int i = 0; i < sizeOfSubArray1; i++){
        leftList.add(keyList.get(left+ i));
    }
    for(int j = 0; j < sizeOfSubArray2; j++){
        rightList.add(keyList.get(middle+1+j));
    }
    int i = 0;
    int j = 0;
    int k = left;

    while(i < sizeOfSubArray1 && j < sizeOfSubArray2)
    {
        if(leftList.get(i).getValue().getCount() <= rightList.get(j).getValue().getCount())
        {
            keyList.set(k, leftList.get(i));
            i++;
        }
        else{
            keyList.set(k, rightList.get(j));
            j++;
        }
        k++;
    }
```

3.TEST CASES AND RESULTS

Case -1 Steps

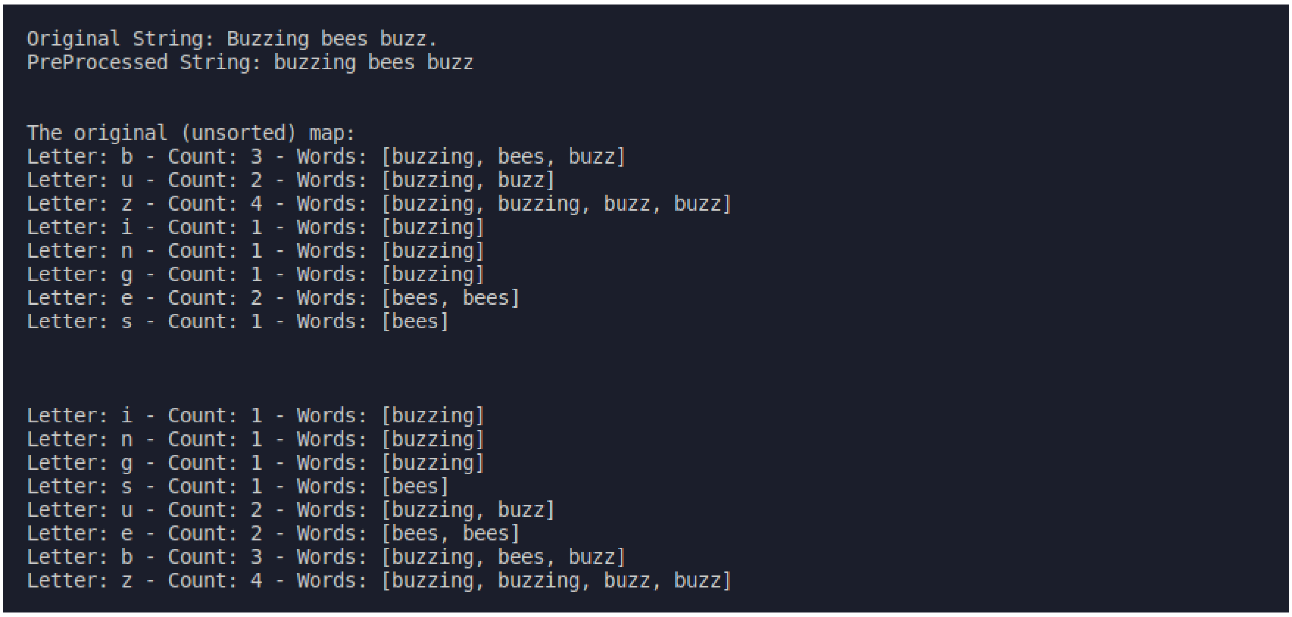
1. give valid string “Buzzing bees buzz.”
2. sorted map will be shown

Expectation:

- sorted map will be shown

Result

- **PASS**



```
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]



