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:	Data Structure & Algorithm
Programme Module	Data Structures and Algorithms 1
Title of Work Submitted (e.g Lab Report 1)	Project Report 1
Lecturers	Guoqing Lu
ID Number	202283890007 202283890008 202283890009 202283890014
Class Group (number)	24

Filename: e.g GroupNumber_DS&A1_Project_1.pdf	Group24_DS&A1_Project_1.pdf
Acknowledgements (if you collaborated as a team then acknowledge colleagues)	I would like to express my gratitude to all my team members and I for our full cooperation. With everyone in this group, we are able to complete this project.

Declaration

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

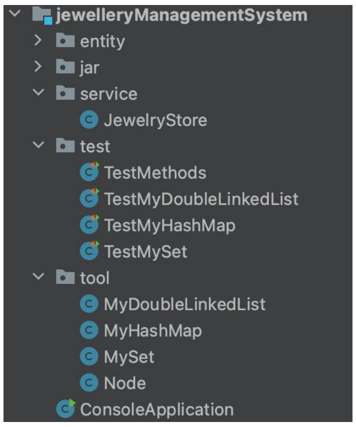
Student: Dongxu Xia Jinru Yao Yao Yao Yuchen Zhang

Date and Time of Submission: October 29, 2023

Table of Contents

1 Introduction

To the structure of the program, there are 4 packages to support our system.The Figure1 shown below display the whole structure. The entity package stores the objects which we need in the program. In the service package, there is jewelry store, which provides all the methods to implement. The tool bag contains two tools, one is hash map and the other is Double Linked List.

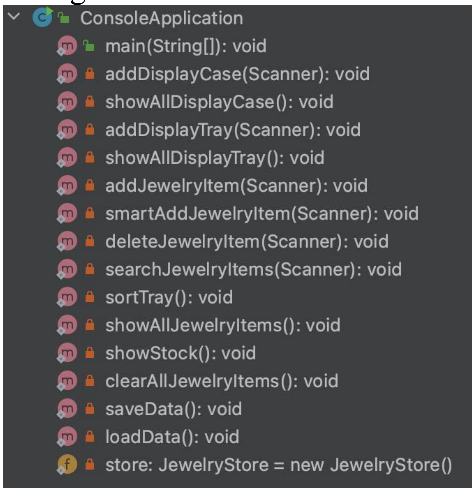


```
jewelleryManagementSystem
  entity
  jar
  service
    JewelryStore
  test
    TestMethods
    TestMyDoubleLinkedList
    TestMyHashMap
    TestMySet
  tool
    MyDoubleLinkedList
    MyHashMap
    MySet
    Node
  ConsoleApplication
```

Figure 1

Jewelry store management system is an application based on console, using a menu to list all the operation options, by creating multiple classes to represent different concepts and entities, to achieve the management and display of jewelry items, display cases and display trays.

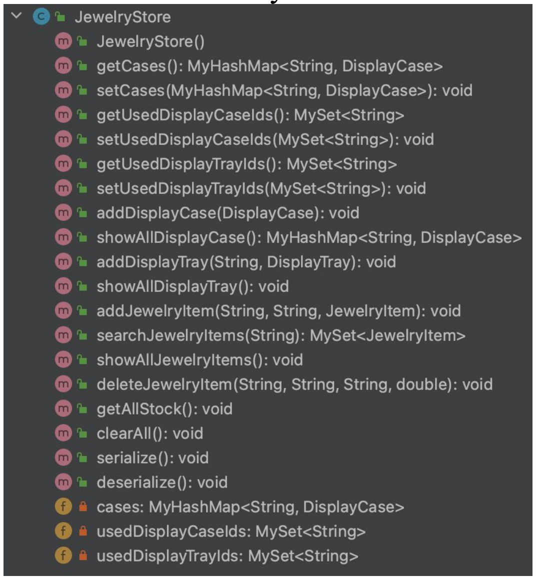
In the ConsoleApplication, the user can select different actions through the menu, such as adding cases, trays, and jewelry items, displaying all jewelry items, deleting specified cases, trays, and jewelry items, clearing all data, and calculating the total value of jewelry items. The Figure 2 shown below:



```
ConsoleApplication
  main(String[]): void
  addDisplayCase(Scanner): void
  showAllDisplayCase(): void
  addDisplayTray(Scanner): void
  showAllDisplayTray(): void
  addJewelryItem(Scanner): void
  smartAddJewelryItem(Scanner): void
  deleteJewelryItem(Scanner): void
  searchJewelryItems(Scanner): void
  sortTray(): void
  showAllJewelryItems(): void
  showStock(): void
  clearAllJewelryItems(): void
  saveData(): void
  loadData(): void
  store: JewelryStore = new JewelryStore()
```

Figure 2

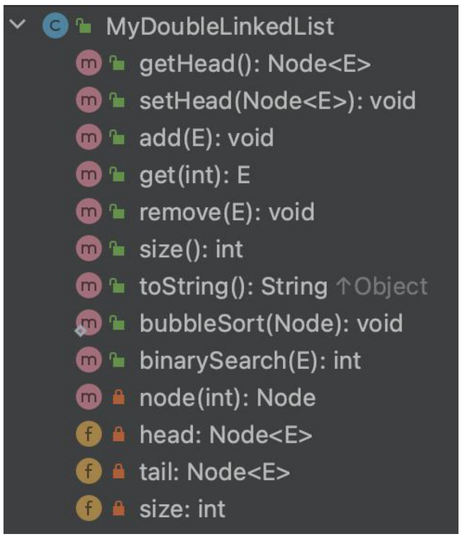
In the JewelryStore class, we have 'add()', 'show()', 'delete()', 'clear()', 'search()', 'calculate()',and 'save and load()'methods that we can call from our ConsoleApplication. These methods take the information entered by the user, create objects, add them to the DoubleLinkedList or HashMap, and retrieve them from the DoubleLinkedList or HashMap to display to the user. At the same time, we also implemented the method to verify the uniqueness of the ID of the display tray, display case and jewelry item to ensure the accuracy of the data.



```
JewelryStore
    JewelryStore()
    getCases(): MyHashMap<String, DisplayCase>
    setCases(MyHashMap<String, DisplayCase>): void
    getUsedDisplayCaseIds(): MySet<String>
    setUsedDisplayCaseIds(MySet<String>): void
    getUsedDisplayTrayIds(): MySet<String>
    setUsedDisplayTrayIds(MySet<String>): void
    addDisplayCase(DisplayCase): void
    showAllDisplayCase(): MyHashMap<String, DisplayCase>
    addDisplayTray(String, DisplayTray): void
    showAllDisplayTray(): void
    addJewelryItem(String, String, JewelryItem): void
    searchJewelryItems(String): MySet<JewelryItem>
    showAllJewelryItems(): void
    deleteJewelryItem(String, String, String, double): void
    getAllStock(): void
    clearAll(): void
    serialize(): void
    deserialize(): void
    cases: MyHashMap<String, DisplayCase>
    usedDisplayCaseIds: MySet<String>
    usedDisplayTrayIds: MySet<String>
```

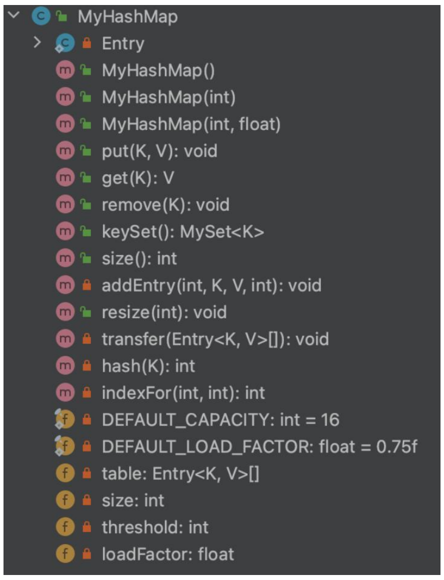
Figure 3

To store the data efficiently, we mainly used DoubleLinkedList and HashMap. DoubleLinkedList is used to store display trays and jewelry items, keeping the order in which they were added. The HashMap, on the other hand, is used to store the showcase, with the ID of the showcase as the key of the key value pair and the display case object as the value. In this way, the display case can be easily found and operated by the ID of the display case.



```
MyDoubleLinkedList
    getHead(): Node<E>
    setHead(Node<E>): void
    add(E): void
    get(int): E
    remove(E): void
    size(): int
    toString(): String ↑Object
    bubbleSort(Node): void
    binarySearch(E): int
    node(int): Node
    head: Node<E>
    tail: Node<E>
    size: int
```

Figure 4



```
MyHashMap
  Entry
  MyHashMap()
  MyHashMap(int)
  MyHashMap(int, float)
  put(K, V): void
  get(K): V
  remove(K): void
  keySet(): MySet<K>
  size(): int
  addEntry(int, K, V, int): void
  resize(int): void
  transfer(Entry<K, V>[]): void
  hash(K): int
  indexFor(int, int): int
  DEFAULT_CAPACITY: int = 16
  DEFAULT_LOAD_FACTOR: float = 0.75f
  table: Entry<K, V>[]
  size: int
  threshold: int
  loadFactor: float
```

Figure 5

Through this design, the jewelry store management system provides a convenient and efficient way to manage and display jewelry items, to help jewelry stores improve operational efficiency and service quality. Users can operate through the console interface to add, search and display jewelry items, while the system also provides data verification and calculation functions to ensure the accuracy of data and facilitate business decisions.

2 Design Record

2.1 Data Structure

In the Jewelry System, we use Double Linked List as our data structure. The linked list can make it easier to insert items, which can shorten programme process and improve programme efficiency.

