```java
package myhashmap;

import java.util.ArrayList;

// Restrict the size of array to 4
public class AyushHashMap<K, V> {

   private ArrayList<HMNode>[] buckets = new ArrayList[4];
   private int size;

   public AyushHashMap() {
       initBrackets();
       size = 0;
   }

   private void initBrackets() {
       for (int i = 0; i < 4; i++) {
           buckets[i] = new ArrayList<>();
       }
   }

   public int getSize() {
       return size;
   }

   /*
   Put -? hm.put(key, value)
   0 -> 10, 20, 30, 98
   1 -> 54
   2 ->
   3 ->
   4 -> (89, true)
   5 -> 98

   0
   1
   2
   3
   4
   5
   6
   7
   8
   9
   10
   11

   put(89, true)
   idx of 89
```

```
 */
public void put(K key, V value) {
    int idx = hashCode(key);
    boolean isPresent = containsKey(key);
    int idxWithinBucket = getIdxOfElementInArrayList(key);
    if (isPresent) {
        // key is found, only update
        buckets[idx].get(idxWithinBucket).value = value;
    } else {
        // key is not found, insert new key-value pair
        HMNode newNode = new HMNode(key, value);
        buckets[idx].add(newNode);
        size++;

        // Check for rehash
        // Predefined threshold = 2.0
        double lambda = (size * 1.0) / buckets.length;
        if (lambda > 0.7) {
            rehash();
        }
    }
}

private void rehash() {
    ArrayList<HMNode>[] oldBucket = buckets;
    buckets = new ArrayList[oldBucket.length * 2];
    for (int i = 0; i < buckets.length; i++) {
        buckets[i] = new ArrayList<>();
    }
    // Redistribution
    for (ArrayList<HMNode> bucket: oldBucket) {
        for (HMNode node: bucket) {
            put((K) node.key, (V) node.value);
        }
    }
}

/*
Hash function
 */
public int hashCode(K key) {
    int hashCode = key.hashCode();
    /*
    It can be any value, so please take this value inside the array
    indices
     */
    int idx = Math.abs(hashCode) % buckets.length;
    return idx;
}
```

```java
public boolean containsKey(K key) {
    int idx = hashCode(key);
    ArrayList<HMNode> elements = buckets[idx];
    for (HMNode node: elements) {
        if (node.key.equals(key)) {
            return true;
        }
    }
    return false;
}

private int getIdxOfElementInArrayList(K key) {
    int idx = hashCode(key);
    ArrayList<HMNode> elements = buckets[idx];
    for (int i = 0; i < elements.size(); i++) {
        HMNode node = elements.get(i);
        if (node.key.equals(key)) {
            return i;
        }
    }
    return -1;
}

// Get - get the value of the key
/*
0 -> 10, 20, 30, 98
1 -> 54
2 ->
3 ->
4 -> (89, true)
5 -> 98
 */
public V get(K key) {
    if (!containsKey(key)) {
        return null;
    }
    int idx = hashCode(key);
    int idxWithinBucket = getIdxOfElementInArrayList(key);
    // Never happen
    if (idxWithinBucket == -1) {
        return null;
    }
    return (V) buckets[idx].get(idxWithinBucket).value;
}

// Remove method

// keySet -> returns all the keys
```

```
    public ArrayList<K> keySet() {
        ArrayList<K> keys = new ArrayList<>();
        for (ArrayList<HMNode> bucket: buckets) {
            for (HMNode node: bucket) {
                keys.add((K) node.key);
            }
        }
        return keys;
    }

}
```