```
package HW4.HashTable;

import java.util.LinkedList;

/**
 *
 * @author rand
 */
public class CustomHashTable<K extends Integer, T> {

    private class Node<T> {

        T val;
        int key;

        public Node(int key, T val) {
            this.val = val;
            this.key = key;
        }

    }

    private LinkedList<Node<T>>[] arr;
    private final float LOAD_FACTOR = 0.75f;
    private int capacity;
    private int size = 0;

    public CustomHashTable() {
        capacity = 15;
        arr = new LinkedList[capacity];

    }

    public CustomHashTable(int capacity) {
        this.capacity = capacity;
        arr = new LinkedList[capacity];
//        new Hashtable<>().pu
    }

    public T put(int key, T val) {
        // invalid key or val, then do not store
        if (val == null || key < 0) {
            return null;
        }

        int index = index(key);
        Node<T> node = new Node(key, val);

        if (arr[index] == null) {
            arr[index] = new LinkedList<>();
            arr[index].push(node);
```

```
            size++;
            expand();
        } else {
            // if key already exits, do update
            for (Node<T> n : arr[index]) {
                if (n.key == key) {
                    n.val = val;
                }
            }
        }
        return val;
    }

    public T remove(int key) {
        int index = index(key);
        // invalid key
        if (key < 0 || arr[index] == null) {
            return null;
        }

        T val = null;
        // if key already exits, do update
        for (Node<T> n : arr[index]) {
            if (n.key == key) {
                val = n.val;
                arr[index].remove(n);
                return val;
            }
        }

        return val;
    }

    public T get(int key) {
        int index = index(key);
        if (key < 0 || arr[index] == null) {
            return null;
        }
        LinkedList<Node<T>> list = arr[index];
        for (Node<T> node : list) {
            if (node.key == key) {
                return node.val;
            }
        }
        return null;
    }

    private int index(int key) {
        return Integer.hashCode(key) % capacity;
    }

    private void expand() {
```

```
        if (LOAD_FACTOR < (capacity / size)) {
            capacity = capacity * 2;
            LinkedList<Node<T>>[] arrCopy = new LinkedList[capacity];
            for (int i = 0; i < arr.length; i++) {
                arrCopy[i] = arr[i];
            }
            arr = arrCopy;
        }
    }

    public int getSize() {
        return size;
    }

    @Override
    public String toString() {

        StringBuilder builder = new StringBuilder();

        builder.append("{");
        for (LinkedList<Node<T>> list : arr) {
            if (list == null) {
                continue;
            }
            for (Node<T> elm : list) {
                builder.append(elm.val + ", ");
            }
        }
        builder.append("}");

        return builder.toString();
    }

}
```