South East Technological University
Department of Computing and Mathematics
official record of school work

Department of Computing and Mathematics
CA Work

Programme that you are enrolled on:	CA1
Programme Module	Data Structures and Algorithms 1
Title of Work Submitted (e.g Lab Report 1)	Project Report 1
Lecturers	Guoqing Lu
ID Number	202383930005,202383930004,202383930036
Class Group (number)	6

Filename: e.g GroupNumber_DS&A1_Project_1.pdf	6_CA1_Project_1.pdf
Acknowledgements (if you collaborated as a team then acknowledge colleagues)	Fang jun, Zang Ziye, Du Xilei

Declaration

I hereby declare that this is my original work produced without the help of any third party unless where referenced within this report.

Student:

Date and Time of Submission:

Table of Contents

1 Introduction 3

2 Design Record 3

3 Implementation (including codes) 8

4 Results and Conclusions (Results analysis and discussion) 16

5 References 18

1 Introduction

The objective of this CA exercise is to create a “jewellery store management system” in Java that makes heavy use of custom-built internal data structures. The system should allow the user to manage a jewellery store that consists of multiple display cases. Every display case contains multiple display trays. Every tray can contain multiple items of jewellery (e.g. rings, watches, necklaces, etc.). Every item of jewellery consists of one or more materials/components (e.g. gold, platinum, diamonds, etc.), and these materials/components will vary in quantity (by a suitable unit of weight e.g. grams or carats) and quality (e.g. karat for gold, clarity for diamonds, etc.) for different items of jewellery. Therefore, in summary, an item of jewellery is comprised of one-or-more materials/components of varying quantity and quality; an item of jewellery has to be stored on a display tray; a display tray must be stored in a display case.

2 Design Record

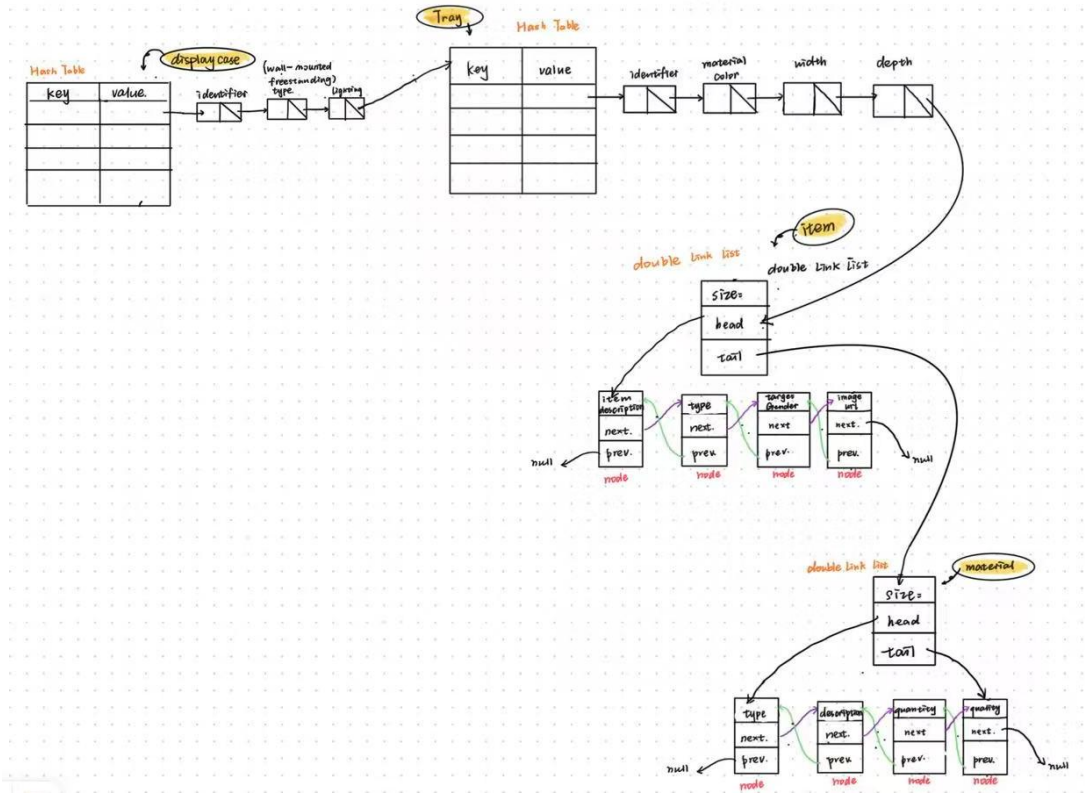
You are supposed to include all high level designs in this section, such as data structure, add/remove method, smart add, search and sort method, iterator/iterable, comparable/comparator, Junit etc.

