**Задача:** Реализовать коллекцию типа HashMap, хранящую данные в виде ключ-значение. Коллекция использует механизмы LinkedList и TreeMap для хранения и обработки данных.

**Алгоритм:**

**1.Создание класса Node<K,V>:**

\*Описываю элементы будущей коллекции в виде узлов (Node), которые будут хранить ключ и значение.

\*Реализую методы equals() и hashCode() для корректного сравнения элементов, а также метод toString() для удобного вывода.

**2.Создание класса MyHashMap<K,V>:**

\*Инициализирую коллекцию с заданной начальной емкостью (16) и коэффициентом загрузки (0.75).

\*Коллекция представлена массивом, где каждый элемент (bucket) может быть либо LinkedList, либо TreeMap в зависимости от количества элементов и необходимости балансировки.

**3.Метод put(K key, V value) для добавления элементов:**

\*Вычисляю индекс по хешу ключа.

\*Если в вычисленном индексе пусто, создаю LinkedList.

\*Если уже есть элементы, проверяю их тип:

-Если это LinkedList, добавляю новый элемент или, если размер списка превышает 8 элементов, преобразую его в TreeMap.

-Если это уже TreeMap, просто добавляю элемент.

**4.Метод get(K key) для получения значения по ключу:**

Определяю индекс по хешу ключа.

В зависимости от того, находится ли в этой ячейке LinkedList или TreeMap, ищу элемент и возвращаю его значение.

**5.Метод delete(K key) для удаления элемента:**

\*Определяю индекс по ключу и нахожу соответствующую коллекцию.

\*Если это LinkedList, удаляю элемент через метод remove().

\*Если это TreeMap, удаляю через метод remove() у дерева.

**6.Метод print() для вывода содержимого:**

\*Прохожу по всем ячейкам массива.

\*Если в ячейке находится LinkedList или TreeMap, вывожу их элементы.

**7.Тестирование работы MyHashMap:**

\*Создаю экземпляр коллекции и провожу несколько тестов:

1.Добавление нескольких элементов.

2.Поиск элементов с помощью метода get.

3.Удаление элементов и проверка их наличия после удаления.

4.Печать коллекции для проверки ее содержимого.

Node:

public class Node<K,V> {  
 K key;  
 V value;  
  
 public Node(K key, V value) {  
 this.key = key;  
 this.value = value;  
 }  
  
 // Переорпределяю equals & hashCode для корректного сравения и добавления в коллекцию  
 @Override  
 public boolean equals(Object o){  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Node<K,V> node = (Node<K,V>)o;  
 return key.equals(node.key) && value.equals(node.value);  
 }  
  
 @Override  
 public int hashCode(){  
 return Objects.*hash*(key,value);  
 }  
  
 @Override  
 public String toString() {  
 return "{" + key + " : " + value + '}';  
 }  
}

myHashMap:

public class MyHashMap<K,V> {  
 private int initialCapacity = 16;  
 private float loadFactor = 0.75f;  
 private ArrayList<Object> table;  
  
 public MyHashMap(){  
 table = new ArrayList<>(initialCapacity);  
 for (int i = 0; i < initialCapacity; i++){  
 table.add(null);  
 }  
 }  
  
  
 public int getIndex(K key){  
 return key.hashCode() % initialCapacity;  
  
 }  
  
 public void put(K key ,V value ){  
  
  
 int index = getIndex(key);  
 // Объявляю bucket для проверки текущей ячейки массива на принадлежность к опрделенному классу  
 Object bucket = table.get(index);  
  
 if (bucket == null){ /\* Если пусто создаю там LInkedList (через приведение типов создаю переменную,  
 чтоб можно было использовать методы LinkedList'а. То есть с table[index] обращаюсь как с LinkdeList, несмотря на то  
 что он все еще по факту Object. (приведение классов)\*/  
 table.set(index, new LinkedList<Node<K,V>>()); //  
 LinkedList<Node<K,V>> linkedList = (LinkedList<Node<K,V>>) table.get(index); // сам элемент, для работы от имени LL  
 linkedList.add(new Node<>(key, value));  
 }else if (bucket instanceof LinkedList){ // Если там уже есть LL то либо добавляю элемент, либо делаю из него TM  
 LinkedList<Node<K,V>> linkedList = (LinkedList<Node<K,V>>) table.get(index);  
 if (linkedList.size() < 8){  
 if (linkedList.stream().noneMatch(node -> node.key.equals(key))) {  
 linkedList.add(new Node<>(key, value));  
 }  
 } else{//Если это LL, и 8 элементов, то делаю из него TM  
  
