Homework-2 Report

Inventory Management System

Introduction

This assignment focuses on developing an efficient **inventory management system** using a **linked list data structure**. The system allows users to dynamically manage inventory items by adding, deleting, and printing them. A notable feature is its ability to display the inventory in an organized and sorted manner, either alphabetically by item names or numerically by item counts. This ensures clarity and ease of use, making the system practical for real-world applications.

Built in **Java**, the project integrates **object-oriented principles** with **data structures** and **sorting algorithms**, demonstrating a seamless application of theoretical knowledge to solve practical problems. The linked list approach ensures efficient memory management and dynamic handling of varying inventory sizes, making the system both scalable and adaptable to diverse scenarios.

Purpose

This program offers a streamlined and highly effective solution for managing inventory items such as resistors, capacitors, inductors, and transistors. By ensuring that the inventory remains dynamically updatable and consistently sorted, the system allows users to easily query, modify, and maintain their inventory without any hassle. Whether organizing items alphabetically by name or numerically by quantity, this program guarantees a user-friendly and organized approach to inventory management.

Central to this solution is the implementation of a **linked list data structure**, which provides the flexibility and efficiency needed to handle dynamic inventory sizes. The use of linked lists ensures seamless insertion, deletion, and traversal of data while preserving the specified order of items. This project not only highlights the practical application of linked lists but also underscores their capability to maintain order and structure in real-world scenarios, delivering a robust and elegant solution for inventory control.

Problem-Solving Approach

The inventory management system consists of two primary classes:

- **Device Class:** Represents individual inventory items as nodes in the linked list.

```java
public class Device {
    private String name;
     private String type;
     private String value;
     private Integer count;
     public Device next; // Pointer to the next node in the linked list

     public Device(String n, String t, String v, Integer c) {
         this.name = n;
         this.type = t;
         this.value = v;
         this.count = c;
         this.next = null; // Initialize the next pointer to null
     }

```

- **Inventory Class:** Manages the linked list of Device nodes and handles operations like addition, deletion, and printing.

```java
public class Inventory implements InventoryInterface {
    private Device head;

    public Inventory() {
        head = null;
    }

```

The linked list approach ensures dynamic memory allocation, and the sorting logic embedded in operations maintains the order of items.

Key Requirements and Features:

1. **Adding Components:** Involves searching for existing items and either updating the count or inserting new nodes at the correct position.

```java

    public void addDevice(String name, String value, String type, Integer count) {
        Device existingDevice = findDevice(name, type, value);
        if (existingDevice != null) {
            existingDevice.setCount(existingDevice.getCount() + count);
        } else {
            Device newDevice = new Device(name, type, value, count);
            if (head == null || head.getName().compareTo(name) > 0) {
                newDevice.next = head;
                head = newDevice;
            } else {
                Device current = head;
                while (current.next != null && current.next.getName().compareTo(name) < 0) {
                    current = current.next;
                }
                newDevice.next = current.next;
                current.next = newDevice;
            }
        }
    }

```

2. **Deleting Components**: Removes or updates nodes based on availability while maintaining the order.

```
84
85      private int deleteDevice(String name, String value, String type, Integer count) {
86          Device current = head;
87          Device previous = null;
88
89          while (current != null) {
90              if (current.getName().equals(name) && current.getType().equals(type) &&
91                  current.getValue().equals(value)) {
92                  if (current.getCount() >= count) {
93                      current.setCount(current.getCount() - count);
94                      if (current.getCount() == 0) {
95                          if (previous == null) {
96                              head = current.next;
97                          } else {
98                              previous.next = current.next;
99                          }
100                     }
101                     return current.getCount();
102                 } else {
103                     return -1;
104                 }
105             }
106             previous = current;
107             current = current.next;
108         }
109         return -1;
110     }
111
```

3. **Printing Inventory**: Sorts and displays items based on the selected criteria.

```
144
145     // Envanteri yazdırma
146     @Override
147     public void printInventory(boolean sortByCount) {
148         sortInventory(sortByCount);  // Listeyi elle sıralıyoruz (Bubble Sort ile)
149
150         Device current = head;
151         while (current != null) {
152             System.out.println(current.getName() + " - Type: " + current.getType() +
153                                ", Value: " + current.getValue() + ", Count: " + current.getCount());
154             current = current.next;
155         }
156     }
157
```