The Figure 6 below is the main structure of the store system. The display cases are stored in the hash map(it will be explained in the following article), then display trays are the attribute of the case. Each case stores trays with double linked list. Tail insertion is used to insert the trays so that trays can be kept sequenced. Similarly, item is also the attribute of tray, and the component is the attribute of item.

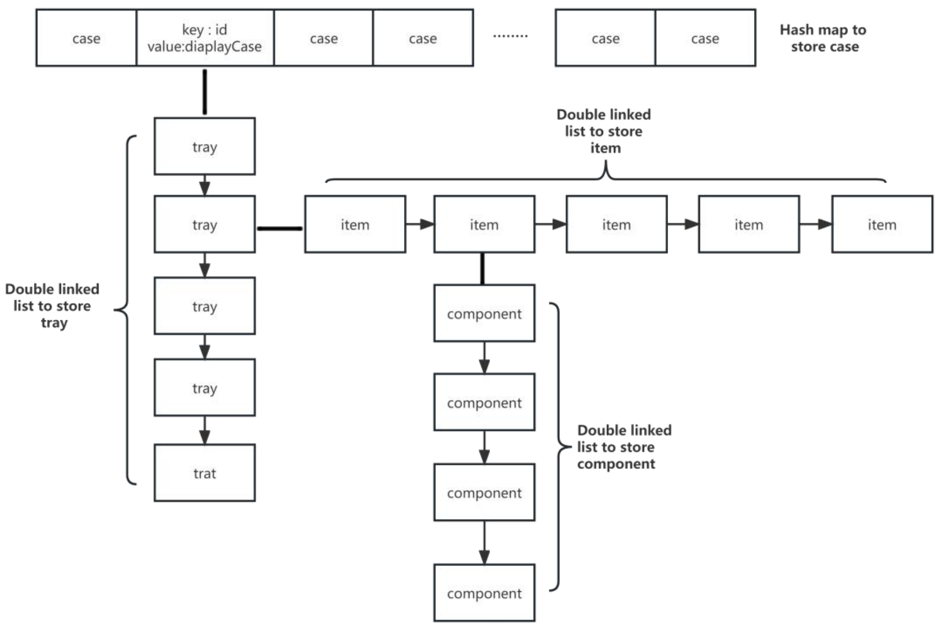


Figure 6

The reason why the Double Linked List is chosen is that, Linked List can grow on demand, and it has no slot to be maintained, and for this reason, Linked List compared to Array List is much more convenient to be maintained in the future. Moreover, Double Linked List will not waste the memory of the computer due to presetting capacity size. Compared to Simple Linked List, Double Linked List can be traversed starting from either direction.

Commonly methods are designed, for instance, add, get and remove function, which make it easier to manage the elements in the system. As the attribution of the linked list is double, not only the head node, but also the tail node is set, and this allow the system to traverse the list at both sides. As Figure 7 followed, we use the simple way to add elements to the Linked List.

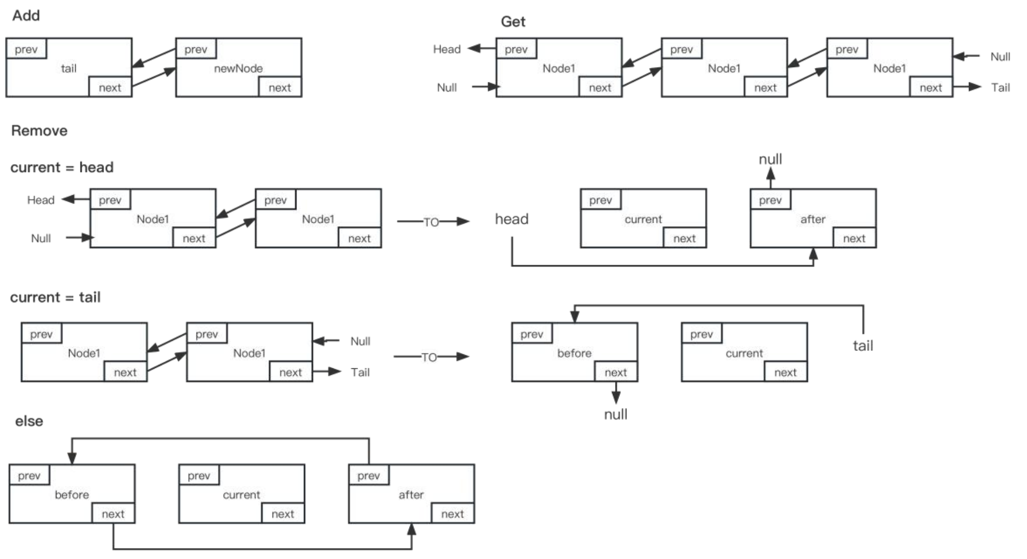


Figure 7

In our binary search, we first define the lower and upper bounds of the search range. Then enter the main loop, calculate the middle position in each loop, and obtain the element values of the middle position. If the element value in the middle position is less than the target value, narrow the search range to the right half; If the element value in the middle position is greater than the target value, narrow the search range to the left half; If the element value in the middle position is equal to the target value, then the middle position is returned. When the search range is empty, it indicates that the target value was not found and returns -1.

2.2 Algorithm

In our system, we use hash map to store display case. In our hash function, first converts the hash code of the input value into binary. Then perform a bitwise sum operation on the binary number and the hash table length minus 1. Finally, use the operation result as the index of the array and store the input value at that index. This ensures that the obtained index can contain all positions of the hash table, but this also results in our initial capacity being only to the power of 2, and in subsequent expansion, we can only double the capacity each time. The process is shown as Figure 8:

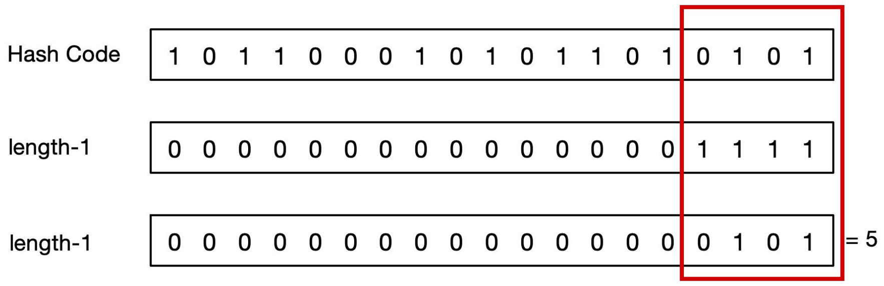


Figure 8

What’s more, to deal with hash collision, we use Separate Chaining. If different elements are calculated to the same key value, we connect them together using a linked list, so that Separate Chaining allows keys to share the same index, improving the flexibility of hash maps, and optimizing insertion and deletion operations by inserting elements in the header, improving efficiency. The process Figure 9 is shown below:

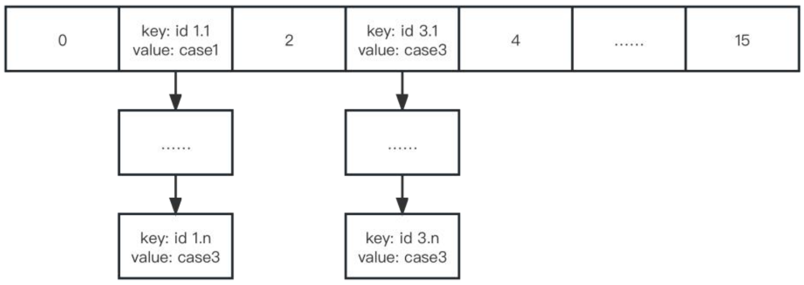


Figure 9

From Figure 10, we can see the main process of our hash map. If we need to add a display case into the hash table, we first determine whether there is a hash table, if so, we use hash code method to calculate the index value, if not, we need to create a new hash table.

Next, we need to determine whether there are elements in the position of table [i]. If so, we will match the key values to be inserted with the existing case. If they are the same, we will overwrite the element. If they are not the same, we will traverse the linked list contained in that position and continue matching. If there is still the same key value, we will overwrite it. If there is no identical key value, insert the case into the head of the linked list (ps: head insertion method places the element at the beginning of the linked list, with the oldest element at the end of the linked list). If there is no case in the table[i] position, what we need to do is that we insert the case directly into the table [i] position.

After inserting the case, we need to increase the size of the entire hash map by one, and finally compare the resulting hash map size with the maximum capacity. If the maximum capacity is reached, we will expand the hash map and end the process. If the maximum capacity is not reached, then end the process directly.

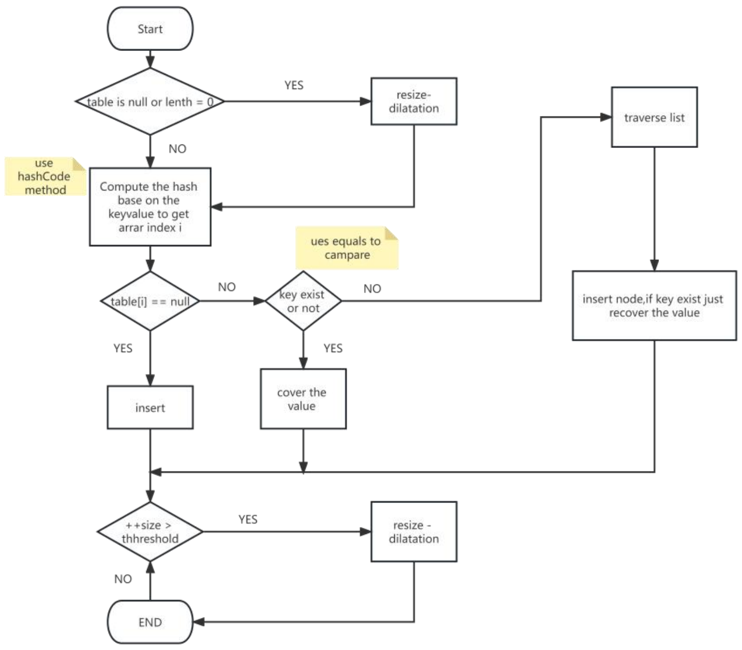


Figure 10

2.3 Function

2.3.1 Add Function

‘addDisplayCase’ adds the display case object to the store by first asking the user to enter the ID of the display case and then doing a format check to ensure that the ID is a combination of uppercase letters and numbers. If the ID meets the requirements, it will ask the user to enter the type of display case and whether the lights are on, and then create a new display case object and add it to the memory.