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]
```

Case -2 Steps

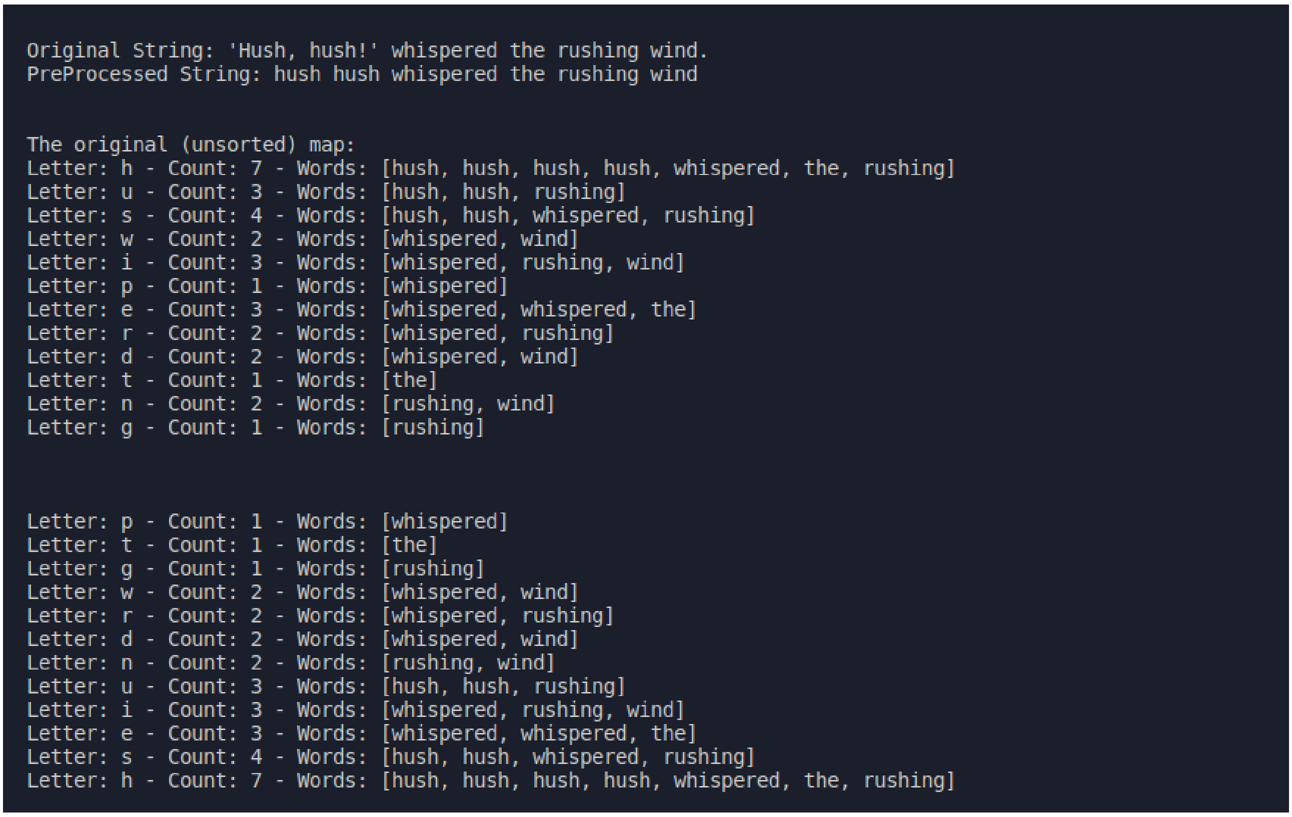
1. give valid string “Hush, hush!’ whispered the rushing wind”
2. sorted map will be shown

Expectation:

- sorted map will be shown

Result

- PASS



```
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]



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]
```

Case -3 Steps

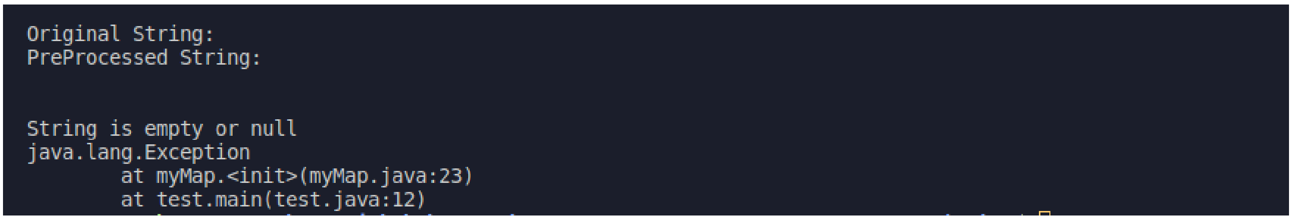
1. enter empty string str = “”

Expectation:

- Throw empty string into the terminal

Result

- PASS



```
Original String:
PreProcessed String:


String is empty or null
java.lang.Exception
        at myMap.<init>(myMap.java:23)
        at test.main(test.java:12)
```