Homework 6 Report

Batuhan Başak

The Implementation

The aim of this homework is sorting instance of class MyMap which is a class described in homework by using merge sort algorithm.
There are 3 classes to do this which are MyMap class and MergeSortMyMap class, and Info class.

Info Class

The Info is a class to store data value of class MyMap. The data value is count of key value and the list of words that contain key value in string. The details of words and the string will been detailed in MyMap class. An instance of Info class is initialized by given the count value. The words are added to list word by word by calling the method called push.
There are also accessor and mutator method for count and overloaded toString method that returns count and list of words in String.

```
public static class Info{
    /** The count of key character appeared in string. */
    private int count;
    /** The list of words that contains key character in string. */
    private final List<String> words;

    /**
     * The constructor that takes count.
     * @param count The count of character.
     */
    private Info( int count) {
        this.count = count;
        this.words = new ArrayList<>();
    }

    /**
     * The getter method for count.
     * @return The count.
     */
    public int getCount(){
```

```
        return count;
    }

    /**
     * The setter method for count
     * @param count The new count.
     * post: Sets the count value with given new count value.
     */
    private void setCount(int count){
        this.count = count;
    }

    /**
     * Adds new word to word list.
     * @param word The word to add it on word list.
     */
    private void push(String word){
        words.add(word);
    }

    /**
     * The toString method for Info class that returns count and word list in string.
     * @return The count and word list in string form.
     */
    public String toString(){
        return "Count: " + count + " - Words: " + words.toString() + "\n";
    }

}
```

MyMap Class

The constructor classes

The MyMap class designed to preproccess given string and store each character in preproccessed string in an underlying LinkedHashMap class. The data value of key character is an Info class that consists of count of key character in preproccessed string and list of words that contains key character in preprocced string.
There are two constructor methods. The first one takes nothing as parameter and just initialize underlying LinkedHashMap. The second one takes string, preprocesses it, and then fills the underlying LinkedHashMap respect to the preproccessed string. The second string also prints given string and preproccessed string respectively.

```
/** The underlying hash map that stores key, Info pairs. */
private final LinkedHashMap<Character, Info> map;

/** The preprocessed string */
String str;


/** The constructor method */
public MyMap() {
    mapSize = 0;
    map = new LinkedHashMap<>();
}


/**
 * The constructor method that takes string.
 * @param s The string being preprocessed.
 */
public MyMap(String s) {
    // Preprocess string s and set str.
    str = preprocessString(s);

    // Display original and preprocessed strings.
    System.out.println("Original String:      " + s + "\n"
        +              "Preprocessed String:  " + str);


    mapSize = 0;
    map = new LinkedHashMap<>();

    // Fill map respect to the preprocessed string str.
    fillMap();
}
```

The fillMap method

The fillMap methods is helper method for second constructor that fills the map with characters in preproccessed string. It first iterates each word in string, for new characters put them to list with count value of 0, then increase count by one and adds the current word to words list of character.

```
private void fillMap(){
    for (String word : str.split(" ")){
        for (int i = 0; i < word.length(); i++){
            if (map.get(word.charAt(i)) == null){
```

```
                map.put(word.charAt(i), new Info(0));
            }
            map.get(word.charAt(i)).setCount(map.get(word.charAt(i)).getCount()+1);
            map.get(word.charAt(i)).push(word);
        }
    }
}
```

The preprocessString method

The preprocessString method takes a string and forms it. What it does is that removes punctuation characters and makes all characters lower case. Then returns the proccessed string.

```
public static String preprocessString(String s) {
    String processedString = "";
    for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        if ((c >= 'a' && c <= 'z') ||
                (c >= 'A' && c <= 'Z') ||
                c == ' ') {
            processedString += Character.toLowerCase(c);
        }
    }
    return processedString;
}
```

The toString method

The overload toString method returns each item in map in string.

```
public String toString(){
    String s = "";
    Set<Character> keySet = map.keySet();
    for (char key : keySet){
        s += "Letter: " + key + " - " + map.get(key).toString();
    }
    return s;
}
```

The getMap method

The getMap method is a getter method for map that basicly returns the underlying LinkedHashMap map.

```
public LinkedHashMap<Character, Info> getMap(){
    return map;
}
```

The put method

Puts a new item char c and value of Info i to the underlying map by using delegation.

```
public void put(char c, Info i) {
    map.put(c, i);
}
```

The get method

Gets the value of key which is given character by using delegation.

```
public Info get(char c) {
    return map.get(c);
}
```