- Data structure:

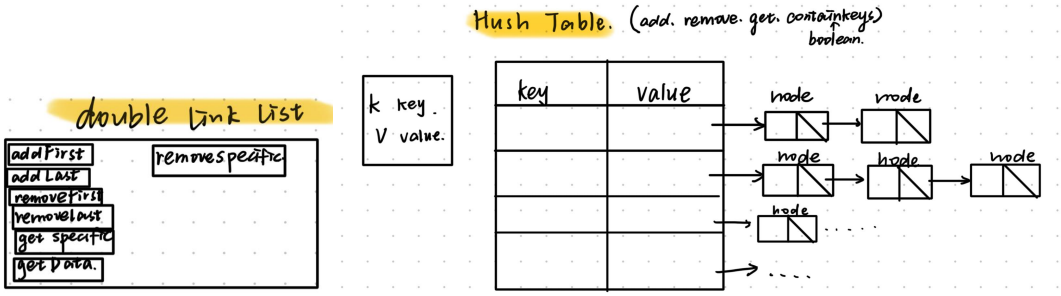
(1) Version 1

We plan to use class to create the case, tray, jewellery and material object, then use inheritance to connect them. But this structure can’t adjust by itself,(e.g. if the same id occurs, the shop can’t add successfully), so we didn’t use it.

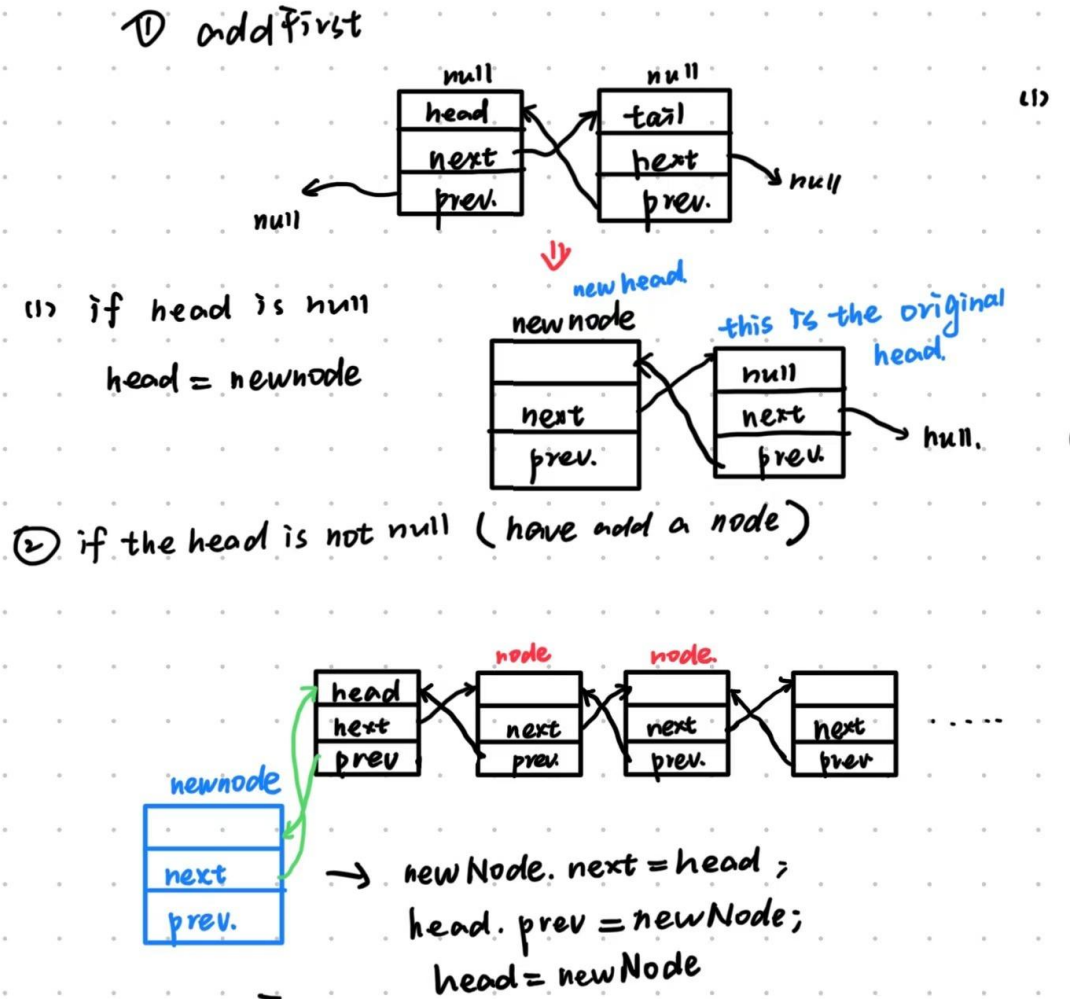
(2) Version2



We use DoubleLinkList and HashTable to connect the entity. As the picture showing, between the case and trays is a linklist store with the information of case , and the case and trays are both hashTable (in order to find them by their id). As for the jewellery and material, their structure are doubleLinkList, to store them one by one.