‘addDisplayTray’ adds a display tray to the display case, finds the corresponding display case object by the ID of the display case, then checks whether the ID of the display tray already exists, and finally adds the display tray to the tray list of the display case and updates the set of used display tray IDs.

‘addJewelryItem’ adds a jewelry item to the display tray by using the ID of the display case and the ID of the tray to find the corresponding display case object and tray object, and then adds the jewelry item to the list of jewelry items in the tray.

2.3.2 Smart Add

In Smart Add, we have designed a method of adding a comparison similarity to the jewelry category to calculate the score for inserting jewelry. After entering a description of the jewelry, calculate the score based on one digit of each category, gender, price, and material. If they agree, they will get 9 points. For example, if the category is consistent, the gender is inconsistent, the price is consistent, and there are three identical materials, the score is 9, 0, 9, and 3. Then traverse all the jewelry, take the one with the highest score, and place it on the corresponding plate. If there are no similar jewelry, place the jewelry in the last display tray.

2.3.3 Show Function

‘showAllDisplayCase’ prints all the cases and returns the case hash table. It achieves the function of displaying all display cases by traversing the key set of the display case, obtaining each display case object and printing its string representation.

‘showAllDisplayTray’ prints information about all the display trays. By traversing the key set of the display case, it gets each display case object, then gets the display tray set in the display case, and then prints the string representation of each display tray object by traversing the display tray set, so as to realize the function of displaying all display trays.

’showAllJewelryItems’ prints information about all the jewelry items. By traversing the key set of the display case, it obtains each display case object, and then obtains the display tray set in the display case, and then obtains each display tray object by traversing the display tray set, and then obtains the jewelry item set by the display tray object, and finally prints the string representation of each jewelry item object by traversing the jewelry item set, so as to realize the function of displaying all jewelry items.

2.3.4 Search Function

‘searchJewelryItems’ searches for jewelry items based on a keyword. It obtains each display case object by traversing the key set of the display case, and then obtains the display tray set in the display case, and then obtains each display tray object by traversing the display tray set, and then obtains the jewelry item set by the display tray object, and finally traversing the jewelry item set to determine whether the description information of the jewelry item contains the search keyword. And whether the composition information of the jewelry item contains the search keywords, so as to filter out the eligible jewelry items and add them to the result set, and finally return the result set.

2.3.5 Delete Function

‘deleteJewelryItem’ by traversing all the display cases, display plates and jewelry items in the jewelry store, enter the nature of the jewelry to be searched, and then match the eligible jewelry in the process of traversing, and then delete the found

jewelry, and update the original display cases and display plates that store the jewelry, and then delete the information about the jewelry.
‘clearAllJewelryItems’ will empty all the jewelry stored in the display case. The main task here is to initialize an empty display case and print a successful clearing prompt. Then delete the local file that stores the jewelry data, and finally display the deleted results.

2.3.6 Value Function

‘getAllStock’ creates 6 variables, including the total number of jewels (allCount), the total value of the jewels (allPrice), the number of jewels in each display case (caseCount) and the total value (casePrice). And the total number (trayCount) and total value (trayPrice) of the jewelry on each display plate. And set the initial values of all 6 variables to 0. These variables are linked using a nested loop, where the display case is traversed with for-each, and the display plate and jewelry items are traversed with a for loop. Finally, add up the quantity and value of the jewelry, and finally print out the display case object, the quantity and price of the jewelry respectively; Display plate object, jewelry quantity and price; The total number and value of jewelry in a jewelry store.

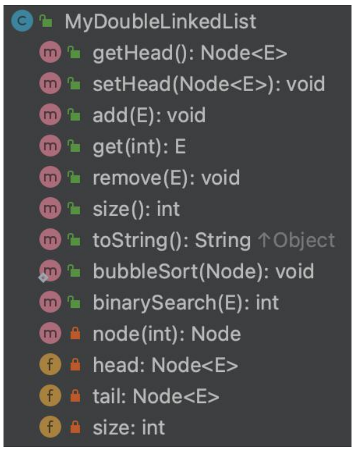
2.3.7 Save and Load Function

'Serialize' Because the file stored locally is binary, we need to serialize the object. Open the file 'store.txt' for serialization, create an object output stream, write the object to the output stream, and finally close the output stream and print out the result.
'Deserialize' To read binary files, open the file 'store.txt' to deserialize, create an input stream, convert the object type in the input stream, close the input stream, set the deserialized data to the current object, and print the result.

3 Implementation (including codes)

3.1 Data Structure

We write commonly function for Double Linked List to manage elements, for instance, add, get and remove function. And the Figure7 below is the UML of Double Linked List:



```
MyDoubleLinkedList
  getHead(): Node<E>
  setHead(Node<E>): void
  add(E): void
  get(int): E
  remove(E): void
  size(): int
  toString(): String ↑Object
  bubbleSort(Node): void
  binarySearch(E): int
  node(int): Node
  head: Node<E>
  tail: Node<E>
  size: int
```

Figure 11

3.1.1 Add Function

By using this function, we can add elements into the Double Linked List. When we add elements, there are two conditions:

- The linked list is empty, the new node becomes the head node, and the new node is also the tail node.
- The linked list is not empty, the next node of the current tail node become the new node, and the previous node of the new node become the current tail node.

The code of the Add Function is shown below:

```
    public void add(E element) {
            Node<E> newNode = new Node<>(element);    // Create new node
    if (head == null) { // If linked list is empty
                head = newNode; // The new node becomes the new head node
                tail = newNode; // The new node becomes the tail node
            } else {
                  tail.next = newNode;       // The next node of the current tail node
  point to                the new node
                newNode.prev = tail;       // The previous node of the new node
points to the             current tail node
                tail = newNode; // The new node becomes the new tail node
            }
            size++; // size+1
        }
```

3.1.2 Get Function

By using Get Function, we can obtain the element at the specified index position in the linked list. This method first checks whether the index is out of bounds, and if it is, throws an "IndexOutOfBoundsException" exception. Then, start from the head node and traverse the linked list until it reaches the node at the specified index position. Finally, return the data of the node. The codes are shown below:

```
    public E get(int index) {
            if (index < 0 || index >= size) {
                throw new IndexOutOfBoundsException("Index: " + index + ",
Size: " +                          size);
            }

            Node<E> current = head; // Begin from the head node
            for (int i = 0; i < index; i++) {
                current = current.next; // Move to the node at the specified index
position
            }
            return current.data;       // return the data of this node
        }
```

3.1.3 Remove Function

The remove method takes an element as input and searches for it in the linked list. If the element is found, it is removed from the list. There are three situations:

- If the element is at the head of the list, the head is updated to point to the next node.
- If the element is at the tail of the list, the tail is updated to point to the previous node.
- If the element is in the middle of the list, its previous node's next pointer and its next node's previous pointer are updated to bypass this node.

Finally, the size of the linked list is decremented by one. The codes are shown bellow:

```
public void remove(E element) {
        Node<E> current = head; // Begin from the head node
        while (current != null) {
         if (current.data.equals(element)) { // Find the matched element
              if (current == head) {    // If the head node is matched
                   head = current.next;       // Update the head node
                        if (head != null) {
                             head.prev = null;
                        }
              } else if (current == tail) {      // If the tail node is matched
                        tail = current.prev;        // Update the tail node
                        if (tail != null) {
                             tail.next = null;
                        }
                   } else {
                         current.prev.next = current.next;       // Update the next node of the previous node
                          current.next.prev = current.prev;       // Update the previous node of the next node
                   }
                   size--; // size-1
                   break;
              }
         current = current.next; // Continue searching for the next node
    }
   }
```

3.1.4 Bubble Sort

In this code, we use the Bubble Sort to sequence the elements when they are inserted into the list. And we first determine whether the linked list is empty or has only one node. If so, return directly. Otherwise, we defined three variables: current, switched, and end. Among them, current is used to record the nodes currently being traversed; Swapped is used to mark whether swapping has occurred and optimize the algorithm; End is used to record the end node of the sorted section.

Next, we enter the main loop. In the main loop, we first mark swapped as false, indicating that it is assumed that no swapping has occurred during this iteration. Then start traversing the linked list from the head node. In each iteration, we compare the values of the current node and the next node. If the current node value is greater than the next node value, swap their positions and mark swapped as true. Otherwise, we will continue to traverse the next node backwards.

When a round of traversal ends, we take the current node as the end node of the sorted part and continue with the next round of traversal. If there is no exchange during this iteration, it indicates that the sorting has been completed and the loop can be exited.

