Name: Vaishnavi Ingole

Mail: vaishnavingole54@gmail.com

Task 2: Linked List Middle Element Search You are given a singly linked list. Write a function to find the middle element without using any extra space and only one traversal through the linked list.

Solution:

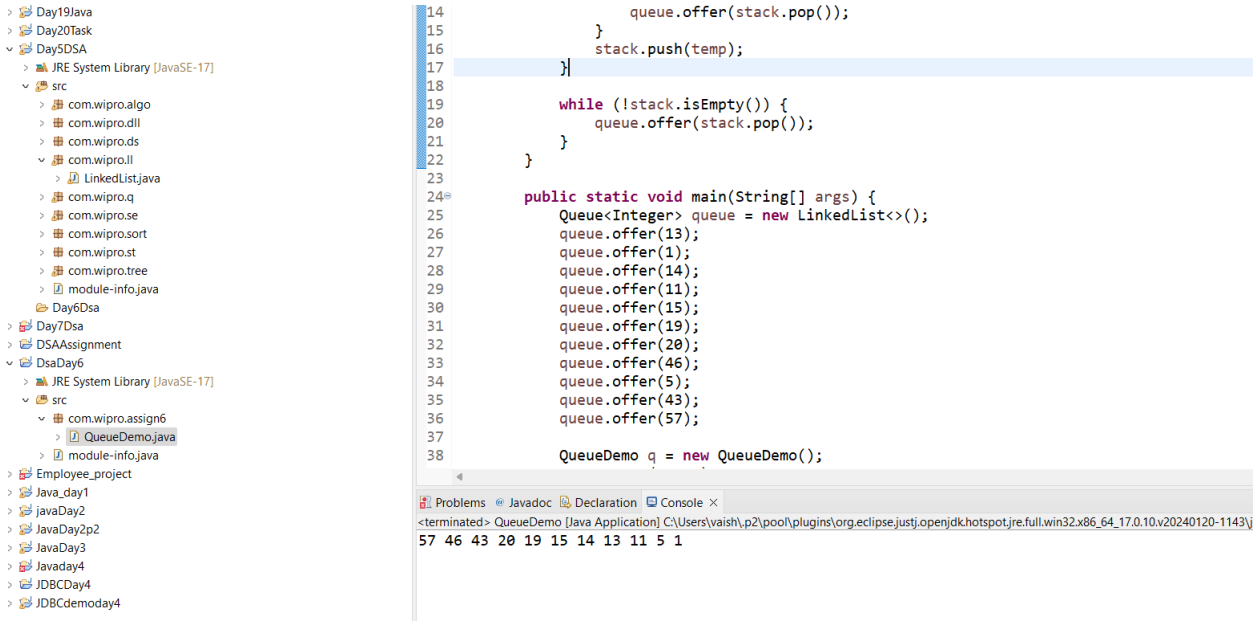
```
package com.assign.dsa;



public class LinkedList {
	class Node{
		int value;
		Node next;

		public Node(int value) {
			super();
			this.value=value;
		}
	}

	private Node head;
	private Node tail;
	private int length;
	public Node getHead() {
		return head;
	}
	public void setHead(Node head) {
		this.head = head;
	}
	public Node getTail() {
		return tail;
	}
	public void setTail(Node tail) {
		this.tail = tail;
	}
	public int getLength() {
		return length;
```

```
	}
	public void setLength(int length) {
		this.length = length;
	}

	public void printList() {
		Node temp=head;
		System.out.println("Head: "+getHead().value);
		System.out.println("Tail: "+getTail().value);
		System.out.println("length: "+getLength());
		while(temp!=null) {
			System.out.print("--->"+temp.value);

			temp=temp.next;
		}
	}
	public void append(int value) {
		Node newNode=new Node(value);
		if(length==0) {
			head=newNode;
			tail=newNode;
		}else {
			tail.next=newNode;
			tail=newNode;
		}
		length++;
	}

private void findMiddle() {

		Node slow=head;
		Node fast=head;

		while(fast!=null && fast.next!=null && slow!=null) {
			fast=fast.next.next;
			slow=slow.next;
	}
		System.out.println("Middle element is:-> "+slow.value);
}
	public LinkedList(int value ) {
		super();
		Node newNode=new Node(value);
		System.out.println("Node :"+newNode);
		head=newNode;
		tail=newNode;
		length=1;
```

```
}

public static void main(String args[]) {
	LinkedList myll =new LinkedList(4);
	System.out.println(myll.head);
	System.out.println(myll.tail);
	System.out.println(myll.length);

	myll.append(5);
	myll.append(6);
	myll.append(7);
	myll.append(8);
	myll.append(9);
	myll.append(10);
	myll.printList();

	myll.findMiddle();


}




}
```