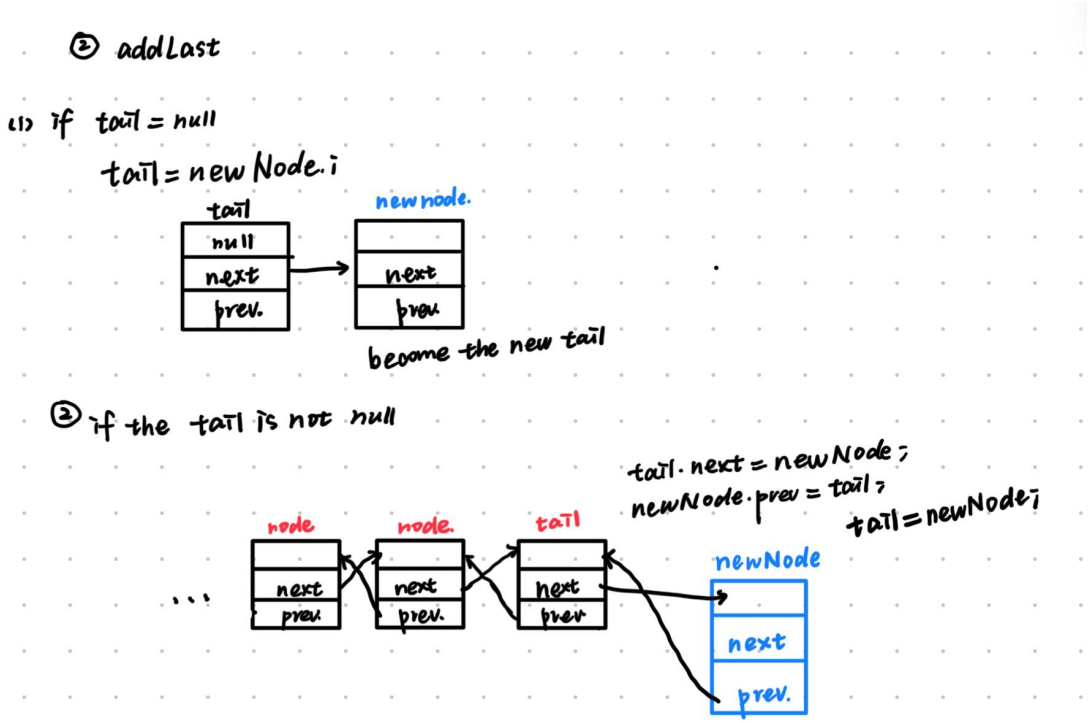


- Add method



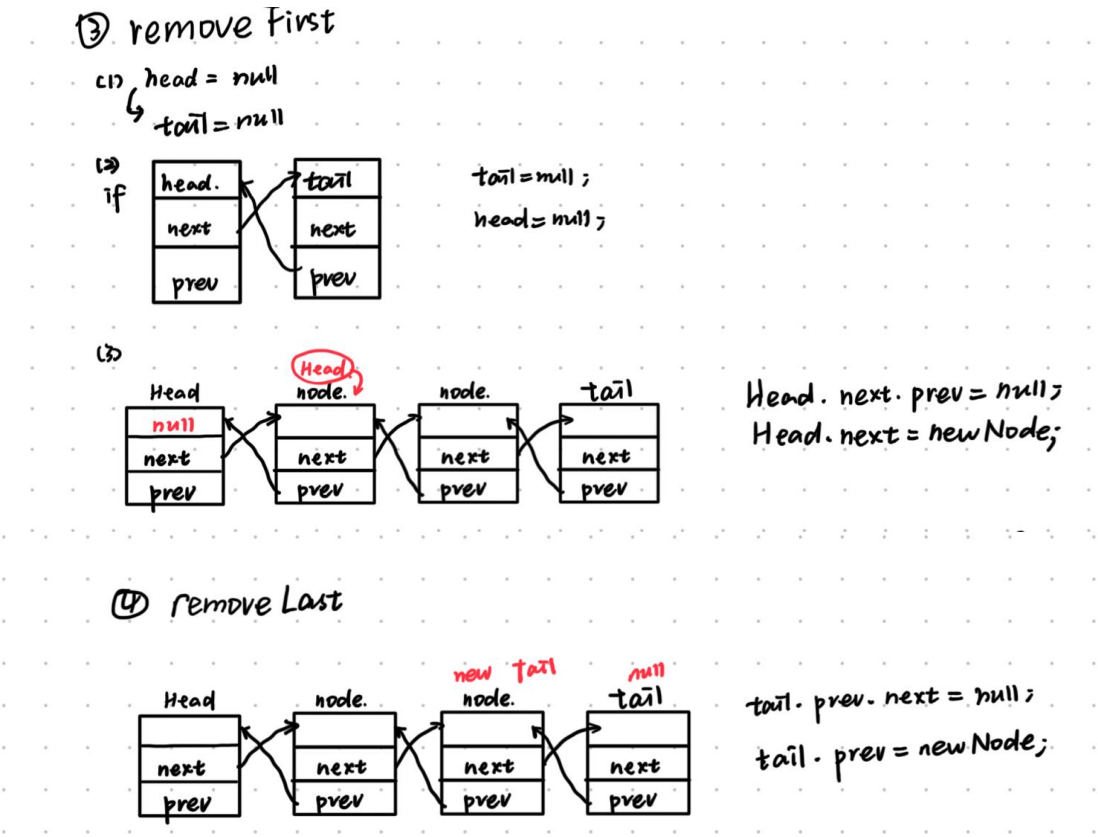
Version 1
We use head = new Node at first when design the add last method , but this will make mistake when search the last node.