The codes are shown below:

```
        public static void bubbleSort(Node head) {
                if (head == null || head.next == null) {
                    // If the list is empty or has only one node, it returns
                    return;
                }

                // The end node used to record the sorted part
                Node current; // The node currently being traversed
                boolean swapped = true; // The mark whether an exchange has
    occurred is used to optimize the algorithm

                while (swapped) {
                    swapped = false; // Assume that no swapping takes place in
    this traverse
                    current = head; // The traversal starts at the beginning node

                    while (current.next != null) {
                        if (current.getData().hashCode()>current.next.getData().
                         hashCode())
                          {
                            // If the current node value is greater than the next
    node value, swap their positions
                            Object temp = current.data;
                            current.data = current.next.data;
                            current.next.data = temp;
                            swapped = true; // A swap has taken place, mark
    true
                        } else {
                            // The current node value is less than or equal to the
    next node value, and the loop continues to the next node
                            current = current.next;
                        }
                    }
                }
            }
```

3.1.5 Binary Search

This is a Java code that implements the binary search algorithm on a double linked list. The Binary Search method takes an element as input and searches for it in the linked list. If the element is found, its index is returned. If the element is not found, -1 is returned. Firstly, determine the upper and lower bounds of the search range, then obtain the nodes in the middle position, and obtain the element values in the middle position. Here are two conditions:

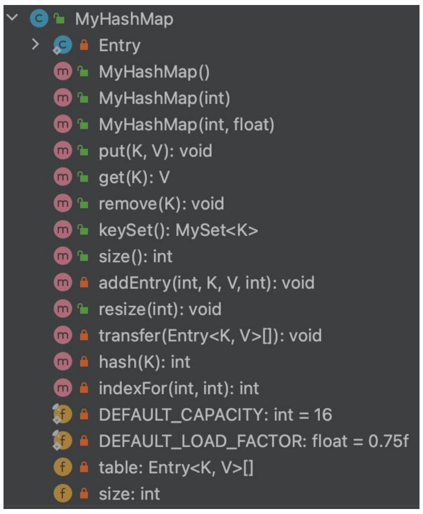
- ➢ The the element values in the middle position is lower than the key, narrow the search scope to the right half.
- ➢ The the element values in the middle position is higher than the key, narrow the search scope to the left half.

The codes are shown below:

```
        public int binarySearch(E key) {
                int low = 0; // Define a lower bound on the search range
                int high = size - 1; // Define a upper bound on the search range
                while (low <= high) { // When the search range is not empty
                        int mid = (low + high) >>> 1; // Calculating the middle
    position
                        Node midNode = node(mid); // Get the node in the middle
    position
                        int midVal = midNode.getData().hashCode(); // Get the value
    of the element in the middle position
                        if (midVal < key.hashCode()) { // If the value in the middle is
    less than the target value
                                low = mid + 1; // Narrow the search to the right half
                        } else if (midVal > key.hashCode()) { // If the value of the
    middle element is greater than the target value
                                high = mid - 1; // Narrow the search to the left half
                        } else { // If the value in the middle is equal to the target value
                                return mid; // Return the value of the middle
                        }
                }
                return -1; // If the target value is not found, return -1
            }
```

3.2 Algorithm

We use Hash Map to improve the efficiency of data operations, making it more efficient to search, insert, or sort large amounts of data. And the picture below is the UML of Hash Map:



```
MyHashMap
  Entry
  MyHashMap()
  MyHashMap(int)
  MyHashMap(int, float)
  put(K, V): void
  get(K): V
  remove(K): void
  keySet(): MySet<K>
  size(): int
  addEntry(int, K, V, int): void
  resize(int): void
  transfer(Entry<K, V>[]): void
  hash(K): int
  indexFor(int, int): int
  DEFAULT_CAPACITY: int = 16
  DEFAULT_LOAD_FACTOR: float = 0.75f
  table: Entry<K, V>[]
  size: int
```

Figure 12

3.2.1 Check Initialization

We need to check whether the initialization is valid and usable. Both of the initialization and load factor are required to be greater than 0. If either of them is below 0, the program will report an error and display the reason for the initialization setting error.

The code are shown below:

```
public MyHashMap(int initialCapacity, float loadFactor) {
        if (initialCapacity <= 0) {
            // Check if the initial capacity is legal
            throw new IllegalArgumentException("Illegal initial capacity:
" +                             initialCapacity);
        }
        if (loadFactor <= 0 || Float.isNaN(loadFactor)) {
         // Check if the load factor is legal
            throw new IllegalArgumentException("Illegal load factor: " +
                            loadFactor);
        }
        this.loadFactor = loadFactor;
        this.table = new Entry[initialCapacity];      // Initializing array
    this.threshold = (int) (initialCapacity * loadFactor);    // Set the upper
     limit of capacity
    }
```

3.2.2 Put Function

When we need to add a display case into the hash table, we first determine whether there is a hash table:

- if there has been already a hash map, the index value will be calculated by using hash code method;
- if there hasn't been already a hash map, we need to create a new hash table.

The codes are shown below:

```
public void put(K key, V value) {
        if (key == null) {
         throw new NullPointerException("Null key is not allowed!"); //
           Check if the key is null
        }
        int hash = hash(key);
        int index = indexFor(hash, table.length);
        for (Entry<K, V> entry = table[index]; entry != null; entry =
           entry.next) {
         if (entry.key.equals(key)) {        // Update the value if exists the
                same key
               entry.value = value;
               return;
            }
        }
        addEntry(hash, key, value, index);    // Add new key value pair
        }
```

3.2.3 Get Function

Firstly, check if the key is null, and if so, throw a NullPointerException exception. Then, it calculates the hash value of the key and uses the hash value to calculate the index of the key in the hash table. Finally, it traverses the linked list at that index in the hash table to find elements that match the specified key.

- If a matching element is found, its value is returned;
- Otherwise, it returns null.

The codes are shown below:

```
    public V get(K key) {
            if (key == null) {
             throw new NullPointerException("Null key is not allowed!");
             // Check if the key is null
            }
            int hash = hash(key);
            int index = indexFor(hash, table.length);
            // traverse the link list
            for (Entry<K, V> entry = table[index]; entry != null; entry =
            entry.next){
             if (entry.key.equals(key)) {       // Find the value for the key
                    return entry.value;
                 }
            }
        return null;       // Find unmatched key
        }
```

3.2.4 Remove Function

Firstly, check if the key is null, and if so, throw a NullPointerException exception. Then, calculate the hash value of the key and use the hash value to calculate the index of the key in the hash table. Finally, it traverses the linked list at that index in the hash table to find elements that match the specified key. If a matching element is found, its value is returned; Otherwise, it returns null.

```
    public void remove(K key) {
            if (key == null) {
             throw new NullPointerException("Null key is not allowed!");
             // Check if the key is null
            }
            int hash = hash(key);
            int index = indexFor(hash, table.length);
            Entry<K, V> prev = null;
            Entry<K, V> entry = table[index];
            while (entry != null) {
             if (entry.key.equals(key)) {
             // Find and remove key value pairs for the specified key
```

```
                if (prev == null) { // The key value pair to be deleted is the
                        first node in the list
                        table[index] = entry.next;    // Update the head of the
                        list
                  } else {
                        prev.next = entry.next;
                  }
                  size--; // size-1
                  return;
                }
                prev = entry;
                entry = entry.next;
            }
        }
```

3.3 Function

3.3.1 Add Function

This part is the method for adding cases to a collection of cases.

a. First, get the ID of the display case via the "displayCase.getId()" method.
   - ➢ If the 'usedDisplayCaseIds' collection already contains the ID of the case, print an error message and return it.
   - ➢ If the case ID is not duplicated, add the case object to the 'cases' hash table, using the case ID as the key.

b. Then, add the case ID to the 'usedDisplayCaseIds' collection to keep track of used case ids.

This ensures that the ID of the display case is unique in the collection.

The codes are shown below:

```
    public void addDisplayCase(DisplayCase displayCase) {
          if (usedDisplayCaseIds.contains(displayCase.getId())) {
                System.out.println("Error: Display case ID already exists");
                return;
          }
          cases.put(displayCase.getId(), displayCase);        // Add the display case
          to the hash table
          usedDisplayCaseIds.add(displayCase.getId());        // Update the set of ID
          of used display case}
```

This code defines a static method, addDisplayCase, that takes a Scanner as an parameter.

a. Firstly, the user is asked to enter the case ID, and then the entered ID is checked for normativity. If the length of the ID is 0, then a RuntimeException is thrown.

b. Then, it converts the ID into an array of characters and iterates over each character. For the first character, it checks if it is a capital letter. For later characters, it checks if they are numbers. If any characters don't conform to the specification, the method throws a RuntimeException and prints an error message. Inside the catch block, it prints an error message and returns it.

- ➢ If the ID doesn't fit the specification, this method will print an error and return without adding the DisplayCase object to storage.
- ➢ If the ID meets the specification requirements, the method proceeds by asking the user to enter the case type and a Boolean value of whether to turn on the lights. When reading a Boolean input, the method uses the nextBoolean() method. After reading the Boolean value, the method calls the scanner.nextLine() method to consume the newline character and prepare it to read the next line of input.

c. Finally, the method creates a DisplayCase object and sets its properties with the values entered by the user.

It then adds the DisplayCase object to the store.