Task 3: Queue Sorting with Limited Space You have a queue of integers that you need to sort. You can only use additional space equivalent to one stack. Describe the steps you would take to sort the elements in the queue.

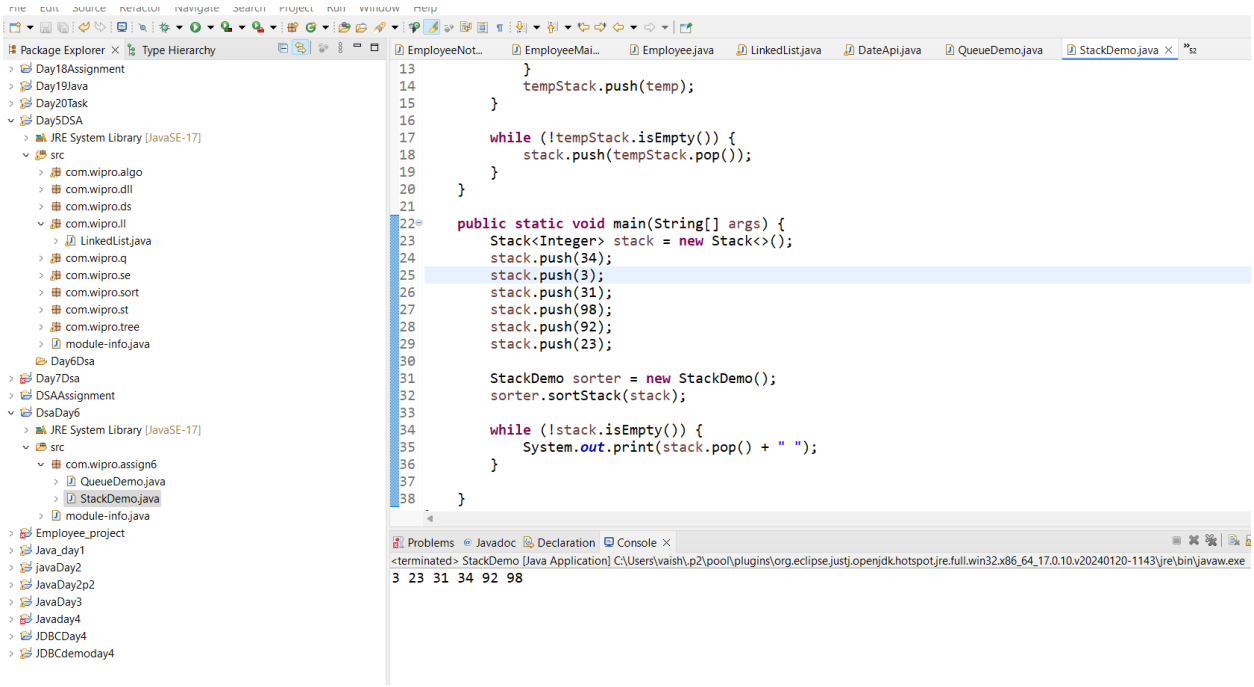
```java
package com.wipro.assign6;

import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class QueueDemo {
	public void sortQueue(Queue<Integer> queue) {
		Stack<Integer> stack = new Stack<>();

		while (!queue.isEmpty()) {
			int temp = queue.poll();
			while (!stack.isEmpty() && stack.peek() > temp) {
				queue.offer(stack.pop());
			}
			stack.push(temp);
		}

		while (!stack.isEmpty()) {
			queue.offer(stack.pop());
		}
	}

	public static void main(String[] args) {
		Queue<Integer> queue = new LinkedList<>();
		queue.offer(13);
		queue.offer(1);
		queue.offer(14);
		queue.offer(11);
		queue.offer(15);
		queue.offer(19);
		queue.offer(20);
		queue.offer(46);
		queue.offer(5);
		queue.offer(43);
		queue.offer(57);

		QueueDemo q = new QueueDemo();
		q.sortQueue(queue);

		while (!queue.isEmpty()) {
			System.out.print(queue.poll() + " ");
		}

	}
}
```