Version 2
We change the new Node to tail.

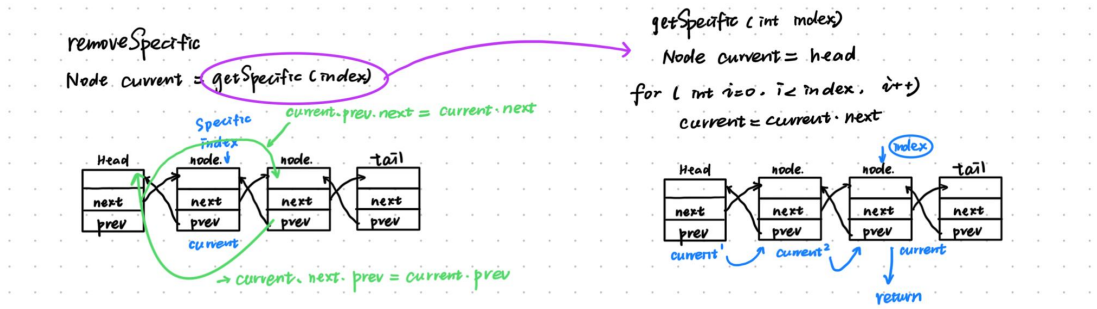


- Remove method

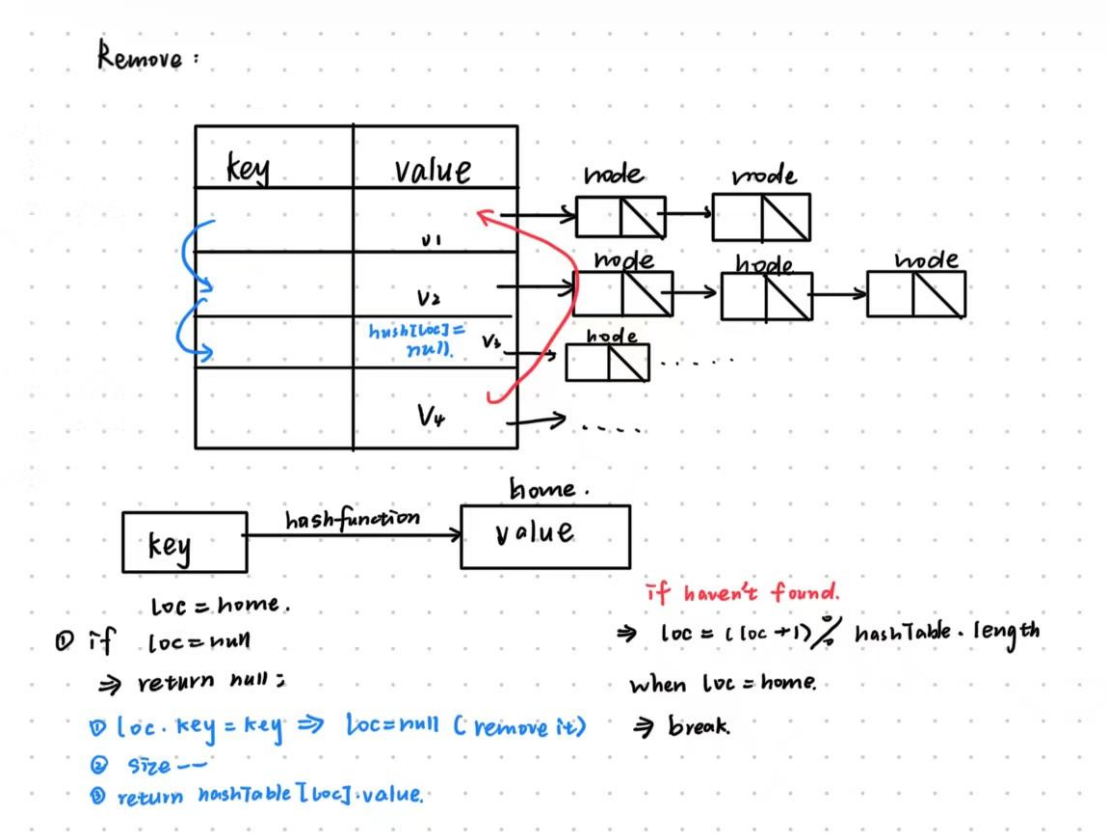
We let the tail Node to its pre Node,and make the new tail.next point to null ,skip the Last/first node to remove them.



