Linked List Hands On Day - 2

1. Creation of Doubly linked list

```
package user.hands_on;

import user.collection.DoublyLinkedListADT;

import java.util.Scanner;

public class Qn_1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        DoublyLinkedListADT<Integer> list = new DoublyLinkedListADT<>();
        System.out.println("Enter the element of the Doubly linked List: (End with -1)");
        while(true) {
            int val = sc.nextInt();
            if(val == -1) {
                break;
            }
            list.add(val);
        }
        System.out.print("The Doubly Linked List is:
        ");
        list.display();
    }
}
```

Output

```
Enter the element of the Doubly linked List: (End with -1)
1 2 3 4 5 -1
The Doubly Linked List is:
        Head <--> 1 <--> 2 <--> 3 <--> 4 <--> 5 <--> Null
```

---

2. Length of doubly linked list

```
package user.hands_on;

import user.collection.DoublyLinkedListADT;

import java.util.Scanner;

public class Qn_2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        DoublyLinkedListADT<Integer> list = new DoublyLinkedListADT<>();
        System.out.println("Enter the element of the Doubly linked List: (End with -1)");
        while(true) {
            int val = sc.nextInt();
            if(val == -1) {
                break;
            }
            list.add(val);
        }
        System.out.print("The Length of Doubly Linked List is: " + list.size());
    }
}
```

Output

```
Enter the element of the Doubly linked List: (End with -1)
1 2 3 4 5 -1
The Length of Doubly Linked List is: 5
```

---

3. Insert Node at Specific Position in Linked List

```
package user.hands_on;

import user.collection.DoublyLinkedListADT;

import java.util.Scanner;

public class Qn_3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        DoublyLinkedListADT<Integer> list = new DoublyLinkedListADT<>();
        System.out.println("Enter the number of querys: ");
        int t = sc.nextInt();
        while (t-- > 0) {
            System.out.println("Enter the element of the Doubly linked List: (End with -1)");
            while(true) {
                int val = sc.nextInt();
                if(val == -1) {
                    break;
                }
                list.add(val);
            }
            list.display();
            System.out.println("Enter the element and position of the node: ");
            int ele = sc.nextInt();
            int pos = sc.nextInt();
            list.add(ele, pos);
            System.out.println("After insertion: ");
            list.display();

        }

    }
}
```

Output

```
Enter the number of querys:
1
Enter the element of the Doubly linked List: (End with -1)
1 2 3 4 6 7
-1
Head <--> 1 <--> 2 <--> 3 <--> 4 <--> 6 <--> 7 <--> Null
Enter the element and position of the node:
5 4
After insertion:
Head <--> 1 <--> 2 <--> 3 <--> 4 <--> 5 <--> 6 <--> 7 <--> Null
```

4. Insert in Middle of Linked List

```java
package user.hands_on;

import user.collection.DoublyLinkedListADT;

import java.util.Scanner;

public class Qn_4 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        DoublyLinkedListADT<Integer> list = new DoublyLinkedListADT<>();
        System.out.println("Enter the number of querys: ");
        int t = sc.nextInt();
        while (t-- > 0) {
            System.out.println("Enter the element of the Doubly linked List: (End with -1)");
            while(true) {
                int val = sc.nextInt();
                if(val == -1) {
                    break;
                }
                list.add(val);
            }
            list.display();
            System.out.println("Enter the element need to be inserted: ");
            int pos = list.size() / 2;
            int ele = sc.nextInt();
            list.add(ele, pos);
            System.out.println("After insertion: ");
            list.display();

        }
    }
}
```

Output

```
Enter the number of querys:
1
Enter the element of the Doubly linked List: (End with -1)
1 2 4 5
-1
Head <--> 1 <--> 2 <--> 4 <--> 5 <--> Null
Enter the element need to be inserted:
3
After insertion:
Head <--> 1 <--> 2 <--> 3 <--> 4 <--> 5 <--> Null
```

5. Insertion In Doubly Linked List

```java
package user.hands_on;

import user.collection.DoublyLinkedListADT;

import java.util.Scanner;

public class Qn_5 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        DoublyLinkedListADT<Integer> list = new DoublyLinkedListADT<>();
        System.out.println("Enter the number of querys: ");
        int t = sc.nextInt();
        while (t-- > 0) {
            System.out.println("Enter the element of the Doubly linked List: (End with -1)");
            while (true) {
                int val = sc.nextInt();
                if (val == -1) {
                    break;
                }
                list.add(val);
            }
            list.display();
            System.out.println("Enter the element and position of the node: ");
            int ele = sc.nextInt();
            int pos = sc.nextInt();
            list.add(ele, pos);
            System.out.println("After insertion: ");
            list.display();
        }
    }
}
```