Task 4: Stack Sorting In-Place You must write a function to sort a stack such that the smallest items are on the top. You can use an additional temporary stack, but you may not copy the elements into any other data structure such as an array. The stack supports the following operations: push, pop, peek, and isEmpty.

```
package com.wipro.assign6;

import java.util.Stack;

public class StackDemo {
      public void sortStack(Stack<Integer> stack) {
         Stack<Integer> tempStack = new Stack<>();

         while (!stack.isEmpty()) {
             int temp = stack.pop();
             while (!tempStack.isEmpty() && tempStack.peek() > temp) {
                 stack.push(tempStack.pop());
             }
             tempStack.push(temp);
         }

         while (!tempStack.isEmpty()) {
             stack.push(tempStack.pop());
         }
     }

     public static void main(String[] args) {
         Stack<Integer> stack = new Stack<>();
```

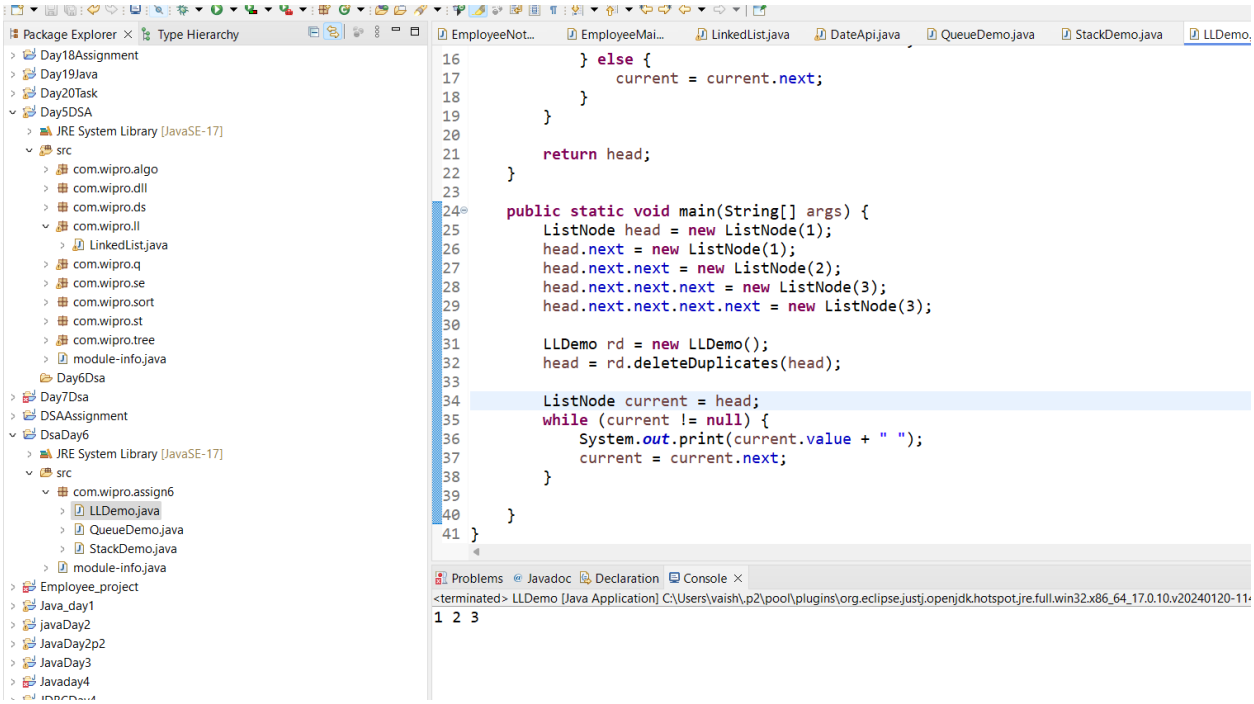
```
        stack.push(34);
        stack.push(3);
        stack.push(31);
        stack.push(98);
        stack.push(92);
        stack.push(23);

        StackDemo sorter = new StackDemo();
        sorter.sortStack(stack);

        while (!stack.isEmpty()) {
            System.out.print(stack.pop() + " ");
        }

    }
}
```