We use the linear search to get the specific node location ,then skip the specific node to remove it.



We create a remove method in the hash table.



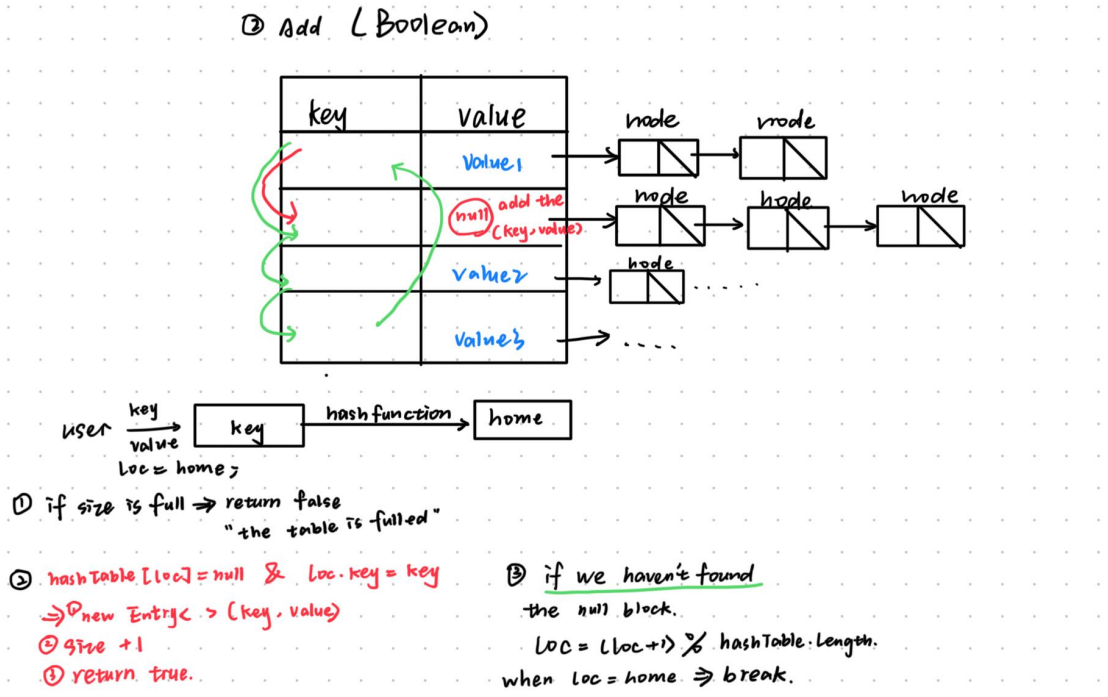
- Smart add

Version 1

We want to use index to find specific location, but it can't be add if the index has be exit.