The codes are shown below:

```
        private static void addDisplayCase(Scanner scanner) {
            System.out.print("Enter case id: ");
            String id = scanner.nextLine();

            //Determine whether the ID meets the specification requirements
            try{
                if (id.length() == 0 ){
                    throw new RuntimeException();
                }
                char[] idCharArray = id.toCharArray();
                for (int i = 0; i < idCharArray.length; i++) {
                    if (i == 0){
                        if (idCharArray[i] < 65 || idCharArray[i] > 90)
{    //Uppercase letters                    correspond to 65-90, and lowercase
letters correspond to 96-122
                            throw new RuntimeException();
                        }
                    } else {
                        if (idCharArray[i] < 48 || idCharArray[i] > 57) {    //The
numbers 0-9                   correspond to 48-57
                            throw new RuntimeException();
                        }
                    }
                }
            } catch(Exception e) {
                System.out.println("Please enter uppercase English + number
combination, for example: A12");
                return;
          }

          System.out.print("Enter case type: ");
              String type = scanner.nextLine();
              System.out.print("Is light on (true/false): ");
              boolean isLightOn = scanner.nextBoolean();
              scanner.nextLine();
            DisplayCase displayCase = new DisplayCase();
            displayCase.setId(id);
            displayCase.setType(type);
            displayCase.setLightOn(isLightOn);
            store.addDisplayCase(displayCase);
        }
```

This code is a method to add a display tray to a specified display case.

a. First, get the case object from the 'cases' hash table with the given 'caseId '.
   - If the case object is empty, print an error message and return.Next, get the tray ID via the 'tray.getId ()' method.
   - If the 'usedDisplayTrayIds' collection already contains the tray ID, print an error message and return it.
   - If the tray collection in the case is empty, create a new 'MyDoubleLinkedList<DisplayTray>' object 'displayTrays' to which to add the display trays and set it as the collection of display trays for the display case.
   - If the display tray set in the display case is not empty, directly add the display tray to the display tray set in the display case.

b. Finally, add the tray ID to the 'usedDisplayTrayIds' collection to keep track of used tray IDs.

This ensures that the display tray ID is unique across the collection.

The code are shown below:

```
    public void addDisplayTray(String caseId, DisplayTray tray) {
        DisplayCase displayCase = cases.get(caseId);
        if (displayCase == null) {
            System.out.println("Error: Display case not found");
            return;
        }

        if (usedDisplayTrayIds.contains(tray.getId())) {
            System.out.println("Error: Display tray ID already exists");
            return;
        }

        if (displayCase.getTrays() == null) {      // The case is empty
            MyDoubleLinkedList<DisplayTray> displayTrays = new
                MyDoubleLinkedList<>();
            displayTrays.add(tray);
            displayCase.setTrays(displayTrays);
        } else {
            displayCase.getTrays().add(tray);
        }

        usedDisplayTrayIds.add(tray.getId());
    }
```

This code defines a static method, addDisplayTray, that takes a Scanner as an parameter.

a. First ask the user to enter the case ID, tray ID, tray color, tray width and height, and store them in the caseId, id, color, width and height variables, respectively. When reading an integer input, the method uses the nextInt() method. After reading the integer, the method calls scanner.nextLine() to consume the newline character and prepare it to read the next line of input, because nextInt() will only read integers, not newlines.
b. Next, create a DisplayTray object and set its properties with the values entered by the user.
c. Finally, call the store.addDisplayTray(caseId, tray) method to add the tray object to the store for the specified case.

The codes are shown below:

```
        private static void addDisplayTray(Scanner scanner) {
                System.out.print("Enter case id: ");
                String caseId = scanner.nextLine();
                System.out.print("Enter tray id: ");
                String id = scanner.nextLine();
                System.out.print("Enter tray color: ");
                String color = scanner.nextLine();
                System.out.print("Enter tray width: ");
                int width = scanner.nextInt();
                System.out.print("Enter tray height: ");
                int height = scanner.nextInt();
                scanner.nextLine();
                DisplayTray tray = new DisplayTray();
                tray.setId(id);
                tray.setColor(color);
                tray.setWidth(width);
                tray.setHeight(height);
                store.addDisplayTray(caseId, tray);
        }
```

This code is a method to add a jewelry item to the jewelry display case.

a. First, get the 'DisplayCase' object from the 'cases' collection with the given' caseId 'parameter.
   - ➢ If 'displayCase' is empty, print an error message and return.
b. Next, get the 'trays' from' displayCase ', which is a 'MyDoubleLinkedList&lt; DisplayTray&gt; 'type.
   - ➢ If 'trays' is empty or has size 0, print an error message and return.
c. - Create a Boolean variable 'flag', initialized to false.Create a 'tray' object and initialize it as the first element in 'trays'.Use a loop to iterate over each element in the 'trays', and if a' tray 'matching the given' trayId 'is found, set' flag 'to true and break out of the loop.
   - ➢ If no matching 'tray' is found, print an error message and return.
d. Check if the 'items' in the' tray 'are empty.

- ➢ If 'items' is empty, create a new' MyDoubleLinkedList&lt; JewelryItem&gt; 'type object' jewelryItems', add 'item' to it, and set it as' items' of 'tray'.
- ➢ If 'items' is not empty, add' item 'directly to' items' on 'tray'.

e. The purpose of this code is to add jewelry items to the designated display cases and trays. It performs various checks, such as checking whether the display case and display tray exist and whether there are already items in the display tray.

- ➢ If there are any error conditions, it prints an error message and returns.

The codes are shown below:

```
public void addJewelryItem(String caseId, String trayId, JewelryItem item) {
    DisplayCase displayCase = cases.get(caseId);
    if (displayCase == null) {
        System.out.println("Error: Display case not found");
        return;
    }

    MyDoubleLinkedList<DisplayTray> trays = displayCase.getTrays();
    if (trays == null || trays.size() == 0) {
        System.out.println("Error: Display case not have this tray");
        return;
    }
    boolean flag = false;
    DisplayTray tray = trays.get(0);
    for (int i = 0; trays != null && i < trays.size(); i++) {
        tray = trays.get(i);
        if (tray.getId().equals(trayId)) {
            flag = true;
            break;
        }
    }
    if (!flag) {
        System.out.println("Error: Display case not have this tray");
        return;
    }

    if (tray.getItems() == null) {    // If the tray is empty
        MyDoubleLinkedList<JewelryItem> jewelryItems = new
MyDoubleLinkedList<>();
        jewelryItems.add(item);
        tray.setItems(jewelryItems);
    } else {
        tray.getItems().add(item);
    }
}
```

This code defines a static method, addJewelryItem, that takes a Scanner as an parameter.

a. First the user is asked to enter the case ID, tray ID, item description, item type, item gender, item picture URL, and item price.

b. Then, it creates a JewelryItem object and sets its properties with the values entered by the user.
c. Next, it creates an empty doubly linked components list to store the components information of the items.
d. Then, it goes into a loop where the user can enter the ingredient information of the item. On each pass through the loop, it asks the user to enter the name, description, and quantity of the ingredient and creates a Component object to store this information.
e. Then, it adds that Component object to the components list. After each loop, it asks the user whether to continue adding ingredients, and if the user chooses not to continue, the loop ends. When reading a Boolean input, the method uses the nextBoolean() method. After reading the Boolean value, the method calls the scanner.nextLine() method to consume the newline character and prepare it to read the next line of input. Inside the loop, to consume newline characters, the method calls scanner.nextLine() at the beginning of each loop.
f. Finally, it sets the components list to the components property of the JewelryItem object and adds the item to the store for the specified case and tray.

The codes are shown below:

```
        private static void addJewelryItem(Scanner scanner) {
            System.out.print("Enter case id: ");
            String caseId = scanner.nextLine();
            System.out.print("Enter tray id: ");
            String trayId = scanner.nextLine();
            System.out.print("Enter item description: ");
            String description = scanner.nextLine();
            System.out.print("Enter item type: ");
            String type = scanner.nextLine();
            System.out.print("Enter item gender: ");
            String gender = scanner.nextLine();
            System.out.print("Enter item image url: ");
            String imageUrl = scanner.nextLine();
            System.out.print("Enter item price: ");
            double price = scanner.nextDouble();
            JewelryItem item = new JewelryItem();
            item.setDescription(description);
            item.setType(type);
            item.setTargetGender(gender);
            item.setImageUrl(imageUrl);
            item.setPrice(price);
            MyDoubleLinkedList<Component> components = new
MyDoubleLinkedList<>();
            System.out.println("Enter item components ");
            while (true) {   // Add the conponent information of the jewelry in a loop
                Component component = new Component();
                scanner.nextLine();
                System.out.println("Enter component name: ");
                String name = scanner.nextLine();
                System.out.println("Enter component description: ");
                String desc = scanner.nextLine();
                System.out.println("Enter component count: ");
                int count = scanner.nextInt();
                component.setName(name);
                component.setDescription(desc);
                component.setCount(count);
                components.add(component);
                System.out.print("Is continue (true/false): ");
                boolean isContinue = scanner.nextBoolean(); // Whether to continue
adding
                if (!isContinue) {
                    break;
                }
            }
            item.setComponents(components);
            store.addJewelryItem(caseId, trayId, item);
        }
```

