Name: Kamran khan

Section: B

CMS_ID: 023-23-0372

Assignment: Assignment 1

1. Write a function to get the nth node from the end of the linked list. Function name: int nthFromLast(int n); Case-1: (List Empty) Head=Null then return LIST_EMPTY Case-2: (List Non-Empty) Head != Null then return nth element from the end of list Example Input: 10 -> 20 -> 30 -> 40 -> 50, n = 2 Output: 40 (From the last, second node conatins the data 40)

```
public class LinkedList {
    private static final int LIST_EMPTY = 0;
    static Node head;

    public static int nthFromLast(Node head, int n) {

        if (head == null) {
            return LIST_EMPTY;
        }


        int length = 0;
        Node current = head;
        while (current != null) {
            length++;
            current = current.next;
        }

        // Check if n is valid
        if (n > length) {
```

```
            return LIST_EMPTY; // Invalid input
        }

        // Move to the (length - n)th node from the start
        current = head;
        for (int i = 0; i < length - n; i++) {
            current = current.next;
        }

        return current.data; // Return the value of the nth node from the end
    }

    public static void main(String[] args) {
        // Create a sample linked list: 10 -> 20 -> 30 -> 40 -> 50
        Node head = new Node(10);
        head.next = new Node(20);
        head.next.next = new Node(30);
        head.next.next.next = new Node(40);
        head.next.next.next.next = new Node(50);

        int n = 2;
        System.out.println("Nth node from end is: " + nthFromLast(head, n));
    }
}
```

```
[Running] cd "c:\Users\DELL\Desktop\DSA progrm\" && javac LinkedList.java && java LinkedList
Nth node from end is: 40

[Done] exited with code=0 in 2.42 seconds
```

2. Write a function to sort the given single linked list. (Don’t swap the data present in the nodes, swap the nodes itself.) Function name: void sort(); Case-1: (List Empty) Head=Null then return LIST_EMPTY Case-2: (List Non-Empty) Head != Null then swap the nodes to sort them Example Input: 50 -> 40 -> 30 -> 20 -> 10 Output: 10 -> 20 -> 30 -> 40 -> 50

```
public class Task2 {


   static final int LIST_EMPTY = -1;


   static class Node {
      int data;
      Node next;


      Node(int data) {
         this.data = data;
         this.next = null;
      }
   }


   Node head;


   public int sort() {
      if (head == null) {
         return LIST_EMPTY;
      }


      boolean swapped;
      Node ptr1;
      Node lptr = null;


      do {
```

```
        swapped = false;
        ptr1 = head;

        while (ptr1.next != lptr) {
            if (ptr1.data > ptr1.next.data) {
                // Swap nodes
                swapNodes(ptr1, ptr1.next);
                swapped = true;
            }
            ptr1 = ptr1.next;
        }
        lptr = ptr1;
    } while (swapped);

    return 0;
}

private void swapNodes(Node node1, Node node2) {
    int temp = node1.data;
    node1.data = node2.data;
    node2.data = temp;
}

public void add(int data) {
    Node newNode = new Node(data);
    if (head == null) {
        head = newNode;
    } else {
        Node temp = head;
```

```
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
    }
}

public void printList() {
    Node temp = head;
    while (temp != null) {
        System.out.print(temp.data + " -> ");
        temp = temp.next;
    }
    System.out.println("null");
}

public static void main(String[] args) {
    Task2 list = new Task2();
    list.add(50);
    list.add(40);
    list.add(30);
    list.add(20);
    list.add(10);

    System.out.println("Before sorting:");
    list.printList();

    list.sort();
```

```
        System.out.println("After sorting:");
        list.printList();
    }

}
```

```
[Running] cd "c:\Users\DELL\Desktop\DSA progrm\
Before sorting:
50 -> 40 -> 30 -> 20 -> 10 -> null
After sorting:
10 -> 20 -> 30 -> 40 -> 50 -> null

[Done] exited with code=0 in 1.806 seconds
```

3. Write a function to reverse the single linked list. Function name: void reverse(); Case-1: (List Empty) Head=Null then return LIST_EMPTY Case-2: (List Non-Empty) Head != Null then reverse the list Example Input: 50 -> 40 -> 30 -> 20 -> 10 Output: 10 -> 20 -> 30 -> 40 -> 50

```
public class Reversed {
    static final int LIST_EMPTY = -1;

    static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
```

```
private Node head;

public int reverse() {
    if (head == null) {
        return LIST_EMPTY;
    }

    Node previous = null;
    Node current = head;

    while (current != null) {
        Node next = current.next;
        current.next = previous;
        previous = current;
        current = next;
    }

    head = previous;
    return 0;
}

public void add(int data) {
    Node newNode = new Node(data);
    if (head == null) {
        head = newNode;
    } else {
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
```

```
        }
        temp.next = newNode;
    }
}

public void printList() {
    Node temp = head;
    while (temp != null) {
        System.out.print(temp.data + " -> ");
        temp = temp.next;
    }
    System.out.println("null");
}

public static void main(String[] args) {
    Reversed list = new Reversed();
    list.add(50);
    list.add(40);
    list.add(30);
    list.add(20);
    list.add(10);

    System.out.println("Before reversing:");
    list.printList();

    list.reverse();

    System.out.println("After reversing:");
    list.printList();
```

```
    }
}
```

```
Before reversing:
50 -> 40 -> 30 -> 20 -> 10 -> null
After reversing:
10 -> 20 -> 30 -> 40 -> 50 -> null
```

4. Write a function to remove the duplicates data present in the single linked list. Function name: void removeDuplicates(); Case-1: (List Empty) Head=Null then return LIST_EMPTY Case-2: (List Non-Empty) Head != Null then remove duplicate elements Example Input: 5 -> 3 -> 4 -> 5 -> 2 -> 1 -> 4 -> 5 -> 3 Output: 5 -> 3 -> 4 -> 2 -> 1

```
class Task4 {
    Node head;

    void removeDuplicates() {
        if (head == null) {
            System.out.println("LIST_EMPTY");
            return;
        }

        Node current = head;
        Node previous = null;

        while (current != null) {
            if (current.next != null && current.data == current.next.data) {
                previous.next = current.next; // Skip the duplicate
                current = current.next; // Move to the next node
            } else {
```

```
            previous = current; // Move previous to current
            current = current.next; // Move to the next node
        }
    }
}

// Method to print the list
void printList() {
    Node current = head;
    while (current != null) {
        System.out.print(current.data + " -> ");
        current = current.next;
    }
    System.out.println("null");
}

// Example usage
public static void main(String[] args) {
    Task4 list = new Task4();
    list.head = new Node(5);
    list.head.next = new Node(3);
    list.head.next.next = new Node(4);
    list.head.next.next.next = new Node(5);
    list.head.next.next.next.next = new Node(2);
    list.head.next.next.next.next.next = new Node(1);
    list.head.next.next.next.next.next.next = new Node(4);
    list.head.next.next.next.next.next.next.next = new Node(5);
    list.head.next.next.next.next.next.next.next.next = new Node(3);
```

```
        System.out.println("Original List:");
        list.printList();

        list.removeDuplicates();

        System.out.println("List after removing duplicates:");
        list.printList();
    }
}
```