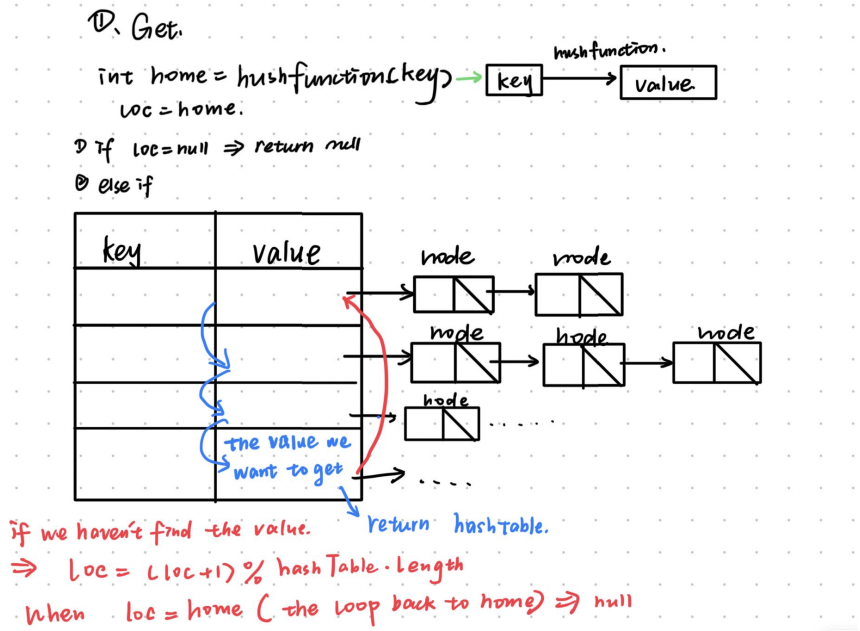
Version 2

We use hash table to achieve the smart add.Both of the case and trays have id ,so we use hash Function to the id to get specific key, according to the key ,we can deal with specific add and if two cases/trays have the same id, it can change the location by itself to find a new place to add, which achieve the smart add.



- search and sort method

We use hash table to search entity by their id, using hash function to calculate the key.



- iterator/iterable,

Version 1

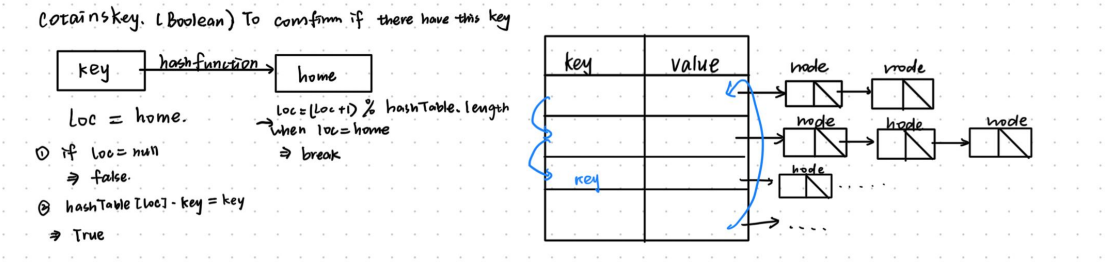
We use for each loop to go through the table and want to print the value according , but we don't know what key is , this design can't be achieve.

Version 2

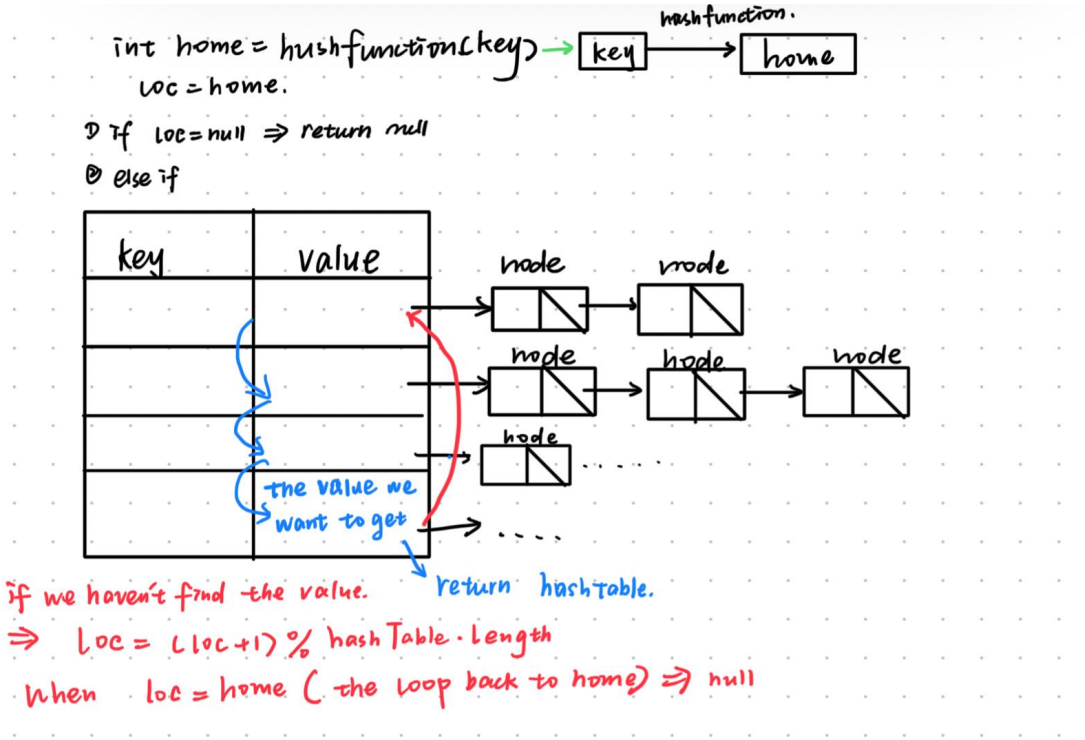
We use to string method , the trays, jewellery, material class all contain string toString, but only jewellery and material can be used.