Algorithm Explanation

Addition of Components

1. Search the linked list for an existing Device with matching properties.
2. If found, update the count.

3. If not found, create a new Device node and insert it while maintaining the order based on:
    - Alphabetical order of name when sortByCount = false.
    - Numerical order of count when sortByCount = true.

```
111
112     // Bağlı listeyi seçilen sıraya göre sıralayan metot (Elle Sıralama - Bubble Sort)
113     private void sortInventory(boolean sortByCount) {
114         if (head == null || head.next == null) return;
115
116         boolean swapped;
117         do {
118             swapped = false;
119             Device current = head;
120             Device prev = null;
121
122             while (current.next != null) {
123                 Device next = current.next;
124                 boolean condition = sortByCount ? (current.getCount() > next.getCount())
125                                                 : (current.getName().compareTo(next.getName()) > 0);
126
127                 if (condition) {
128                     // Swap işlemi
129                     if (prev != null) {
130                         prev.next = next;
131                     } else {
132                         head = next;
133                     }
134                     current.next = next.next;
135                     next.next = current;
136                     swapped = true;
137                 }
138
139                 prev = current;
140                 current = current.next;
141             }
142         } while (swapped);
143     }
```

Deletion of Components

1. Search for the specific Device in the linked list.
2. If enough quantity is available:
    - Decrease the count.
    - Remove the node if the count becomes zero.
3. If insufficient quantity is available, return -1.

```
 9
10     private Device findDevice(String name, String type, String value) {
11         Device current = head;
12         while (current != null) {
13             if (current.getName().equals(name) &&
14                 current.getType().equals(type) &&
15                 current.getValue().equals(value)) {
16                 return current;
17             }
18             current = current.next;
19         }
20         return null;
21     }
22
```

Printing Inventory

1. Traverse the linked list and extract items.
2. Sort the items based on the selected criteria:
    - **Alphabetically** by name or **Numerically** by count.
3. Print items grouped by type.

```
111
112     // Bağlı listeyi secilen sıraya göre sıralayan metot (Elle Sıralama - Bubble Sort)
113     private void sortInventory(boolean sortByCount) {
114         if (head == null || head.next == null) return;
115
116         boolean swapped;
117         do {
118             swapped = false;
119             Device current = head;
120             Device prev = null;
121
122             while (current.next != null) {
123                 Device next = current.next;
124                 boolean condition = sortByCount ? (current.getCount() > next.getCount())
125                                                 : (current.getName().compareTo(next.getName()) > 0);
126
127                 if (condition) {
128                     // Swap işlemi
129                     if (prev != null) {
130                         prev.next = next;
131                     } else {
132                         head = next;
133                     }
134                     current.next = next.next;
135                     next.next = current;
136                     swapped = true;
137                 }
138
139                 prev = current;
140                 current = current.next;
141             }
142         } while (swapped);
143     }
```

Implementation Details

The project is implemented in the **Eclipse IDE** using Java. The code structure adheres to the rules of object-oriented programming and includes the following key elements:

Device Class

The Device class encapsulates details of an inventory item, including its name, type, value, count, and a reference to the next node in the linked list.

```
 2
 3 public class Device {
 4     private String name;
 5      private String type;
 6      private String value;
 7      private Integer count;
 8      public Device next; // Pointer to the next node in the linked list
 9
10      public Device(String n, String t, String v, Integer c) {
11          this.name = n;
12          this.type = t;
13          this.value = v;
14          this.count = c;
15          this.next = null; // Initialize the next pointer to null
16      }
17
```

Inventory Class

Implements the InventoryInterface and provides:

- addResistor, addCapacitor, addInductor, and addTransistor methods.

```
22
23      // Cihaz ekleme metotları
24      @Override
25      public void addResistor(String val, Integer cnt) {
26          addDevice("Resistor", val, "-", cnt);
27      }
28
29      @Override
30      public void addCapacitor(String val, String typ, Integer cnt) {
31          addDevice("Capacitor", val, typ, cnt);
32      }
33
34      @Override
35      public void addInductor(String val, Integer cnt) {
36          addDevice("Inductor", val, "-", cnt);
37      }
38
39      @Override
40      public void addTransistor(String typ, Integer cnt) {
41          addDevice("Transistor", "-", typ, cnt);
42      }
43
```

