9.2.8. Delete the Last Node of LinkedList

Lab8.java

```
package com.jlcindia.linkedlist.singly;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Lab8 {

	static Node deleteLast(Node headNode) {

		if (headNode == null)
			return null;

		if(headNode.next==null) {
			return null;
		}

		Node currentNode = headNode;
		while (currentNode.next.next != null) {
			currentNode = currentNode.next;
		}

		currentNode.next = null;
		return headNode;
	}
	static Node insertLast(Node headNode, int data) {

		//Copy from Previous Labs
	}

	static void displayList(Node headNode) {

		 //Copy from Previous Labs
	}
	public static void main(String[] args) {

		Node head = null;
		head = insertLast(head, 10);
		head = insertLast(head, 20);
		head = insertLast(head, 30);
		head = insertLast(head, 40);
		head = insertLast(head, 50);

		displayList(head);
		System.out.println("--------------");

		head = deleteLast(head);
		displayList(head);
		System.out.println("--------------");
		head = deleteLast(head);
		displayList(head);
	}

}
```

9.2.9. Delete the Node at given position of S.L.L

Lab9.java

```
package com.jlcindia.linkedlist.singly;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Lab9 {
	static Node delete(Node headNode, int postion) {

		if (headNode == null)
			return null;

		if (headNode.next == null)
			return null;

		if (postion == 1) {
			return headNode.next;
		}

		Node currentCode = headNode;

		for (int i = 1; i <= postion - 2 && currentCode!=null ; i++) {
			currentCode = currentCode.next;
		}

		if(currentCode==null) {
			return headNode;
		}

		currentCode.next=currentCode.next.next;

		return headNode;

	}
	static Node insertLast(Node headNode, int data) {

		//Copy from Previous Labs
	}

	static void displayList(Node headNode) {

		//Copy from Previous Labs
	}
	public static void main(String[] args) {

		Node head = null;

		head = insertLast(head, 10);
		head = insertLast(head, 20);
		head = insertLast(head, 30);
		head = insertLast(head, 40);
		head = insertLast(head, 50);

		displayList(head);
		System.out.println("--------------");

		head = delete(head, 2);
		displayList(head);
		System.out.println("--------------");
		head = delete(head, 3);
		displayList(head);

	}
}
```

9.2.10. Search the Node in LinkedList

Lab10.java

```
package com.jlcindia.linkedlist.singly;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Lab10 {

        static int search(Node headNode, int element) {

                int position=1;

                Node currentNode = headNode;
                while(currentNode!=null) {
                        if(currentNode.data==element) {
                                return position;
                        }else {
                                currentNode=currentNode.next;
                                position++;
                        }
                }

                return -1;
        }

        static Node insertLast(Node headNode, int data) {

                //Copy from Previous Labs
        }

        static void displayList(Node headNode) {

                 //Copy from Previous Labs
        }
        public static void main(String[] args) {

                Node head = null;

                head = insertLast(head, 10);
                head = insertLast(head, 20);
                head = insertLast(head, 30);
                head = insertLast(head, 40);
                head = insertLast(head, 50);

                displayList(head);
                System.out.println("--------------");

                int x = search(head, 20);
                System.out.println(x);
                System.out.println("--------------");
                x = search(head, 40);
                System.out.println(x);
                System.out.println("--------------");
                x = search(head, 70);
                System.out.println(x);


        }

}
```