3.3.2 Smart Add

```
private static void smartAddJewelryItem(Scanner scanner) {
    System.out.print("Enter item description: ");
    String description = scanner.nextLine();
    System.out.print("Enter item type: ");
    String type = scanner.nextLine();
    System.out.print("Enter item gender: ");
    String gender = scanner.nextLine();
    System.out.print("Enter item image url: ");
    String imageUrl = scanner.nextLine();
    System.out.print("Enter item price: ");
    double price = scanner.nextDouble();
    scanner.nextLine();
    JewelryItem item = new JewelryItem();
    item.setDescription(description);
    item.setType(type);
    item.setTargetGender(gender);
    item.setImageUrl(imageUrl);
    item.setPrice(price);
    MyDoubleLinkedList<Component> components = new
MyDoubleLinkedList<>();
    System.out.println("Enter item components ");
    while (true) {
        Component component = new Component();
        System.out.println("Enter component name: ");
        String name = scanner.nextLine();
        System.out.println("Enter component description: ");
        String desc = scanner.nextLine();
        System.out.println("Enter component count: ");
        int count = scanner.nextInt();
        component.setName(name);
        component.setDescription(desc);
```

```
            component.setCount(count);
            components.add(component);
            try {
                System.out.print("Is continue (true/false): ");
                boolean isContinue = scanner.nextBoolean(); // Whether to continue
adding
                if (!isContinue) {
                    break;
                }
            }catch (InputMismatchException e){
                System.out.println("Please entry true or false");
                return;
            }

        }
        item.setComponents(components);
        // *** All above is the same as addJewelryItem
        String caseId = "";
        String trayId = "";
        MyHashMap<String, DisplayCase> displayCaseMyHashMap =
store.showAllDisplayCase();

        // Iterate over all the jewels and find the one with the highest similarity
        int tempScore = 0;
        for (String s : displayCaseMyHashMap.keySet()) {      // Iterate over all display
cases
            MyDoubleLinkedList<DisplayTray> tempTrays =
displayCaseMyHashMap.get(s).getTrays();
            if (tempTrays != null && tempTrays.size() != 0) {
                for (int i = 0; i < tempTrays.size(); i++) {
                    MyDoubleLinkedList<JewelryItem> tempItems =
tempTrays.get(i).getItems();
                    if (tempItems != null && tempItems.size() != 0) {
                        for (int j = 0; j < tempItems.size(); j++) {
                            // If the score is higher than the original score, the
jewelry and score will be retained
                            int compareScore = JewelryItem.compare(item,
tempItems.get(j));
                            if (compareScore > tempScore && compareScore > 120)
{
                                // In the case of different types and genders, at least
the amount difference is within 10 times and the two elements are the same
                                tempScore = compareScore;
                                caseId = displayCaseMyHashMap.get(s).getId();
                                trayId = tempTrays.get(i).getId();
                            }
                        }
                    }
                }
            }
        }
        // No similar jewelry, put it at the end
        if (tempScore == 0) {
              MySet<String> keys = displayCaseMyHashMap.keySet();
              DisplayCase tempCase =    displayCaseMyHashMap.get((String)
keys.toArray()[keys.size() - 1]);
              caseId = tempCase.getId();
              MyDoubleLinkedList<DisplayTray> tempTrays = tempCase.getTrays();
              trayId = tempTrays.get(tempTrays.size() - 1).getId();
        }

        store.addJewelryItem(caseId, trayId, item);
}
```

3.3.3 Show Function

This code is a method to display all the display cases.

a. First, get the keySet of all the cases in the showcase hash table via "cases.keyset ()". Use an enhanced for loop to iterate over the key set of the case, retrieve the case object for each key through "cases.get(s)", and call its " toString()" method to convert the case object to a string and print it.
b. Finally, return the case hash table "cases".

The codes are shown below:

```
public MyHashMap<String, DisplayCase> showAllDisplayCase() {
      for (String s : cases.keySet()) {
            System.out.println(cases.get(s).toString());
      }
      return cases;
}
```

This code is a method to display all the display plates.

a. First, get the keySet of all the cases in the showcase hash table via "cases.keyset ()". Use an enhanced for loop to iterate over the key collection of the case and retrieve the case object for each key via "cases.get(s)".
b. Then, get the collection of trays in the case via the 'getTrays()' method of the case object. Use a regular for loop to iterate over the dashboard collection and print a string representation of each dashboard object.

The codes are shown below:

```
public void showAllDisplayTray() {
      for (String s : cases.keySet()) {
            DisplayCase displayCase = cases.get(s);
            MyDoubleLinkedList<DisplayTray> trays = displayCase.getTrays();
            for (int i = 0; trays != null && i < trays.size(); i++) {
                  System.out.println(trays.get(i).toString());
            }
      }
}
```

This part of code is a method that displays all the jewelry items.First, get the keySet of all the cases in the showcase hash table via 'cases.keyset ()'.Use an enhanced for loop to iterate over the key collection of the case and retrieve the case object for each key via 'cases.get(s)'.Then, get the collection of trays in the case via the 'getTrays()' method of the case object.Use a regular for loop to iterate over the dashboard collection and get each dashboard object.Then, get the collection of jewelry items on the display panel using the 'getItems()' method of the display panel object.Use a regular for loop again to loop through the collection of jewelry items, printing a string representation of each item object.

The code are shown below:

```
public void showAllJewelryItems() {
    for (String s : cases.keySet()) {     // Iterate over all the DisplayCase
        DisplayCase displayCase = cases.get(s);
        MyDoubleLinkedList<DisplayTray> trays = displayCase.getTrays();
        // Get all DisplayTray in the case
        for (int i = 0; trays != null && i < trays.size(); i++) {
            // Iterate over all
            DisplayTray in the DiaplayCase
            DisplayTray tray = trays.get(i);
            MyDoubleLinkedList<JewelryItem>  items  =  tray.getItems();
            // Get all jewelries in the DispalyTray
            for (int j = 0; items != null && j < items.size(); j++) {
                System.out.println(items.get(j).toString());
            }
        }
    }
}
```

3.3.4 Search Function

This code is a method that searches for jewelry items based on a keyword.

a. First, create a 'results' variable of type' HashSet 'to store the search results. Use 'cases.keyset ()' to get the set of keys for all the cases in the showcase hash table. Use an enhanced for loop to iterate over the key collection of the case and retrieve the case object for each key via 'cases.get(s)'.

b. Then, get the collection of trays in the case via the 'getTrays()' method of the case object.Use a regular for loop to iterate over the dashboard collection and get each dashboard object.

c. Then, get the collection of jewelry items on the display panel using the 'getItems()' method of the display panel object. Use a regular for loop again to iterate over the collection of jewelry items and get each item object.
   - If the search term is included in the item description. Add the item to 'results' if the search term is included in the item description.
   - Otherwise, get the component information from the jewelry object's 'getComponents()' method.

a. We use a regular for loop again to traverse the collection of constituent information, getting each constituent information object, and add the jewelry item to 'results' if the search keyword is part of the name or description of the constituent information.
b. By using the 'continue' keyword in the inner loop, we can skip the current item loop and proceed to the next item loop. Then use the 'break' keyword in the inner loop allows it to break out of the current loop of composing information and continue to the next loop of jewelry items.
c. Finally, return the 'results' variable where the search results will be stored.

The codes are shown below:

```
    public Set<JewelryItem> searchJewelryItems(String keyword) {
        Set<JewelryItem> results = new HashSet<>();
        Set<String> keySet = cases.keySet();
        for (String s : keySet) {      // Iterate over all cases based on the key value
            DisplayCase displayCase = cases.get(s);
            MyDoubleLinkedList<DisplayTray> trays = displayCase.getTrays();
            // Get all trays of the case
            for (int i = 0; trays != null && i < trays.size(); i++) {
                // Iterate over all DisplayTray
                MyDoubleLinkedList<JewelryItem> items =
                trays.get(i).getItems();
                // Get all jewelries of the DispalyTray
                for (int j = 0; items != null && j < items.size(); j++) {
                    // Iterate over all jewelries
                    JewelryItem item = items.get(j);
                    if (item.getDescription().contains(keyword)) {
                      // If the description of the jewelry contains the search
                      keywords
                        results.add(item);
                        // Add the jewelry to the result set
                        continue;
                    }
                    MyDoubleLinkedList<Component> components =
                    item.getComponents();
                    // Get information about the composition of the jewelry
                    for (int k = 0; components != null && k <
                    components.size(); k++) {
                        // Iterate over the composition information
                        Component component = components.get(k);
                        if (component.getName().contains(keyword) ||
                                component.getDescription().contains(keyword))
                      {       // Composition and names contain keywords
                            results.add(item);
                            break;
                        }
                    }
                }
            }
        }
        return results;
    }
```

3.3.5 Delete Function

This code finds and removes eligible jewelry items from display cases and trays based on the given jewelry item description, type, applicable gender, and price. Determines whether the jewelry item's description, type, applicable gender, and price match the given parameters, and if so, removes the jewelry item from the collection and updates the related objects and collections.

a. First of all, in the JewelryStore, we mainly use nested loops to go through the display cases, display plates and all jewelry items. The outermost layer uses a for loop to iterate through the key sets of all display cabinets in the display cabinet hash table.
b. Then in the loop, get each display case object by the 'get()' method.
c. Then get the display trays collection in the display cabinet with the getTrays() method.
d. Then the second layer of for loops through the display disk, obtaining each display disk object.
e. Next, the collection of jewelry items in the display tray is obtained using the 'getItems()' method.
f. Then add a third layer for loop to the second layer to iterate through the collection of jewelry items, obtaining each jewelry item object.
g. Finally, Compare the input to the found jewelry item for a match
   - ➢ It doesn't match. It can't be deleted
   - ➢ the jewelry item is removed from the collection using the 'remove()' method.

Update the display tray collection and jewelry items in the display tray with the setTrays() and setItems() methods. Use the 'put()' method to update the display cabinet object in the display cabinet hash table. After the deletion is successful, print "remove success".