Task 5: Removing Duplicates from a Sorted Linked List A sorted linked list has been constructed with repeated elements. Describe an algorithm to remove all duplicates from the linked list efficiently.

```
package com.wipro.assign6;

class ListNode {
```

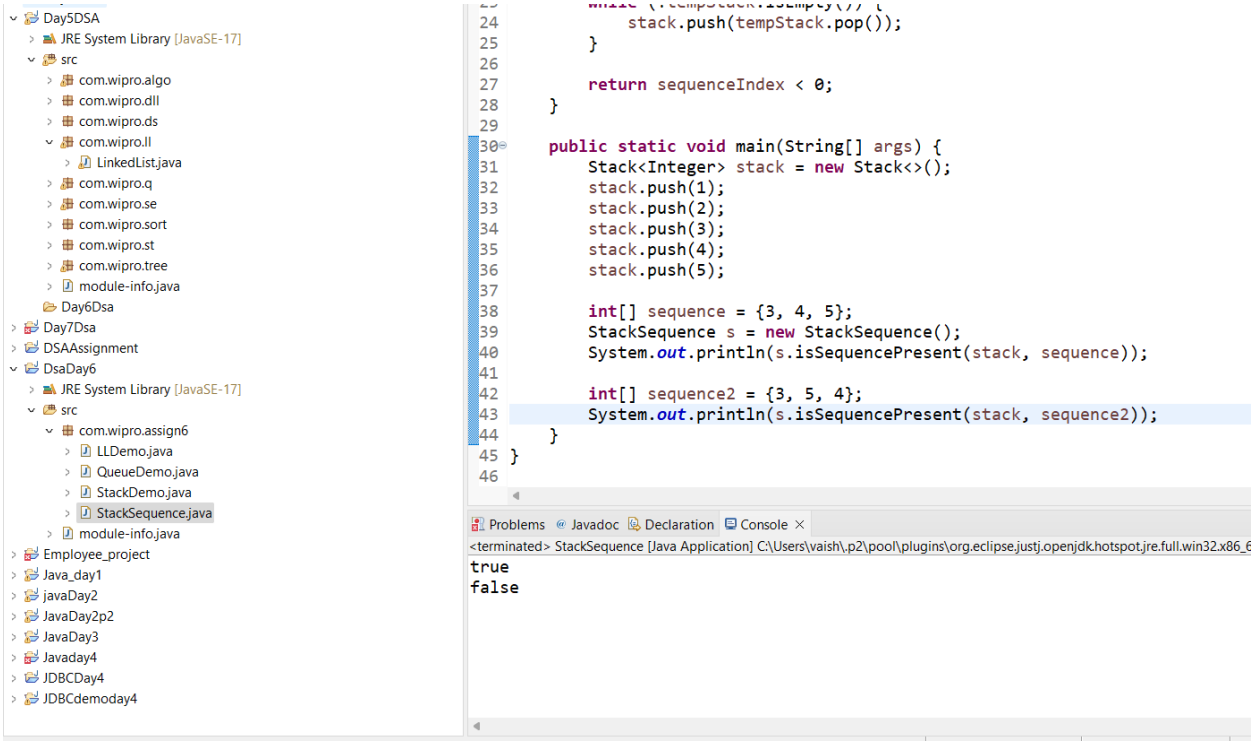
```
    int value;
    ListNode next;
    ListNode(int x) { value = x; }
}

public class LLDemo {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode current = head;

        while (current != null && current.next != null) {
            if (current.value == current.next.value) {
                current.next = current.next.next;
            } else {
                current = current.next;
            }
        }

        return head;
    }

    public static void main(String[] args) {
        ListNode head = new ListNode(1);
        head.next = new ListNode(1);
        head.next.next = new ListNode(2);
        head.next.next.next = new ListNode(3);
        head.next.next.next.next = new ListNode(3);

        LLDemo rd = new LLDemo();
        head = rd.deleteDuplicates(head);

        ListNode current = head;
        while (current != null) {
            System.out.print(current.value + " ");
            current = current.next;
        }

    }
}
```