Output

```
Enter the number of querys:
1
Enter the element of the Doubly linked List: (End with -1)
1 2 3 4 5 -1
Head <--> 1 <--> 2 <--> 3 <--> 4 <--> 5 <--> Null
Enter the element and position of the node:
4 4
After insertion:
Head <--> 1 <--> 2 <--> 3 <--> 4 <--> 4 <--> 5 <--> Null
```

---

6. Creation of Circular linked list

```
package user.hands_on;

import user.collection.CircularLinkedListADT;

import java.util.Scanner;

public class Qn_6 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CircularLinkedListADT<Integer> list = new CircularLinkedListADT<>();
        System.out.println("Enter the elements in the list: (Ends with -1) ");
        while (true) {
            int val = sc.nextInt();
            if (val == -1)
                break;
            list.add(val);
        }
        sc.close();
        list.display();
    }
}
```

Output

```
Enter the elements in the list: (Ends with -1)
1 2 3 4 -1
Head -> 1 -> 2 -> 3 -> 4 -> Head
```

7. Length of circular linked list

```
package user.hands_on;

import user.collection.CircularLinkedListADT;

import java.util.Scanner;

public class Qn_7 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CircularLinkedListADT<Integer> list = new CircularLinkedListADT<>();
        System.out.println("Enter the element of the Doubly linked List: (End with -1)");
        while(true) {
            int val = sc.nextInt();
            if(val == -1) {
                break;
            }
            list.add(val);
        }
        System.out.print("The Length of Doubly Linked List is: " + list.size());
    }
}
```

Output

```
Enter the element of the Doubly linked List: (End with -1)
1 2 3 4 5 -1
The Length of Doubly Linked List is: 5
```

8. Delete at beginning

```
package user.hands_on;

import user.collection.CircularLinkedListADT;

import java.util.Scanner;

public class Qn_8 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CircularLinkedListADT<Integer> list = new CircularLinkedListADT<>();
        System.out.println("Enter the element of the Doubly linked List: (End with -1)");
        while(true) {
            int val = sc.nextInt();
            if(val == -1) {
                break;
            }
            list.add(val);
        }
        System.out.println("Before deleting");
        list.display();
        System.out.println("After deleting");
        list.remove(0);
        list.display();
    }
}
```

Output

```
Enter the element of the Doubly linked List: (End with -1)
1 2 3 4 5 -1
Before deleting
Head -> 1 -> 2 -> 3 -> 4 -> 5 -> Head
After deleting
Head -> 2 -> 3 -> 4 -> 5 -> Head
```

---

9. Delete a node in circular linked list

```
package user.hands_on;

import user.collection.CircularLinkedListADT;

import java.util.Scanner;

public class Qn_9 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CircularLinkedListADT<Integer> list = new CircularLinkedListADT<>();
        System.out.println("Enter the element of the Doubly linked List: (End with -1)");
        while(true) {
            int val = sc.nextInt();
            if(val == -1) {
                break;
            }
            list.add(val);
        }
        list.display();
        System.out.println("Enter the position need to be deleted: ");
        list.remove(sc.nextInt());
        list.display();
    }
}
```

Output

```
Enter the element of the Doubly linked List: (End with -1)
1 2 3 4 5 -1
Head -> 1 -> 2 -> 3 -> 4 -> 5 -> Head
Enter the position need to be deleted:
4
Head -> 1 -> 2 -> 3 -> 5 -> Head
```

---

10. Delete At the End

```
package user.hands_on;

import user.collection.CircularLinkedListADT;

import java.util.Scanner;

public class Qn_10 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CircularLinkedListADT<Integer> list = new CircularLinkedListADT<>();
        System.out.println("Enter the element of the Doubly linked List: (End with -1)");
        while (true) {
            int val = sc.nextInt();
            if (val == -1) {
                break;
            }
            list.add(val);
        }
        System.out.println("Before Deletion");
        list.display();
        System.out.println("After deletion");
        list.remove();
        list.display();
    }
}
```

Output

```
Enter the element of the Doubly linked List: (End with -1)
1 2 3 4 5 -1
Before Deletion
Head -> 1 -> 2 -> 3 -> 4 -> 5 -> Head
After deletion
Head -> 1 -> 2 -> 3 -> 4 -> Head
```