The codes are shown below:

```
    public void deleteJewelryItem(String description, String type, String gender,
double price) {
        for (String s : cases.keySet()) {     // Iterate over all display cases

            DisplayCase displayCase = cases.get(s);
            MyDoubleLinkedList<DisplayTray> trays = displayCase.getTrays();
          //Get all display tray in the case
            for (int i = 0; trays != null && i < trays.size(); i++) {
                // Iterate over all display tray in this case
                DisplayTray tray = trays.get(i);
                MyDoubleLinkedList<JewelryItem> items = tray.getItems();
                // Get all jewelry item in the tray

                for (int j = 0; items != null && j < items.size(); j++) {
                    // Iterate over all    jewelry item in this tray
```

```
                        JewelryItem item = items.get(j);
                        // Remove a jewel based on the information
                        if (item.getDescription().equals(description) &&
item.getType().equals(type) && item.getTargetGender().equals(gender) &&
item.getPrice() == price) {
                            items.remove(item);
                            tray.setItems(items);
                            displayCase.setTrays(trays);
                            cases.put(s, displayCase);
                            System.out.println("remove success");
                        }
                    }
                }
            }
        }
```

This code implements an interactive function to delete jewelry items. The deletion of the corresponding jewelry item is achieved by prompting the user to enter the description, type, gender, and price of the jewelry item, and then passing this information as parameters to the store object's 'deleteJewelryItem' method.

a. First,print a prompt message using the 'System.out.print' method, prompting the user to enter a description, type, gender, and price of the jewelry item.
b. Then,use the 'scanner.nextLine()' method to get the user's input jewelry item description, type, gender and price and save it to the 'description' variable.
c. Finally, call the 'deleteJewelryItem' method in 'JewelryStore' in the main program, passing the description, type, gender and price of the jewelry item as parameters, in order to delete the corresponding jewelry item from the store.

```
private static void deleteJewelryItem(Scanner scanner) {
    System.out.print("Enter item description: ");
    String description = scanner.nextLine();
    System.out.print("Enter item type: ");
    String type = scanner.nextLine();
    System.out.print("Enter item gender: ");
    String gender = scanner.nextLine();
    System.out.print("Enter item price: ");
    double price = scanner.nextDouble();
    store.deleteJewelryItem(description, type, gender, price);
}
```

The purpose of this code is to clear all jewelry items and clean up the associated data structures and files. Clear all jewelry items by reinitializing the display cabinet hash table and deleting the files that store the jewelry items.

a. First,initialize the display cabinet hash table to empty and print "clear all items success." to indicate successful clearing.
b. Then create a file object 'file' to store the jewelry item. Check whether the file exists.

- The file exists, the 'delete()' method attempts to delete the file object and returns a Boolean value indicating whether the deletion was successful.

➢ The deletion is successful, "file has been deleted." is printed. If the deletion fails, print "Failed to delete file.".

The codes are shown below:

```
public void clearAllJewelryItems() {
cases = new MyHashMap<>();
System.out.println("clear all items success.");
    File file = new File("store.txt");
// Check whether the file exists
    if (file.exists()) {
        // Attempt to delete file
        boolean deleted = file.delete();
        if (deleted) {
            System.out.println("file has been deleted.");
        } else {
            System.err.println("Failed to delete file.");
        }
    }
}
```

This code calls the claerAllJewelryItems method in the JewelryStore in the main program.

The codes are shown below:

```
private static void clearAllJewelryItems() {
    store.clearAll();
}
```

3.3.6 Value Function

This code goes through all the display cases and display plates, counts the number and total value of the jewelry in each display plate and display case, and the total number and total value of the entire jewelry inventory, and finally prints the results.

a. First, six variables are defined, including the total number of jewels in all display cases (allCount) and total value (allPrice), and the number of jewels in each display case (caseCount) and total value (casePrice). And the trayCount and the total value of each display plate are used to store the statistical results.
b. Then 'cases.keyset ()' traverses all display cases and retrieves from each display case all display plates in that case.
c. Then use trays.size() to get the number of trays. From each tray, get all the jewelry in that tray.
d. Finally, use 'items.size()' to get the number of jewels. When walking through the jewelry, the amount and total value of the jewelry are added to the display plate, the display case and the overall statistical results. Use 'System.out.println()' to print out statistics for each display plate and display case and for all jewelry inventory in the jewelry store.

```
public void getAllStock() {
    int allCount = 0;
    double allPrice = 0;
```

```
        for (String s : cases.keySet()) {      // Iterate over all display cases

            int caseCount = 0;
            double casePrice = 0;
            DisplayCase displayCase = cases.get(s);
            MyDoubleLinkedList<DisplayTray> trays = displayCase.getTrays();
            // Get all display tray in the case
            for (int i = 0; trays != null && i < trays.size(); i++) {
                // Iterate over all display tray in this case
                int trayCount = 0;
                double trayPrice = 0;
                DisplayTray tray = trays.get(i);
                MyDoubleLinkedList<JewelryItem> items = tray.getItems();
                // Get all jewelry item in the tray
                for (int j = 0; items != null && j < items.size(); j++) {
                // Iterate over all    jewelry item in this tray
                    JewelryItem item = items.get(j);

                    trayCount++;
                    caseCount++;
                    allCount++;
                    trayPrice += item.getPrice();
                    casePrice += item.getPrice();
                    allPrice += item.getPrice();
                }
                System.out.println(tray + " -> count: " + trayCount + " price: "
+ trayPrice);
            }
            System.out.println(displayCase + " -> count: " + caseCount + " price:
" + casePrice);
        }
        System.out.println("all -> count: " + allCount + " price: " + allPrice);
    }
```

This code calls the getAllStock method in the JewelryStore in the main program

```
    private static void showStock() {
        store.getAllStock();
    }
```

3.3.7 Save and Load Function

This code serializes the current object and saves the serialized data to the file 'store.txt'. After the data is saved successfully, a message indicating that the data is saved successfully is printed. If the save fails, a message indicating that the save failed is displayed.

a. First create a file output stream object fileOut with the FileOutputStream class and open the file store.txt for serialization.
b. Then use 'ObjectOutputStream' class to create an object output stream object 'out', and then use 'Out.WriteObject (this)' to write the current object to the output stream to achieve serialization operation.

c. Finally, close the object output stream and file output stream by out.close() and fileOut.close(). 'System.out.println()' prints the message that the data was saved successfully.

```
public void serialize() {
    try {
        // Open the file 'store.txt' for serialization
        FileOutputStream fileOut = new FileOutputStream("store.txt");
        // Create an object output stream
        ObjectOutputStream out = new ObjectOutputStream(fileOut);
        // Writes the current object to the output stream
        out.writeObject(this);
        // Close output stream
        out.close();
        // Close the file output stream
        fileOut.close();
        // Print a successful message
        System.out.println("Data saved successfully");
    } catch (IOException e) {
        System.err.println("Data save failure");
    }
}
```

This code calls the serialize method in the JewelryStore in the main program

```
private static void saveData() {
    store.serialize();
}
```

This code deserializes the file 'store.txt' and reads the data, setting the read data to the corresponding property of the current object. After reading successfully, print out the message that the data has been read successfully; If the read fails, a message indicating that the read failed is printed.

a. First create a file input stream object 'fileIn' with the class 'FileInputStream' and open the file 'store.txt' for deserialization.
b. Then create an object input stream object 'in' with the ObjectInputStream class, read the object from the input stream with '(JewelryStore) in.readobject ()', convert the type to 'JewelryStore' type, And assign to an object variable 'obj'.
c. Then close the object input stream and the file input stream with in.close() and fileIn.close(). Assign the deserialized 'cases' property value to the current object's' cases' property with 'obj.cases',
d. Finally use ` obj. GetUsedDisplayCaseIds () ` and ` obj. GetUsedDisplayTrayIds () ` gain after deserialization ` usedDisplayCaseIds ` and ` usedDisplayTrayIds ` attribute values, And assigns values to the 'usedDisplayCaseIds' and' usedDisplayTrayIds' properties of the current object. 'System.out.println()' prints the message that the data was read successfully.

```
 public void deserialize() {
        try {
            // Open the file 'store.txt' for deserialization
            FileInputStream fileIn = new FileInputStream("store.txt");
```

```
                // Create an object input stream
                ObjectInputStream in = new ObjectInputStream(fileIn);
                // Reads an object from an input stream and casts it
                JewelryStore obj = (JewelryStore) in.readObject();
                // Close input stream
                in.close();
                // Close the file input stream
                fileIn.close();
                // Sets the deserialized data to the current object
                this.cases = obj.cases;
                this.usedDisplayCaseIds = obj.getUsedDisplayCaseIds();
                this.usedDisplayTrayIds = obj.getUsedDisplayTrayIds();
                // Print a successful message
                System.out.println("Data read successfully");
            } catch (IOException | ClassNotFoundException e) {
                System.err.println("Data read failure");
            }
        }
}
```

This code calls the deserialize method in the JewelryStore in the main program

```
 private static void loadData() {
            store.deserialize();
        }
}
```

3.4 Junit Test

3.4.1 TestMethod

```
public class TestMethods {

        @Test
        public void testSort() {
            MyDoubleLinkedList<Object> list = new MyDoubleLinkedList<>();
            list.add("A1");
            list.add("B1");
            list.add("B02");
            list.add("C01");
            list.add("C04");
            list.add("A01");
            for (int i = 0; i < list.size(); i++) {
                System.out.println(list.get(i).toString() + "       ,     hasecode :"   +
list.get(i).toString().hashCode());
            }
            System.out.println("-------------------------------------After sort-----------------
-----------------");
            MyDoubleLinkedList.bubbleSort(list.getHead());
```

```
		for (int i = 0; i < list.size(); i++) {
			System.out.println(list.get(i).toString() + "        ,      hasecode :"    +
list.get(i).toString().hashCode());
		}
	}//check the result,the object in list have been sorted

	@Test
	public void testSearch() {
		MyDoubleLinkedList<Object> list = new MyDoubleLinkedList<>();
		list.add("A01");
		list.add("B01");
		list.add("B02");
		list.add("C01");
		list.add("C04");
		list.add("A01");

		MyDoubleLinkedList.bubbleSort(list.getHead());
		for (int i = 0; i < list.size(); i++) {
			System.out.println(list.get(i).toString() + "        ,      hasecode :"    +
list.get(i).toString().hashCode());
		}
		int i =    list.binarySearch("B02");
		System.out.println("------------------------------------search result:"+ i +"-----
-----------------------------" );
	}
}
//check the result,it will show the position of B02(after sort)
```