Version 3
We use hashTable method to printTable, firstly ,we print all the keys in the table and save them to a array , then use the while loop to iterator the table.

- Containkey method



- Get method



3 Implementation (including codes)

- Data structure

The Double Link List

```
public class Node{
    public T Data;
    public Node next;
    public Node prev;
    public Node(T Data){
        this.Data = Data;
    }
```

```
}
private Node head;
private Node tail;
private int size;
public DoublyLinkedList(){
    this.head = null;
    this.tail = null;
    size = 0;
}
```

The hash table

```
// inner class Hash Table
public static class Entry<K, V> {
    public K key;
    public V value;
    public Entry<K, V> next;// For handling collisions (chaining)

    public Entry(K key, V value) {
        this.key = key;
        this.value = value;
        this.next = null;
    }
    public void setKey(){
        this.key = key;
    }
    public K getKey(){
        return key;
    }

}
```

●Add method

```
//add method
public void addFirst(T data){
    Node newNode = new Node(data);
    if (head == null){
        head = newNode;//the head is the first
    }else {
        newNode.next = head;
        head.prev = newNode;//change the newNode to head
    }
```

```
    head = newNode;//the newNode become the new head
    size ++;
}
public void addLast(T data){
    Node newNode = new Node(data);
    if (tail == null){
        tail = newNode;//the last is the tail
    }
    else {
        tail.next = newNode;
        newNode.prev = tail;//change tailNode to tailPrevNode
    }
    tail = newNode;//the newNode become the new tail
    size ++;
}
```

●Remove method

```
//remove method in the doubleLinkList
public void removeFirst(){
    if (head==null){
        tail=null;//didn't find
    }else if (tail.prev==head.next){
        head = null;
        tail = null;
        size --;//only one
    }else {
        head.next.prev = null;
        head = head.next;//skip the node to remove it
        size--;
    }
}
public void removeLast(){
    if (head == null){
        tail=null;//didn't find
    }else if (tail.prev == head.next){
        head = null;
        tail = null;
        size --;//remove the only one
    }else{
        tail.prev.next = null;
```

```
            tail = tail.prev;
            size --;// let the tailNode to its prevNode, and make the new tail.next point
to null
        }
    }
    // remove method in Hashtable
    public V remove(K key) {
        int home = hashFunction(key),loc = home;
        do {
            if (hashTable[loc] == null) {
                return null; //can't find
            }
            if (hashTable[loc].key.equals(key)) {
                hashTable[loc] = null; //remove
                size--;
                return  hashTable[loc].value;
            }
            loc = (loc + 1) % hashTable.length;
        } while (loc != home);

        return null; // didn't have
    }
```

●Specific method

```
    // specific method
    public Node getSpecific(int index){
        Node current = head;
        if (head == null) {
            throw new IllegalStateException("The list is empty.");
        }
        for(int i = 0 ;i < index ; i++){
            current = current.next;
        }//use the index to get the node
        return current;
    }

    public T getData(int index) {
        if (index < size){
            return null;
```

```
    }else {
        return getSpecific(index).Data;
    }
}

public void removeSpecific(int index){
    Node current = getSpecific(index);
    if (current == null){
        return;
    }else {
        current.prev.next = current.next;
        current.next.prev = current.prev;//skip the node to remove the node
    }
    size --;
}
```

●Smart add

```
// add method in hash table
public boolean put(K key, V value) {
    int home = hashFunction(key),loc = home;
    if (size >= hashTable.length) {
        System.out.println("the table has fulled");
        return false;
    }
    do {
        //if the table has space, add the linked list to the hashtable
        if (hashTable[loc] == null || hashTable[loc].key.equals(key)) {
            hashTable[loc] = new Entry<>(key, value);
            size++;
            return true;
        } else {
        //if not, change the original location
            loc = (loc + 1) % hashTable.length;
        }
    } while (loc != home); // if loc = home ,indicate that the table has no space

    return false;
}
```

●search and sort method

```
// get method
public V get(K key) {
    int home = hashFunction(key),loc = home;
    do {
        if (hashTable[loc] == null) {
            return null;//can't find
        }else if (hashTable[loc].key.equals(key)) {
            return hashTable[loc].value;
        }
        loc = (loc + 1) % hashTable.length;
    } while (loc != home);
    return null;
}
// identify if key is contained
public boolean containsKey(K key) {
    int home = hashFunction(key),loc = home;
    do {
        if (hashTable[loc] == null) {
            return false; // didn't find
        }
        if (hashTable[loc].key.equals(key)) {
            return true; // find
        }
        loc = (loc + 1) % hashTable.length;
    } while (loc != home);
    return false; // didn't find
}
```