 TreeMap<K,V> tree = new TreeMap<>();  
 for (Node<K,V> item: linkedList){ // Из LL добаляю элементы в treeMap  
 tree.put(item.key, item.value);  
 }  
 tree.put(key,value);// также помещаю новый элемент  
 table.set(index, tree);//В table помещаю TreeMap вместо LL  
 }  
 }else{  
 TreeMap<K,V> treeMap = (TreeMap<K,V>)table.get(index);// Просто добавляю новый элемент в TM  
 treeMap.put(key, value);  
 }  
 }  
  
 // получение значения по ключу  
 // комментировать не буду, тут все думаю понятно  
 public V get(K key){  
 int index = getIndex(key);  
 Object bucket = table.get(index);  
  
 if (bucket instanceof LinkedList){  
 LinkedList<Node<K,V>> list = (LinkedList<Node<K,V>>)bucket;  
 for (Node<K,V> item: list){  
 if (item.key.equals(key)) return item.value;  
 }  
 }else if (bucket instanceof TreeMap){  
 TreeMap<K,V> tree = (TreeMap<K,V>)bucket;  
 return tree.get(key);  
 }  
 return null;  
 }  
  
 // Удаление элемета из коллекции  
 public boolean delete(K key){  
 int index = getIndex(key);  
 Object bucket = table.get(index);  
  
 if (bucket instanceof LinkedList){  
 LinkedList<Node<K,V>> list = (LinkedList<Node<K,V>>)bucket;  
 for (Node<K,V> item: list){  
 if (item.key.equals(key)){  
 list.remove(item);  
 return true;  
 }  
 }  
 }else if (bucket instanceof TreeMap){  
 TreeMap<K,V> tree = (TreeMap<K,V>)bucket;  
 if (tree.remove(key) != null)  
 return true;  
 }  
 return false;  
 }  
  
 public void print(){  
 for (Object bucket: table){  
 if (bucket != null && bucket instanceof LinkedList){  
 LinkedList<Node<K,V>> list = (LinkedList<Node<K,V>>)bucket;  
 System.*out*.println(list);  
 }  
 if (bucket != null && bucket instanceof TreeMap){  
 TreeMap<K,V> tree = (TreeMap<K,V>)bucket;  
 System.*out*.println(tree);  
 }  
 }  
  
 }  
}

Main:

public class Main {  
 public static void main(String[] args) {  
  
 MyHashMap<String, Integer> myHashMap = new MyHashMap<String, Integer>();  
  
 // Тесты  
 // 1. Добавление элементов  
 myHashMap.put("One", 1);  
 myHashMap.put("Two", 2);  
 myHashMap.put("Three", 3);  
  
 // 2. Метод Get  
 Integer valueOne = myHashMap.get("One");  
 System.*out*.println("Value for 'One': " + valueOne);  
  
 // 3. Проверка для несуществующего элемента  
 Integer valueFour = myHashMap.get("Four");  
 System.*out*.println("Value for 'Four': " + valueFour);  
  
 // 4. Метод delete  
 boolean deleted = myHashMap.delete("Two");  
 System.*out*.println("Deleted 'Two': " + deleted);  
  
 // 5. Попытка получить удаленный элемент  
 Integer valueTwoAfterDeletion = myHashMap.get("Two");  
 System.*out*.println("Value for 'Two' after deletion: " + valueTwoAfterDeletion);  
  
 // 6. Удаления несуществующего элемента  
 boolean deletedNonExistent = myHashMap.delete("Four");  
 System.*out*.println("Deleted 'Four': " + deletedNonExistent);  
 myHashMap.print();  
 }  
}

Результат:

Value for 'One': 1

Value for 'Four': null

Deleted 'Two': true

Value for 'Two' after deletion: null

[{One : 1}, {Three : 3}]