CircularLinkedListADT.java

```
package user.collection;

class Node<T extends Comparable<T>> {
    T data;
    Node<T> next;
    public Node(T data) {
        this.data = data;
        this.next = null;
    }
}

public class CircularLinkedListADT<T extends Comparable<T>> {
    private Node<T> head;

    public CircularLinkedListADT() {
        head = null;
    }

    public boolean isEmpty() {
        return head == null;
    }

//     insertion of the Node
//    **************************************

    public void add(T data) {
        Node<T> newNode = new Node<>(data);
        if (isEmpty()) {
            head = newNode;
            head.next = head;
        } else {
            Node<T> current = head;
            while (current.next != head) {
                current = current.next;
            }
            newNode.next = head;
            current.next = newNode;
        }
    }

    public void add(T data, int index) {
        Node<T> newNode = new Node<>(data);
        if (head == null) {
            head = newNode;
            head.next = head;
        }
        if (index == 0) {
            newNode.next = head;
            Node<T> current = head;
            while (current.next != head) {
                current = current.next;
            }
            current.next = newNode;
            head = newNode;
        } else {
            Node<T> current = head;
            for (int i = 1; current.next != head && i < index - 1; i++) {
                current = current.next;
            }
            newNode.next = current.next;
            current.next = newNode;
        }
    }

//    **************************************


//    Displaying the list
    public void display() {
        Node<T> current = head;
        System.out.print("Head -> ");
        while (current.next != head) {
            System.out.print(current.data + " -> ");
            current = current.next;
        }
        System.out.println(current.data + " -> Head");
    }

    public int size() {
        Node<T> current = head;
        int size = 0;
        while (current.next != head) {
            current = current.next;
            size++;
        }
        return size+1;
    }

//    removing the node
//    ********************************************

    public T remove() {
        if (isEmpty()) {
            return null;
        } else {
            Node<T> current = head;
            while (current.next.next != head) {
                current = current.next;
            }
            T data = current.next.data;
            current.next = head;
            return data;
        }
    }

    public T remove(int index) {
        if (isEmpty()) {
            return null;
        }
        if (index == 0) {
            Node<T> current = head;
            head = head.next;
            Node<T> temp = head;
            while (temp.next != current) {
                temp = temp.next;
            }
            temp.next = head;
            return current.data;
        }
        else {
            Node<T> current = head;
            Node<T> prev = head;
            for (int i = 1; current.next != head && i < index; i++) {
                prev = current;
                current = current.next;
            }
            T data = current.data;
            prev.next = current.next;
            return data;
        }
    }

//    *********************************************

//    Searching
    public int indexOf(T data) {
        Node<T> current = head;
        int count = 0;
        while (current.next != head) {
            if (current.data.equals(data)) {
                return count;
            }
            count++;
            current = current.next;
        }
        return -1;
    }


//    reversing the list
    public void reverse()
    {
        if (head == null)
            return;
        Node<T> prev = null;
        Node<T> current = head;
        Node<T> next;
        do {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        } while (current != head);

        head.next = prev;
        head = prev;
    }

//    Sorting
    public void swap(Node<T> ptr1, Node<T> ptr2 ) {
        T tmp = ptr1.data;
        ptr1.data = ptr2.data;
        ptr2.data = tmp;
    }
    public void sort() {
        boolean swapped;
        do {
            swapped = false;
            Node<T> current = head;
            while (current.next != head) {
                if (current.data.compareTo(current.next.data) > 0) {
                    swap(current, current.next);
                    swapped = true;
                }
                current = current.next;
            }
        } while (swapped);
    }


//    merge
    public void merge(CircularLinkedListADT<T> other) {
        Node<T> current = head;
        while (current.next != head) {
            current = current.next;
        }
        current.next = other.head;
        current = other.head;
        while (current.next != other.head) {
            current= current.next;
        }
        current.next = head;
    }

//    Contains
    public boolean contains(T data) {
        Node<T> current = head;
        while (current.next != head) {
            if (current.data.compareTo(data) == 0) {
                return true;
            }
        }
        return false;
    }

}
```