Task 6: Searching for a Sequence in a Stack Given a stack and a smaller array representing a sequence, write a function that determines if the sequence is present in the stack. Consider the sequence present if, upon popping the elements, all elements of the array appear consecutively in the stack.

```
package com.wipro.assign6;

import java.util.Stack;

public class StackSequence {
      public boolean isSequencePresent(Stack<Integer> stack, int[]
sequence) {
        if (sequence.length == 0) return true;
        if (stack.isEmpty()) return false;

        Stack<Integer> tempStack = new Stack<>();
        int sequenceIndex = sequence.length - 1;

        while (!stack.isEmpty()) {
            int element = stack.pop();
            tempStack.push(element);

            if (element == sequence[sequenceIndex]) {
                sequenceIndex--;
                if (sequenceIndex < 0) break;
```

```
            }
        }

        while (!tempStack.isEmpty()) {
            stack.push(tempStack.pop());
        }

        return sequenceIndex < 0;
    }

    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        stack.push(5);

        int[] sequence = {3, 4, 5};
        StackSequence s = new StackSequence();
        System.out.println(s.isSequencePresent(stack, sequence));

        int[] sequence2 = {3, 5, 4};
        System.out.println(s.isSequencePresent(stack, sequence2));
    }
}
```

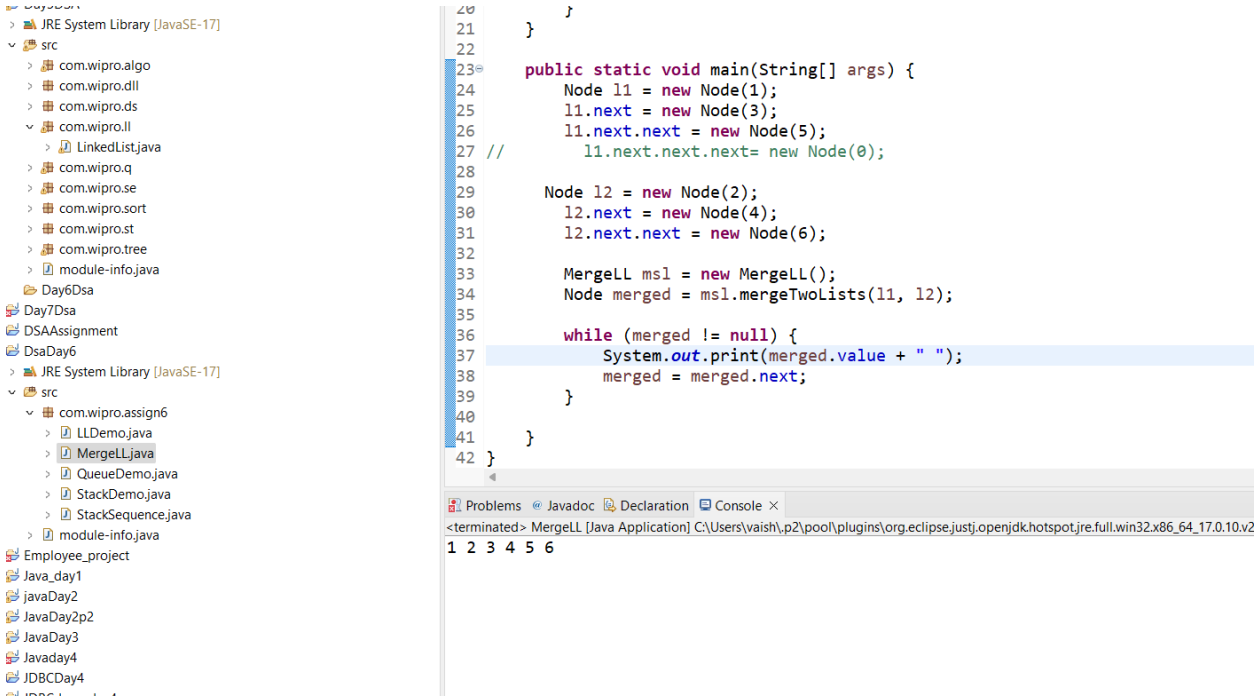


Task 7: Merging Two Sorted Linked Lists You are provided with the heads of two sorted linked lists. The lists are sorted in ascending order. Create a merged linked list in ascending order from the two input lists without using any extra space (i.e., do not**package** com.wipro.assign6;

```
class Node {
    int value;
    Node next;
    Node(int x) { value = x; }
}

public class MergeLL {
    public Node mergeTwoLists(Node l1, Node l2) {
        if (l1 == null) return l2;
        if (l2 == null) return l1;

        if (l1.value < l2.value) {
            l1.next = mergeTwoLists(l1.next, l2);
            return l1;
        } else {
            l2.next = mergeTwoLists(l1, l2.next);
            return l2;
        }
    }
```

```
