myMap Class:

Constructor Methods

myMap()

The constructor method myMap() creates an object of the myMap class with a default input string "ThiS is An Examp!e InpUt StRinG." and an empty map.

myMap(String str)

The constructor method myMap(String str) creates an object of the myMap class with the specified input string and an empty map. If the specified input string is empty or null, it will be defaulted to "ThiS is An Examp!e InpUt StRinG."

Getter Methods

getMap()

The method getMap() returns the map object of this class as a LinkedHashMap.

size()

The method size() returns the size of the map.

getStr()

The method getStr() returns the input string of the myMap object.

keyArrayList()

The method keyArrayList() returns an ArrayList of String keys in the map.

get(String key)

The method get(String key) returns the info object of the given letter.

getCount(String key)

The method getCount(String key) returns the count of the given letter.

Other Methods

preProcess()

The method preProcess() takes in a String and preprocesses it by removing all non-alphabetic characters and converting all characters to lowercase. This method returns the preprocessed string.

buildMap()

The method buildMap() builds a map with letters of the input string, where each letter has an info object that keeps letter's count and words that contain that letter. The method splits the input string into an array of words, iterates through each word, and for each character in the word, it adds the word to the list associated with the character if the character exists in the map. If the character does not exist in the map, it creates a new info object with the word and adds the character and info object to the map. This method returns the string that was used to build the map.

put(String key, info value)

The method put(String key, info value) adds a new letter and its info object to the map. It returns the added info object.

clone()

The method clone() creates and returns a deep copy of the myMap object. It calls the superclass's clone() method to create a shallow copy, then creates a new LinkedHashMap for the map and copies the info objects for each letter in the map using the clone() method of the info class.

Display Method

show()

The method show() prints the map in the "Letter: - Count: - Words:" format using the toString() method of the info class.

Overall, the myMap class is a custom map structure that preprocesses a given string and builds a map based on it, where each letter has an info object that keeps letter's occur count and words that contain that letter. The class provides various methods to access and manipulate the map, as well as a display method to print the map in a user-friendly format.

info Class:

Constructor Methods

info(String word)

This constructor creates a new Info object with the given word. It initializes the words ArrayList with the given word and sets the count to 1.

info()

This constructor creates an empty Info object with zero count and an empty list of words.

Getter Methods

getCount()

The method returns the count of the letter occurrence.

Other Methods

push(String word)   
 This method adds a new word to the list of words and increments the count. It takes a string argument that represents the new word to be added.

clone()   
 This method creates and returns a new Info object that is a copy of this one. It overrides the clone() method from the Cloneable interface. The method creates a shallow copy of the object using the super.clone() method, which creates a new object with the same instance variables as the original object. It then creates a new ArrayList for the words and copies the words from the original ArrayList to the new ArrayList.

getCount()   
 The method that returns the count of the letter occurrence.

toString()   
 The method that returns a string representation of this Info object, including the count and the list of words. It overrides the toString() method from the Object class.

In summary, the Info class provides a container for storing information about words that contain a specific letter and their occurance count. It provides methods for adding words, creating a clone of the object, getting the count of the letter occurrence, and generating a string representation of the object.

mergeSort Class:

Constructor Methods

info(String word)

mergeSort(myMap originalMap, List<String> aux)  
 This constructs a mergeSort object with the given parameters. It takes the original map to be sorted, the sorted map, and the auxiliary list used for sorting. This constructor is currently commented out and not used in the class.

mergeSort(myMap originalMap)  
 This constructer constructs a mergeSort object with the given map. It takes the original map to be sorted as a parameter and initializes the auxiliary list with the keys of the original map.

mergeSort()  
 The constructer that constructs a mergeSort object with default parameters. It initializes the originalMap and sortedMap as empty myMap objects, and the auxiliary list as an empty ArrayList of Strings.

Methods  
 sortMap()

The sortMap() method sorts the original map and builds a new sorted map using the mergeSort algorithm based on the letters' occurrence count. It prints the original and sorted maps to the console. The method first prints the unsorted original map by show() method of the originalMap object. Then, it calls the sortAux() method with the starting and ending indexes of the auxiliary list. After sorting the auxiliary list, it builds the sorted map using the buildSortedMap() method. Finally, it prints the sorted map by show() method of the sortedMap object.

buildSortedMap()

The buildSortedMap() method builds the sorted map based on the sorted keys in the auxiliary list. The method first initializes the sortedMap object as a new myMap object. Then, it iterates over the auxiliary list and puts the corresponding key-value pairs in the sorted map using the put() method of the sortedMap object.

sortAux(int left, int right)

The sortAux() method sorts the auxiliary list using the merge sort algorithm. It takes the leftmost and rightmost indexes of the list to sort as parameters. The method first checks if the left index is greater than or equal to the right index. If true, it returns without performing any operation. Otherwise, it calculates the middle index and recursively calls the sortAux() method on the left and right sublists. After sorting the sublists, it merges the two sorted sublists into one sorted sublist using the merge() method.

merge(int left, int mid, int right)

The merge() method merges two sorted sublists of the auxiliary list into one sorted sublist. It takes the leftmost index of the left sublist, the rightmost index of the left sublist, and the rightmost index of the right sublist as parameters. The method first initializes three index variables, i, j, and k, to the starting index of the left sublist, the starting index of the right sublist, and the starting index of the merged list, respectively. It also initializes a new myMap object as the sortedMap. Then, it iterates over the left and right sublists and compares the values of the original map's key count for each corresponding element. It puts the elements into the sortedMap in ascending order based on the key count. After iterating over both sublists, it puts the remaining elements of the left and right sublists into the sortedMap. Finally, it updates the auxiliary list with the sorted keys of the sortedMap.

In conclusion, the mergeSort class provides a simple and efficient method for sorting a myMap object using the merge sort algorithm. It allows sorting the map based on the letters' occurrence count and building a new sorted map. The class can be useful in various applications that require sorting a myMap object