9.2.11. Design MySinglyLinkedList

Lab11.java

```
package com.jlcindia.linkedlist.singly;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Node {
        int data;
        Node next;
        Node(int data) {
                this.data = data;
                this.next = null;
        }
}
public class MySinglyLinkedList {

        Node headNode = null;
        int nodeCount = 0;

        public String toString() {
                if (this.headNode == null) {
                        return "[]";
                }

                String str = "[";
                Node currentNode = this.headNode;
                while (currentNode != null) {
                        str = str + "" + currentNode.data + " , ";
                        currentNode = currentNode.next;
                }
                str = str.substring(0, str.length() - 2);
                str = str + "]";
                return str;
        }
```

```
public boolean isEmpty() {

        return nodeCount == 0;
}

public void clear() {

        Node currentNode=headNode;
        while(currentNode!=null) {
                Node temp = currentNode.next;

                currentNode.next=null;
                currentNode=temp;
        }

        this.headNode = null;
        this.nodeCount = 0;
}

public int size() {

        return nodeCount;
}

public void insertFirst(int data) {

        Node temp = new Node(data);
        temp.next = this.headNode;
        this.headNode = temp;
        nodeCount++;

}
```

```
public void insertLast(int data) {

        Node temp = new Node(data);
        if (this.headNode == null) {
                this.headNode = temp;
        }

        // To Reach the Last Node of LL
        Node currentNode = headNode;
        while (currentNode.next != null) {
                currentNode = currentNode.next;
        }

        currentNode.next = temp;
        nodeCount++;
}

public void insert(int position, int data) {

        Node temp = new Node(data);

        if (headNode == null) {
                this.headNode = temp;
                return;
        }

        if (position == 1) {
                temp.next = headNode;
                this.headNode = temp;
                return;
        }

        Node currentNode = headNode;
        for (int i = 1; i <= position - 2 && currentNode != null; i++) {
                currentNode = currentNode.next;
        }
        if (currentNode == null) {
            return;
        }

        temp.next = currentNode.next;
        currentNode.next = temp;

        nodeCount++;
    }

    public void deleteFirst() {

        if (this.headNode == null) {
            return;
        }

        this.headNode = headNode.next;
        nodeCount--;
    }

    public void deleteLast() {

        // When No Nodes
        if (this.headNode == null)
            return;

        // When One Node is there
        if (headNode.next == null) {
            this.headNode = null;
            nodeCount--;
            return;
        }

        // When two or more Nodes are there
        Node currentNode = headNode;
        while (currentNode.next.next != null) {
            currentNode = currentNode.next;
        }

        currentNode.next = null;
        nodeCount--;

    }

    public void delete(int postion) {

        // 1.Empty List
        if (headNode == null) {
            this.headNode = null;
            return;
        }

        // 2.List with One Node
        if (headNode.next == null) {
            this.headNode = null;
            nodeCount--;
            return;
        }
        // 3. List with 2 or more Nodes
        // Deleting 1st Node
        if (postion == 1) {
            this.headNode = headNode.next;
            nodeCount--;
            return;
        }

        // 4. List with 2 or more Nodes
        Node currentCode = headNode;

        for (int i = 1; i <= postion - 2 && currentCode != null; i++) {
            currentCode = currentCode.next;
        }
```

```
                // 5. Position is not Present
                if (currentCode == null) {
                        return;
                }

                currentCode.next = currentCode.next.next;
                nodeCount--;
        }

        public int search(int element) {

                int position = 1;

                Node currentNode = this.headNode;
                while (currentNode != null) {
                        if (currentNode.data == element) {
                                return position;
                        } else {
                                currentNode = currentNode.next;
                                position++;
                        }
                }

                return -1;

        }

}
```

```
public class Lab11 {

	public static void main(String[] args) {

	MySinglyLinkedList mylist=new MySinglyLinkedList();
	System.out.println(mylist);
	System.out.println(mylist.size());


	mylist.insertFirst(11);
	mylist.insertFirst(22);
	mylist.insertFirst(33);

	System.out.println("---------------------");
	System.out.println(mylist);
	System.out.println(mylist.size());

	mylist.insertLast(40);
	mylist.insertLast(50);

	System.out.println("---------------------");
	System.out.println(mylist);
	System.out.println(mylist.size());

	mylist.insert(4, 99);
	mylist.insert(5, 88);

	System.out.println("---------------------");
	System.out.println(mylist);
	System.out.println(mylist.size());

	mylist.deleteFirst();
	System.out.println("---------------------");
	System.out.println(mylist);
	System.out.println(mylist.size());
```

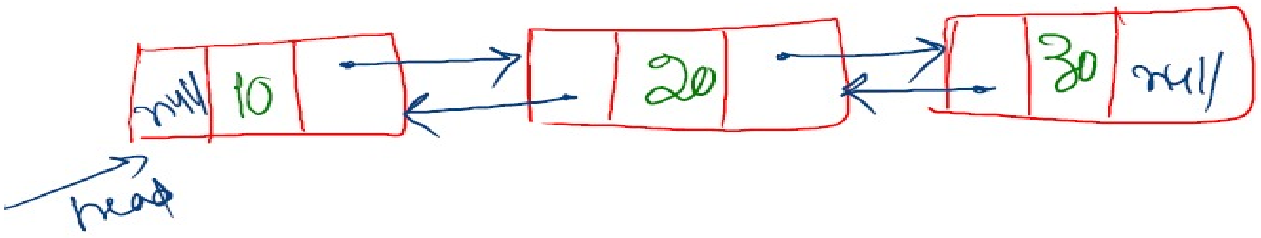
```
        mylist.deleteLast();
        System.out.println("---------------------");
        System.out.println(mylist);
        System.out.println(mylist.size());

        mylist.delete(2);
        System.out.println("---------------------");
        System.out.println(mylist);
        System.out.println(mylist.size());
        System.out.println("---------------------");

        System.out.println(mylist.search(99));
        System.out.println(mylist.search(77));

        System.out.println("---------------------");
        System.out.println(mylist.isEmpty());
        System.out.println(mylist.size());

        mylist.clear();

        System.out.println(mylist.isEmpty());
        System.out.println(mylist.size());

        }

}
```

9.3. Doubly Linked List

- Doubly Linked List consists of a number of nodes
- Each Node has will have 3 parts
  - a) Address of Previous Node
  - b) Data
  - c) Address of Next Node

- Next Address of the Last node in the list is NULL, which indicates that No Next Element.
- Previous Address of the First/Head node in the list is NULL, which indicates that No Previous element.