    public static void main(String[] args) {
        Node l1 = new Node(1);
        l1.next = new Node(3);
        l1.next.next = new Node(5);
//         l1.next.next.next= new Node(0);

       Node l2 = new Node(2);
        l2.next = new Node(4);
        l2.next.next = new Node(6);

        MergeLL msl = new MergeLL();
        Node merged = msl.mergeTwoLists(l1, l2);

        while (merged != null) {
            System.out.print(merged.value + " ");
            merged = merged.next;
        }

    }
}
```

create any new nodes).



Task 8: Circular Queue Binary Search Consider a circular queue (implemented using a fixed-size array) where the elements are sorted but have been rotated at an unknown index. Describe an approach to perform a binary search for a given element within this circular queue.

```
package com.wipro.assign6;

public class CircularQueue {
    public int search(int[] queue, int front, int rear, int target) {
        if (queue == null || queue.length == 0) {
            return -1;
        }

        int n = queue.length;
        int low = 0, high = n - 1;

        while (low <= high) {
            int mid = (low + high) / 2;
            int midValue = queue[(mid + front) % n];

            if (midValue == target) {
                return (mid + front) % n;
            }

            if (queue[(low + front) % n] <= midValue) {
                if (target >= queue[(low + front) % n] && target < 
midValue) {
                    high = mid - 1;
                } else {
                    low = mid + 1;
                }
            } else {
                if (target > midValue && target <= queue[(high + 
front) % n]) {
                    low = mid + 1;
                } else {
                    high = mid - 1;
                }
            }
        }

        return -1;
    }

    public static void main(String[] args) {
        CircularQueue cqs = new CircularQueue();
```

```
        int[] queue = {4, 5, 6, 7, 0, 1, 2};
        int front = 0;
        int rear = 6;
        int target = 0;

        System.out.println(cqs.search(queue, front, rear, target));

        target = 3;
        System.out.println(cqs.search(queue, front, rear, target));
    }
}
```