DoublyLinkedListADT.java

```
package user.collection;

class Nodes<T extends Comparable<T>> {
    Nodes<T> prev;
    T data;
    Nodes<T> next;

    public Nodes(T data) {
        this.data = data;
        this.prev = null;
        this.next = null;
    }
}

public class DoublyLinkedListADT<T extends Comparable<T>> {
    Nodes<T> head;

    public DoublyLinkedListADT() {
        head = null;
    }

    public boolean isEmpty() {
        return head == null;
    }

/*    Addition of the Nodes
 *    *************************************
 */
    public void add(T data) {
        Nodes<T> newNode = new Nodes<T>(data);
        if (isEmpty()) {
            head = newNode;
        } else {
            Nodes<T> temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
            newNode.prev = temp;
        }
    }

    public void add(T data, int index) {
        Nodes<T> newNode = new Nodes<T>(data);
        if (isEmpty()) {
            head = newNode;
        } else {
            if (index == 0) {
                newNode.next = head;
                head.prev = newNode;
                head = newNode;
            } else {
                Nodes<T> temp = head;
                for (int i = 1; temp.next != null && i < index; i++) {
                    temp = temp.next;
                }
                newNode.next = temp.next;
                temp.next = newNode;
                newNode.prev = temp;
            }
        }
    }


//     **************************************


/*    Deletion of the Node
 *    *************************************
 */
    public T remove() {
        if (isEmpty()) {
            return null;
        } else {
            Nodes<T> temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.prev.next = null;
            temp.prev = null;
            return temp.data;
        }
    }

    public T remove(int index) {
        if (index == 0) {
            T val = head.data;
            head = head.next;
            head.prev = null;
            return val;
        } else {
            Nodes<T> temp = head;
            for (int i = 0; temp.next != null && i < index; i++) {
                temp = temp.next;
            }
            temp.prev.next = temp.next;
            temp.prev = null;
            return temp.data;
        }
    }

    public void removeVal(T data) {
        if (head == null) {
            return;
        }
        if (head.data.compareTo(data) == 0){
            head = head.next;
            head.prev = null;
        }  else {
            Nodes<T> temp = head;
            while (temp.next != null) {
                if (temp.data.compareTo(data) == 0) {
                    temp.next.prev = temp.prev;
                    temp.prev.next = temp.next;
                    break;
                }
                temp = temp.next;
            }
            if (temp.next != null) {
                if (temp.data.compareTo(data) == 0) {
                    temp.prev.next = null;
                }
            }
        }
        if (contains(data)) {
            removeVal(data);
        }
    }
//     **************************************

/*    Displaying the Doubly Linked List
*     *************************************
*/
    public void display() {
        if (isEmpty()) {
            System.out.println("List is empty");
        } else {
            Nodes<T> temp = head;
            System.out.print("Head <--> ");
            while (temp != null) {
                System.out.print(temp.data + " <--> ");
                temp = temp.next;
            }
            System.out.println("Null");
        }
    }
//     **************************************

/*    Checking for the values
 *    *************************************
 */
    public boolean contains(T data) {
        if (head == null) {
            return false;
        } else {
            Nodes<T> temp = head;
            while (temp != null) {
                if (temp.data.compareTo(data) == 0) {
                    return true;
                }
                temp = temp.next;
            }
        }
        return false;
    }

    public int size() {
        if(isEmpty()) {
            return 0;
        } else {
            Nodes<T> temp = head;
            int count = 0;
            while (temp != null) {
                temp = temp.next;
                count++;
            }
            return count;
        }
    }

//    Sort Linked List
    void swap(Nodes<T> ptr1, Nodes<T> ptr2) {
        T tmp = ptr2.data;
        ptr2.data = ptr1.data;
        ptr1.data = tmp;
    }

    public void sort() {
        if (head == null || head.next == null) {
            return;
        }
        boolean swapped;
        do {
            Nodes<T> current = head;
            swapped = false;

            while (current != null && current.next != null) {
                if (current.data.compareTo(current.next.data) > 0) {
                    swap(current, current.next);
                    swapped = true;
                }
                current = current.next;
            }
        } while (swapped);
    }

//    Reverse Linked List

    public void reverse() {
        Nodes<T> temp = null;
        Nodes<T> current = head;
        while (current != null) {
            temp = current.prev;
            current.prev = current.next;
            current.next = temp;
            current = current.prev;
        }
        if (temp != null) {
            head = temp.prev;
        }
    }


//    Merging Doubly Linked List
    public void merge(DoublyLinkedListADT<T> list2) {
        Nodes<T> temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = list2.head;
        list2.head.prev = temp;
    }

    public int search(T dat) {
        Nodes<T> temp = head;
        int count = 0;
        while (temp.next != null) {
            if (temp.data.equals(dat)) {
                return count;
            }
            temp = temp.next;
            count+=1;
        }
        return -1;
    }

//    Union of List
    public void union(DoublyLinkedListADT<T> head1, DoublyLinkedListADT<T> head2) {
        Nodes<T> t1 = head1.head, t2 = head2.head;
        while (t1 != null) {
            push(t1.data);
            t1 = t1.next;
        }
        while (t2 != null) {
            if (!contains(t2.data))
                push(t2.data);
            t2 = t2.next;
        }
    }

    void push(T new_data) {
        Nodes<T> new_node = new Nodes<>(new_data);
        new_node.next = head;
        head = new_node;
    }

}
```