The Entire code of MyMap class

```
public class MyMap {
    /** The underlying hash map that stores key, Info pairs. */
    private final LinkedHashMap<Character, Info> map;
    /** The size of map */
    private int mapSize;

    /** The preprocessed string */
    String str;



    /** The constructor method */
    public MyMap() {
        mapSize = 0;
        map = new LinkedHashMap<>();
    }



    /**
     * The constructor method that takes string.
     * @param s The string being preprocessed.
     */
    public MyMap(String s) {
        // Preprocess string s and set str.
        str = preprocessString(s);

        // Display original and preprocessed strings.
        System.out.println("Original String:      " + s + "\n"
```

```
        +              "Preprocessed String:  " + str);


    mapSize = 0;
    map = new LinkedHashMap<>();

    // Fill map respect to the preprocessed string str.
    fillMap();
}


/**
 * Info is a class to store data of class MyMap that stores information of
 * key character in map.<br/>
 * The <em>count</em> is the count of key occurred in preprocessed string.<br/>
 * The <em>words</em> is the list of words that contain key in string.
 *
 */
public static class Info{
 /** The count of key character appeared in string. */
    private int count;
    /** The list of words that contains key character in string. */
    private final List<String> words;

    /**
     * The constructor that takes count.
     * @param count The count of character.
     */
    private Info( int count) {
        this.count = count;
        this.words = new ArrayList<>();
    }

    /**
     * The getter method for count.
     * @return The count.
     */
    public int getCount(){
        return count;
    }

    /**
     * The setter method for count
     * @param count The new count.
     * post: Sets the count value with given new count value.
     */
```

```
    private void setCount(int count){
        this.count = count;
    }

    /**
     * Adds new word to word list.
     * @param word The word to add it on word list.
     */
    private void push(String word){
        words.add(word);
    }

    /**
     * The toString method for Info class that returns count and word list in string.
     * @return The count and word list in string form.
     */
    public String toString(){
        return "Count: " + count + " - Words: " + words.toString() + "\n";
    }

}


/**
 * Fills map
 */
private void fillMap(){
    for (String word : str.split(" ")){
        for (int i = 0; i < word.length(); i++){
            if (map.get(word.charAt(i)) == null){
                map.put(word.charAt(i), new Info(0));
            }
            map.get(word.charAt(i)).setCount(map.get(word.charAt(i)).getCount()+1);
            map.get(word.charAt(i)).push(word);
        }
    }
}



/**
 * Preprocesses the given string and returns the result.<br/>
 * What it does that removes the punctuation characters, make the string lower case, and
 * keeps the white spaces as they are.
 * @param s The string to preprocess.
 * @return The preprocessed string.
```

```
     */
    public static String preprocessString(String s) {
        String processedString = "";
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if ((c >= 'a' && c <= 'z') ||
                    (c >= 'A' && c <= 'Z') ||
                    c == ' ') {
                processedString += Character.toLowerCase(c);
            }
        }
        return processedString;
    }


    /**
     * toString method for MyMap class.
     * @return The map in string form.
     */
    public String toString(){
        String s = "";
        Set<Character> keySet = map.keySet();
        for (char key : keySet){
            s += "Letter: " + key + " - " + map.get(key).toString();
        }
        return s;
    }

    public LinkedHashMap<Character, Info> getMap(){
     return map;
    }

    public void put(char c, Info i) {
     map.put(c, i);
    }

    public Info get(char c) {
     return map.get(c);
    }

}
```

MergeSortMyMap Class

It has two underlying MyMap instance that one of them holds unsorted version of given map and the other one holds the sorted version of given map. The given map is the map that is takes as parameter by the constructor method of this class. When the constructor method is invoke it sets unsorted map, sorts it by using merge sort algorithm, and stores the sorted version in sortedMap. You can get sorted map by getter method called getSortedMap which returns the sortedMap.

The constructor class

The constructor that takes intance of MyMap class to create sorted version of it. It uses an auxiliary array, how recommended in homework, that holds the key characters. The mergeSort method takes this array gets the values of the keys from unsortedMap, compare them and respect to their count values of key characters, it sorts key characters in auxiliary array. After sorting is done, respect to order in auxiliary array key characters and their values are putted to sorted map.

```
/** The unsorted map */
private MyMap unsortedMap;
/** The sorted, result, map */
private MyMap sortedMap;

public MergeSortMyMap(MyMap unsortedMap) {
    this.unsortedMap = unsortedMap;
    this.sortedMap = new MyMap();

    // Create an auxiliary array to hold keys of unsorted map.
    int unsortedMapSize = unsortedMap.getMap().size();
    char[] arr = new char[unsortedMapSize];
    // Fill the auxiliary array.
    Set<Character> keySet = unsortedMap.getMap().keySet();
    int i = 0;
    System.out.println("keySet : " + keySet);
    for (char c : keySet) {
        arr[i++] = c;
    }

    // Sort the key array respect to its count value.
    mergeSort(arr);
    // The keys in arr are sorted. Fill the sorted map respect to
    // the order of arr.
    for (i = 0; i < arr.length; i++) {
        sortedMap.put(arr[i], unsortedMap.get(arr[i]));
    }
```

```
}
```