3.4.2 TestMyDoubleLinkedList

```
public class TestMyDoubleLinkedList {
	MyDoubleLinkedList<String> t;
	//add four item to list
	@Before
	public void setUp() throws Exception {
		t=new MyDoubleLinkedList<>();
		t.add("A");
		t.add("B");
		t.add("C");
		t.add("D");
	}

	@After
	public void tearDown() throws Exception {
		t=null;//clear list
	}

	@Test
	public void testAdd(){
		int size = t.size();
		t.add("E");
		t.add("F");
		assertEquals(t.size(),size+2);//add two new item to list,the size of list should
add two too.
	}

	@Test
	public void testRemove() {
		int size = t.size();
		t.remove("A");
		assertEquals(t.size(),size-1);//remove two item from list,the size of list
should decrease two.
	}

	@Test
	public void testSize(){
		assertEquals(t.size(),4);//size method should return the size of list
	}

}
```

3.4.3 TestMyHashMap

```
public class TestMyHashMap {

	@Test
	public void testMyHashMapPut() {
		MyHashMap<String,Object> map = new MyHashMap<>();
		map.put("6666","666");
		System.out.println(map.size());
	}//should print 1


	@Test
	public void testMyHashMapGet() {
		MyHashMap<String,Object> map = new MyHashMap<>();
		map.put("6666","666");
		System.out.println(map.get("6666"));
	}//should print 666


	@Test
	public void testMyHashMapRemove() {
		MyHashMap<String,Object> map = new MyHashMap<>();
		map.put("6666","666");
		map.remove("6666");
		System.out.println(map.size());
	}//should print 0
```

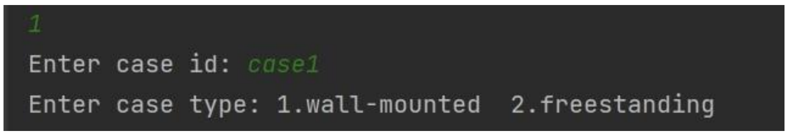
```
    @Test
    public void testMyHashMapResize() {
        MyHashMap<String,Object> map = new MyHashMap<>();
        map.resize(5);
    }//the capacity of hashmap should change
}
```

3.4.4 TestMySet

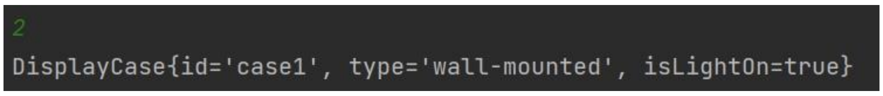
```
public class TestMySet {
    MySet<String> t ;

    @Before
    public void setUp() throws Exception {
        t =new MySet<>();
        t.add("A");
        t.add("B");
        t.add("C");
        t.add("D");
    }

    @After
    public void tearDown() throws Exception {
        t=null;
    }

    @Test
    public void testAdd(){
        int size = t.size();
        t.add("A");
        assertEquals(t.size(),size);//add some item in set,add will false,size of set
does not change
        t.add("E");
        t.add("F");
        assertEquals(t.size(),size+2);//add 2 new item to set,size of item add 2
    }

    @Test
    public void testRemove() {
        int size = t.size();
        t.remove("A");
        assertEquals(t.size(),size-1);//remove 1 item,size decrease 1
    }

    @Test
    public void testContains() {
        assertTrue(t.contains("A"));//check the item in set or not
```

```
            assertTrue(t.contains("B"));//return true
        }
}
```

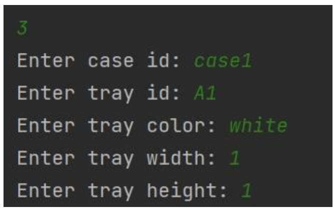
4 Results and Conclusions (Results *analysis and discussion*)



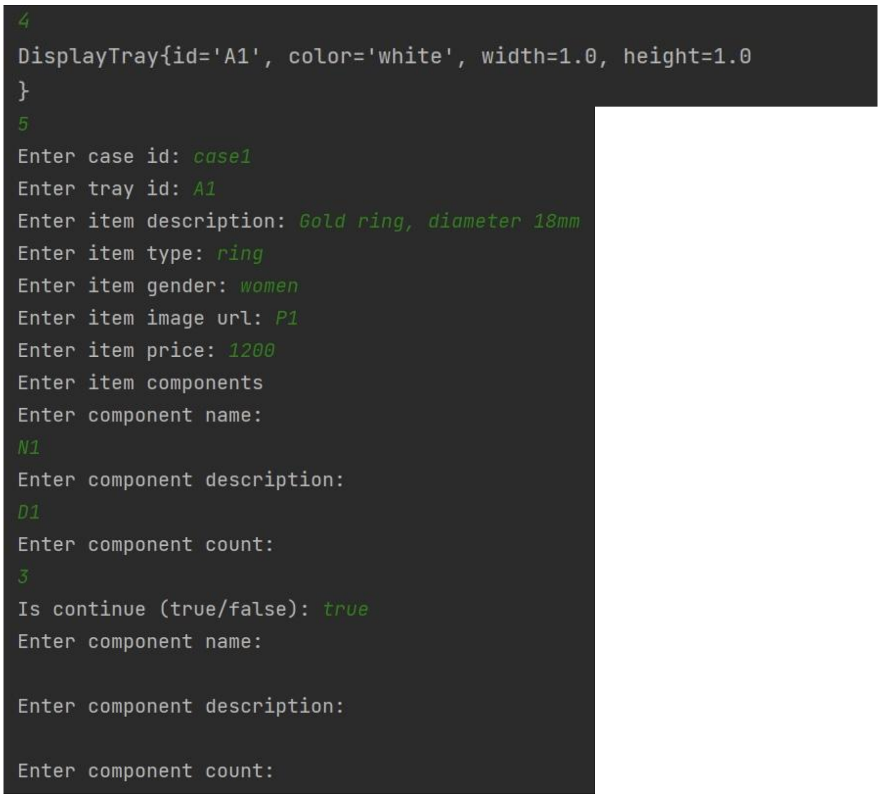
```
1
Enter case id: case1
Enter case type: 1.wall-mounted  2.freestanding
```



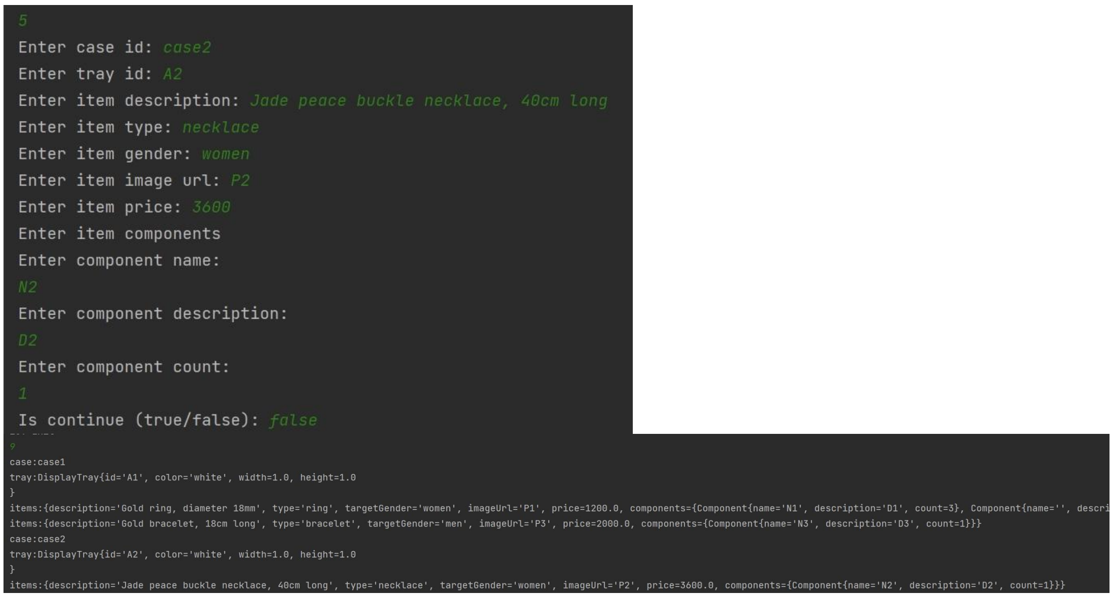
```
2
DisplayCase{id='case1', type='wall-mounted', isLightOn=true}
```



```
3
Enter case id: case1
Enter tray id: A1
Enter tray color: white
Enter tray width: 1
Enter tray height: 1
```



```
4
DisplayTray{id='A1', color='white', width=1.0, height=1.0
}
5
Enter case id: case1
Enter tray id: A1
Enter item description: Gold ring, diameter 18mm
Enter item type: ring
Enter item gender: women
Enter item image url: P1
Enter item price: 1200
Enter item components
Enter component name:
N1
Enter component description:
D1
Enter component count:
3
Is continue (true/false): true
Enter component name:

Enter component description:

Enter component count:
```



```
5
Enter case id: case2
Enter tray id: A2
Enter item description: Jade peace buckle necklace, 40cm long
Enter item type: necklace
Enter item gender: women
Enter item image url: P2
Enter item price: 3600
Enter item components
Enter component name:
N2
Enter component description:
D2
Enter component count:
1
Is continue (true/false): false
```

```
9
case:case1
tray:DisplayTray{id='A1', color='white', width=1.0, height=1.0
}
items:{description='Gold ring, diameter 18mm', type='ring', targetGender='women', imageUrl='P1', price=1200.0, components={Component{name='N1', description='D1', count=3}, Component{name='', descri
items:{description='Gold bracelet, 18cm long', type='bracelet', targetGender='men', imageUrl='P3', price=2000.0, components={Component{name='N3', description='D3', count=1}}}
case:case2
tray:DisplayTray{id='A2', color='white', width=1.0, height=1.0
}
items:{description='Jade peace buckle necklace, 40cm long', type='necklace', targetGender='women', imageUrl='P2', price=3600.0, components={Component{name='N2', description='D2', count=1}}}
```

5 References