● iterator/iterable method

```
public void iterateForward(){
    while (head != null){
        System.out.print(head.Data);
        head = head.next;// from head to tail
    }
    System.out.println("null");
}
```

```
public void iterateBackward(){
    while (tail != null){
        System.out.print(tail.Data);
        tail = tail.prev;//from tail to head
    }
    System.out.println("null");
}
//print method
public void printCase() {
    System.out.print("No." + id + "    " + type + "    ");
    if (lit) {
        System.out.println("LIT");
    } else {
        System.out.println("UNLIT");
    }
    System.out.println(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>");
    Integer[] newKeysArray = trays.keys();
    int i = 0;
    while ( i < newKeysArray.length) {
        DisplayTray current = trays.get(newKeysArray[i]);
        System.out.println(current.printTrays());
        i++;
    }

}
```

●Junit

```
@Test
public void testPutAndGet() {
    Hash<String, Integer> hashTable = new Hash<>();

    hashTable.put("A", 1);
    hashTable.put("B", 2);
    hashTable.put("C", 3);

    assertEquals(1, hashTable.get("A"));
    assertEquals(2, hashTable.get("B"));
    assertEquals(3, hashTable.get("C"));
}
```

```
@Test
public void testUpdateValue() {
    Hash<String, Integer> hashTable = new Hash<>();

    hashTable.put("A", 1);
    hashTable.put("A", 10);

    assertEquals(10, hashTable.get("A"));
}

@Test
public void testGetNonExistentKey() {
    Hash<String, Integer> hashTable = new Hash<>();

    assertNull(hashTable.get("nonexistent"));
}

DoublyLinkedList<String> shop = new DoublyLinkedList<>();

@Test // test the add methods
public void add(){
    shop.addFirst("A");
    shop.addLast("B");
}


@Test // test the get method
public void get(){
    shop.addFirst("A");
    shop.addLast("B");
    System.out.println("Element at index 1: " + shop.getData(0));
}

@Test  // test the remove methods
public void remove(){
    shop.addFirst("A");
    shop.addLast("B");
    shop.removeFirst();
    shop.removeLast();
```

```
}


@Test// test the specific methods
public void specific(){
    shop.addFirst("D");
    shop.addLast("E");
    shop.removeSpecific(1);

}
```

4 Results and Conclusions (Results *analysis and discussion*)

```
public class Main {
    public static void main(String[] args) {
        // Create a jewelry store
        JewelleryShop store = new JewelleryShop();

        // Create a showcase
        DisplayCase displayCase1 = new DisplayCase("1", "glass", true);
        DisplayCase displayCase2 = new DisplayCase("2", "steel", false);

        // Create two display trays
        DisplayTray tray1 = new DisplayTray("A1", "Red Velvet", 20, 20);
        DisplayTray tray2 = new DisplayTray("B2", "Blue Velvet", 25, 15);

        // Create some jewelry items
        JewelleryItem watch = new JewelleryItem(“watch", "Watch", "male", "",
5600.0);
        JewelleryItem necklace = new JewelleryItem("Gold Necklace", "Necklace",
"Female", "", 148.0);

        //Create some material
        Material material1 = new Material("glod", "99.99%", 2, 11.11);

        //Add material to jewelry
        watch.addFirstMaterial(material1);

        // Add jewelry items to tray
```

```
        tray1.addFirstItem(watch);
        tray2.addLastItem(necklace);

        // Add trays to display cabinets
        displayCase1.addTray(tray1);
        displayCase2.addTray(tray2);

        // Add display cabinet to jewelry store
        store.addDisplayCase(displayCase1);

        // Print showcase information
        DisplayCase retrievedCase = store.getDisplayCase("1");
        if (retrievedCase != null) {
            retrievedCase.printCase();

        } else {
            System.out.println("Display case not found.");
        }
    }
```

```
C:\Users\方珺\.jdks\openjdk-21\bin\java.exe ...
<<< No.1   Glass   LIT
Number of Tray: 2
[A1]   Inlay_material_colour: Red Velvet   20.0x20.0
This tray has 1 pieces of jewllery.
>>>>>>>>>>
#1 'watch'
-Type: watch
-Target Gender: male
-Image URL:
-Retail price: 5600.0$
-The jewellery is composed of 0 materials:

[A1]   Inlay_material_colour: Red Velvet   20.0x20.0
This tray has 1 pieces of jewllery.
>>>>>>>>>>
#1 'watch'
-Type: watch
-Target Gender: male
-Image URL:
-Retail price: 5600.0$
-The jewellery is composed of 0 materials:
```

5 References

The method public K[] keys() is use by AI wenxinyiyan, to print a array with all keys in the hash table.