The mergeSort and merge methods

They basically implements the merge sort algorithm.

```
// The merge sort algorithm implementation for MyMap class.
// Gets the array of keys and compare them respect to
// the count value of entry class Info.
private void mergeSort(char[] arr) {
    int size = arr.length;
    if (size <= 1) {
        return;
    }

    int mid = size / 2;
    char[] left = new char[mid];
    char[] right = new char[size-mid];
    for (int i = 0; i < mid; i++)
        left[i] = arr[i];
    for (int i = mid; i < size; i++)
        right[i-mid] = arr[i];

    mergeSort(left);
    mergeSort(right);
    merge(left, right, arr);
}


// The helper method for mergeSort method by merging arrays.
private void merge(char[] left, char[] right, char[] arr) {
    int leftIndex = 0;
    int rightIndex = 0;
    int arrIndex = 0;


    while (leftIndex < left.length && rightIndex < right.length) {
        int countLeftOne = unsortedMap.get(left[leftIndex]).getCount();
        int countRightOne = unsortedMap.get(right[rightIndex]).getCount();
        if (countLeftOne <= countRightOne) {
            arr[arrIndex++] = left[leftIndex++];
        }
        else {
            arr[arrIndex++] = right[rightIndex++];
        }
    }
```

```
    while (leftIndex < left.length) {
        arr[arrIndex++] = left[leftIndex++];
    }

    while (rightIndex < right.length) {
        arr[arrIndex++] = right[rightIndex++];
    }
}
```

getSortedMap method

Returns the sortedMap.

```
public MyMap getSortedMap() {
    return sortedMap;
}
```

The entire code of MergeSortMyMap Class

```
/**
 * Implementation of a class that sorts given MyMap by using merge sort algorithm.
 */
public class MergeSortMyMap {
/** The unsorted map */
private MyMap unsortedMap;
/** The sorted, result, map */
private MyMap sortedMap;

/**
 * The constructor that takes intance of MyMap class to create sorted version of it.
 * @param unsortedMap The intance of MyMap class whose map is unsorted.
 */
public MergeSortMyMap(MyMap unsortedMap) {
this.unsortedMap = unsortedMap;
this.sortedMap = new MyMap();

// Create an auxiliary array to hold keys of unsorted map.
int unsortedMapSize = unsortedMap.getMap().size();
char[] arr = new char[unsortedMapSize];
// Fill the auxiliary array.
Set<Character> keySet = unsortedMap.getMap().keySet();
int i = 0;
System.out.println("keySet : " + keySet);
for (char c : keySet) {
arr[i++] = c;
}
```

```
// Sort the key array respect to its count value.
mergeSort(arr);
// The keys in arr are sorted. Fill the sorted map respect to
// the order of arr.
for (i = 0; i < arr.length; i++) {
sortedMap.put(arr[i], unsortedMap.get(arr[i]));
}
}


// The merge sort algorithm implementation for MyMap class.
// Gets the array of keys and compare them respect to
// the count value of entry class Info.
private void mergeSort(char[] arr) {
int size = arr.length;
if (size <= 1) {
return;
}

int mid = size / 2;
char[] left = new char[mid];
char[] right = new char[size-mid];
for (int i = 0; i < mid; i++)
left[i] = arr[i];
for (int i = mid; i < size; i++)
right[i-mid] = arr[i];

mergeSort(left);
mergeSort(right);
merge(left, right, arr);
}


// The helper method for mergeSort method by merging arrays.
private void merge(char[] left, char[] right, char[] arr) {
int leftIndex = 0;
int rightIndex = 0;
int arrIndex = 0;


while (leftIndex < left.length && rightIndex < right.length) {
int countLeftOne = unsortedMap.get(left[leftIndex]).getCount();
int countRightOne = unsortedMap.get(right[rightIndex]).getCount();
if (countLeftOne <= countRightOne) {
arr[arrIndex++] = left[leftIndex++];
```

```
}
else {
arr[arrIndex++] = right[rightIndex++];
}
}

while (leftIndex < left.length) {
arr[arrIndex++] = left[leftIndex++];
}

while (rightIndex < right.length) {
arr[arrIndex++] = right[rightIndex++];
}
}

/**
 * The getter method for sortedMap. The sortedMap is an instance of MyMap class.
 * @return The sortedMap.
 */
public MyMap getSortedMap() {
return sortedMap;
}

}
```