- deleteResistor, deleteCapacitor, deleteInductor, and deleteTransistor methods.

```
63
64      // Cihaz silme metotları
65      @Override
66      public int deleteResistor(String val, Integer cnt) {
67          return deleteDevice("Resistor", val, "-", cnt);
68      }
69
70      @Override
71      public int deleteCapacitor(String val, String typ, Integer cnt) {
72          return deleteDevice("Capacitor", val, typ, cnt);
73      }
74
75      @Override
76      public int deleteInductor(String val, Integer cnt) {
77          return deleteDevice("Inductor", val, "-", cnt);
78      }
79
80      @Override
81      public int deleteTransistor(String typ, Integer cnt) {
82          return deleteDevice("Transistor", "-", typ, cnt);
83      }
84
```

- A printInventory method that outputs the inventory in the specified order.

```
144
145     // Envanteri yazdırma
146     @Override
147     public void printInventory(boolean sortByCount) {
148         sortInventory(sortByCount);  // Listeyi elle sıralıyoruz (Bubble Sort ile)
149
150         Device current = head;
151         while (current != null) {
152             System.out.println(current.getName() + " - Type: " + current.getType() +
153                                ", Value: " + current.getValue() + ", Count: " + current.getCount());
154             current = current.next;
155         }
156     }
157
```

Results

Output Examples

- **Adding and Printing Inventory Sorted by Name:**

```
Inventory sorted by name:
Capacitor - Type: Polar, Value: 2K, Count: 20
Inductor - Type: -, Value: BC128, Count: 45
Resistor - Type: -, Value: 4K, Count: 20
Resistor - Type: -, Value: 2K, Count: 40
Transistor - Type: BC415, Value: -, Count: 45
Transistor - Type: BC128, Value: -, Count: 90
```

- **Adding and Printing Inventory Sorted by Count:**

```
Inventory sorted by count:
Capacitor - Type: Polar, Value: 2K, Count: 20
Resistor - Type: -, Value: 4K, Count: 20
Resistor - Type: -, Value: 2K, Count: 40
Inductor - Type: -, Value: BC128, Count: 45
Transistor - Type: BC415, Value: -, Count: 45
Transistor - Type: BC128, Value: -, Count: 90
```

- **Deleting Components:**

```
There are not enough 2.5K resistors you want to delete left in the inventory!

There are 17 2K resistors left in the inventory.
```

Performance Analysis

Sorting and managing the linked list dynamically ensures that operations remain efficient. Traversal for insertion and deletion operations is **O(n^2)** in complexity, where **n** is the number of nodes in the list.

```
112     // Bağlı listeyi seçilen sıraya göre sıralayan metot (Elle Sıralama - Bubble Sort)
113     private void sortInventory(boolean sortByCount) {
114         if (head == null || head.next == null) return;
115 
116         boolean swapped;
117         do {
118             swapped = false;
119             Device current = head;
120             Device prev = null;
121 
122             while (current.next != null) {
123                 Device next = current.next;
124                 boolean condition = sortByCount ? (current.getCount() > next.getCount())
125                                                 : (current.getName().compareTo(next.getName()) > 0);
126 
127                 if (condition) {
128                     // Swap işlemi
129                     if (prev != null) {
130                         prev.next = next;
131                     } else {
132                         head = next;
133                     }
134                     current.next = next.next;
135                     next.next = current;
136                     swapped = true;
137                 }
138 
139                 prev = current;
140                 current = current.next;
141             }
142         } while (swapped);
143     }
```

Challenges Faced

1. Implementing the sorting logic for a linked list required careful handling of node pointers.
2. Ensuring the order of insertion did not compromise the efficiency of the program.

Conclusion

The inventory management system successfully demonstrates the application of linked lists in managing and sorting data dynamically. The modular implementation ensures scalability and clarity. By adhering to the project guidelines, this system is efficient, user-friendly, and reliable.