Ex:



- Following is a type declaration for a linked list:

```
public class Node {

        int data;
        Node next;
        Node prev;

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

- Main Doubly Linked Lists Operations
  - ✓ Traverse: Access the elements in the List
  - ✓ Count: returns the number of elements in the List
  - ✓ Find nth node from the end of the List
  - ✓ Insert: inserts an element into the List
  - ✓ Delete: removes the specified position element from the List
  - ✓ Reverse the List

9.3.1. Traverse the Doubly LinkedList in forward order

Lab12.java

```
package com.jlcindia.linkedlist.doubly;

/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Node {

	int data;
	Node next;
	Node prev;

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

}
```

```
public class Lab12 {

    static String toString(Node headNode) {
        if (headNode == null) {
            return "[]";
        }

        String str = "[";
        Node currentNode = headNode;
        while (currentNode != null) {
            str = str + "" + currentNode.data + " , ";
            currentNode = currentNode.next;
        }
        str = str.substring(0, str.length() - 2);
        str = str + "]";

        return str;
    }


    public static void main(String[] args) {

        Node head = new Node(10);
        Node node2 = new Node(20);
        Node node3 = new Node(30);

        head.next = node2;
        node2.prev = head;

        node2.next = node3;
        node3.prev = node2;

        System.out.println(toString(head));
    }
}
```

9.3.2. Traverse the Doubly LinkedList in Reverse order

Lab13.java

```
package com.jlcindia.linkedlist.doubly;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Lab13 {

        static String toString(Node headNode) {

                if (headNode == null) {
                        return "[]";
                }

                Node tailNode = headNode;

                while(tailNode.next!=null) {
                        tailNode=tailNode.next;
                }

                String str = "[";
               Node currentNode = tailNode;
                while (currentNode != null) {
                        str = str + "" + currentNode.data + " , ";
                        currentNode = currentNode.prev;
                }

                str = str.substring(0, str.length() - 2);
                str = str + "]";

                return str;
        }
```

```
    public static void main(String[] args) {

        Node head = new Node(10);
        Node node2 = new Node(20);
        Node node3 = new Node(30);

        head.next = node2;
        node2.prev = head;

        node2.next = node3;
        node3.prev = node2;

        System.out.println(toString(head));

    }

}
```

9.3.3. Insert the Node at beginning of D.L.L

Lab14.java

```
package com.jlcindia.linkedlist.doubly;

/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Lab14 {

        static Node insertFirst(Node headNode,int data) {

                Node temp = new Node(data);

                //1.Empty List
                if(headNode==null) {
                        headNode=temp; //head node changed
                        return headNode;
                }

                //2.List with 1 or more Nodes
                temp.next=headNode;
                headNode.prev=temp;

                headNode=temp; //head node changed

                return headNode;
        }
        static String toString(Node headNode) {

                if (headNode == null) {
                        return "[]";
                }

                String str = "[";
                Node currentNode = headNode;
                while (currentNode != null) {
                        str = str + "" + currentNode.data + " , ";
                        currentNode = currentNode.next;
                }
                str = str.substring(0, str.length() - 2);
                str = str + "]";

                return str;
        }

        public static void main(String[] args) {

        Node head=null;
        head = insertFirst(head,10);
        head = insertFirst(head,20);
        head = insertFirst(head,30);

        System.out.println(toString(head));

        }

}
```

9.3.4. Insert the Node at end of D.L.L

Lab15.java

```
package com.jlcindia.linkedlist.doubly;

/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Lab15 {

        static Node insertLast(Node headNode,int data) {

                Node temp = new Node(data);

                //1.Empty List
                if(headNode==null) {
                        headNode=temp; //head node changed
                        return headNode;
                }

                //2.List with 1 or more Nodes
                Node currentNode = headNode;
                while (currentNode.next != null) {
                        currentNode = currentNode.next;
                }

                currentNode.next=temp;
                temp.prev=currentNode;
                return headNode;
        }
```

```
	static String toString(Node headNode) {

		if (headNode == null) {
			return "[]";
		}

		String str = "[";
		Node currentNode = headNode;
		while (currentNode != null) {
			str = str + "" + currentNode.data + " , ";
			currentNode = currentNode.next;
		}
		str = str.substring(0, str.length() - 2);
		str = str + "]";

		return str;
	}

	public static void main(String[] args) {

	Node head=null;
	head = insertLast(head,10);
	head = insertLast(head,20);
	head = insertLast(head,30);
	head = insertLast(head,99);

	System.out.println(toString(head));

	}

}
```