Student ID: 202112047

IT623 - Lab Assignment 2

1. Reverse singly list

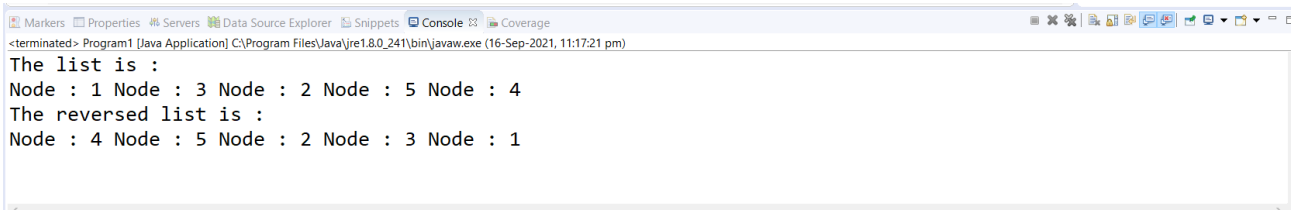
Code:

```
public class Program1 {

	Node head = null;
	Node tail = null;

	class Node {
		int data;
		Node next;

		Node(int x) {
			data = x;
			next = null;
		}
	}

	public void add(int data) {
		Node node = new Node(data);

		if (head == null) {
			head = node;
			tail = node;
		} else {
			tail.next = node;
			tail = node;
		}
```

```
}

void printList() {
        Node temp = head;
        if (head == null) {
                System.out.println("List is empty!!");
                return;
        }

        while (temp != null) {
                System.out.print("Node : " + temp.data + " ");
                temp = temp.next;
        }
}

void printListReverse() {
        Node prev = null;
        Node temp = head;

        if (head == null) {
                System.out.println("List is empty!!");
                return;
        }

        while (temp != null) {
                Node next = temp.next;
                temp.next = prev;
                prev = temp;
                temp = next;
        }
        head = prev;
}
```

```
	public static void main(String[] args) {
		Program1 p1 = new Program1();

		p1.add(1);
		p1.add(3);
		p1.add(2);
		p1.add(5);
		p1.add(4);

		System.out.println("The list is : ");
		p1.printList();

		System.out.println();

		System.out.println("The reversed list is : ");
		p1.printListReverse();
		p1.printList();
	}
}
```

Output Snapshot:



2. Remove duplicated from sorted linked list.

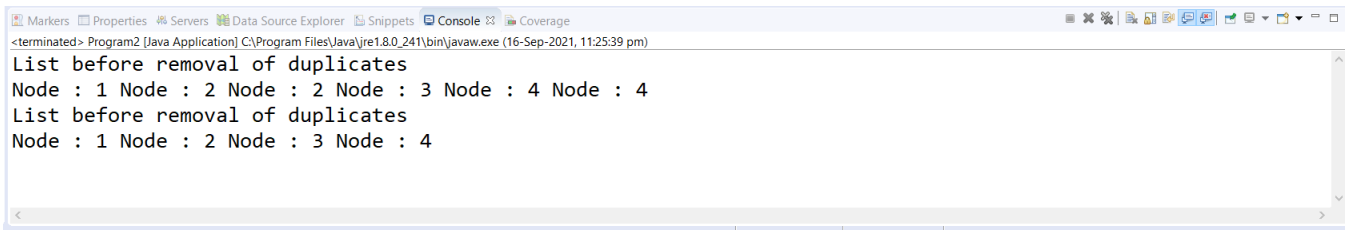
Code:

```
public class Program2 {
	Node head = null;
	Node tail = null;

	class Node {
		int data;
		Node next;

		Node(int x) {
			data = x;
			next = null;
		}
	}

	public void removeDuplicates() {
		Node current = head;

		while (current != null) {
			Node temp = current;
			while (temp != null && temp.data == current.data) {
				temp = temp.next;
			}

			current.next = temp;
			current = current.next;
		}
	}

	void add(int data) {
```

```
        Node node = new Node(data);

        if (head == null) {
                head = node;
                tail = node;
        } else {
                tail.next = node;
                tail = node;
        }
}

void printList() {
        Node temp = head;

        if (head == null) {
                System.out.println("List is empty!!");
        }

        while (temp != null) {
                System.out.print("Node : " + temp.data + " ");
                temp = temp.next;
        }
}

public static void main(String[] args) {
        Program2 p2 = new Program2();

        p2.add(1);
        p2.add(2);
        p2.add(2);
        p2.add(3);
        p2.add(4);
        p2.add(4);
```

```
            System.out.println("List before removal of duplicates");
            p2.printList();

            System.out.println("\nList before removal of duplicates");
            p2.removeDuplicates();
            p2.printList();

        }
}
```

Output Snapshot:



3. Count nodes in circular linked list.

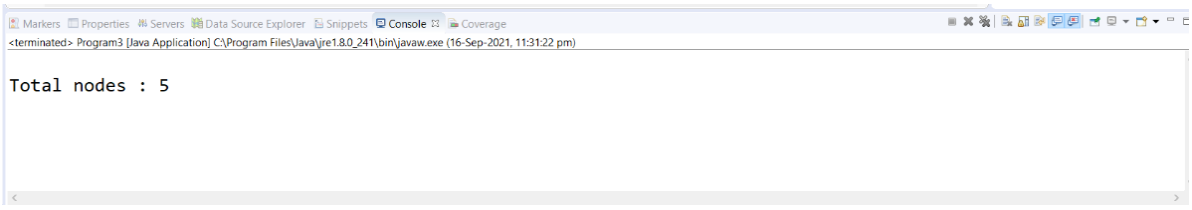
Code:

```
public class Program3 {

        Node head;
        int count;
        Node tail = null;

        class Node {
            int data;
            Node next;
```

```
        Node(int x) {
                data = x;
        }
}

public void add(int data) {
        Node node = new Node(data);

        if (head == null) {
                head = node;
                tail = node;
                node.next = head;
        } else {
                tail.next = node;
                tail = node;
                tail.next = head;
        }
}

public void countNodes() {
        Node current = head;

        if (head != null) {
                do {
                        current = current.next;
                        count++;
                } while (current != head);
        }

        System.out.println("\nTotal nodes : " + count);
```

```
    }

    public static void main(String[] args) {
        Program3 p3 = new Program3();

        p3.add(1);
        p3.add(2);
        p3.add(3);
        p3.add(4);
        p3.add(5);

        p3.countNodes();
    }
}
```

Output Snapshot:



4. Delete element from doubly linked list(position wise).

Code:

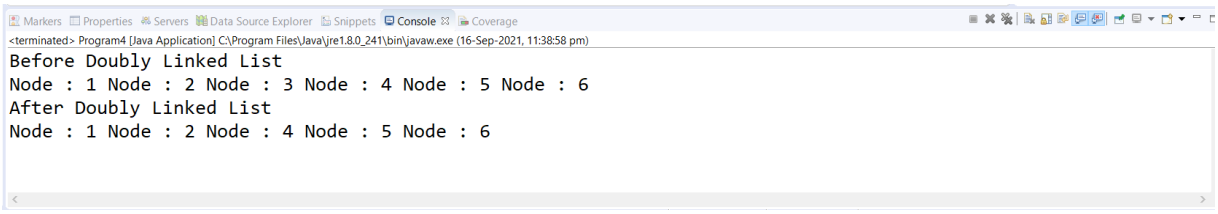
```
public class Program4 {

    Node head, tail;
```

```
Node next, prev;

class Node {
	int data;
	Node prev, next;

	Node(int x) {
		data = x;
	}
}

void add(int newDate) {
	Node newNode = new Node(newDate);

	if (head == null) {
		head = tail = newNode;
		head.prev = null;
		tail.next = null;
	} else {
		tail.next = newNode;
		newNode.prev = tail;
		tail = newNode;
		tail.next = null;
	}
}

void delete(int n) {
	if (head == null) {
		System.out.println("List is empty");
	} else {
		Node current = head;

		for (int i = 1; i < n; i++) {
			current = current.next;
```

```
                }

                if (current == head) {
                        current = current.next;
                } else if (current == tail) {
                        tail = tail.prev;
                } else {
                        current.prev.next = current.next;
                        current.next.prev = current.prev;
                }
                current = null;
        }
}

void print() {
        Node current = head;
        if (head == null) {
                System.out.println("List is empty!!");
                return;
        }

        while (current != null) {
                System.out.print("Node : " + current.data + " ");
                current = current.next;
        }
}

public static void main(String[] args) {
        Program4 p4 = new Program4();

        p4.add(1);
        p4.add(2);
        p4.add(3);
        p4.add(4);
```

```
            p4.add(5);
            p4.add(6);

            System.out.println("Before Doubly Linked List");
            p4.print();

            p4.delete(3);

            System.out.println("\nAfter Doubly Linked List");
            p4.print();

        }
}
```

Output Snapshot:



5. Find the middle element of the singly linked list.

Code:

```
public class Program5 {

        Node head = null;
        Node tail = null;

        class Node {
```

```
        int data;
        Node prev, next;

        Node(int x) {
                data = x;
        }
}

void add(int data) {
        Node node = new Node(data);

        if (head == null) {
                head = node;
                tail = node;
        } else {
                tail.next = node;
                tail = node;
        }
}

void print() {
        Node current = head;
        if (head == null)
                System.out.println("List is empty");

        while (current != null) {
                System.out.print("Node : " + current.data + " ");
                current = current.next;
        }
}

void printMiddle() {
        Node slow_ptr = head;
        Node fast_ptr = head;
```

```
        while (fast_ptr != null && fast_ptr.next != null) {
                fast_ptr = fast_ptr.next.next;
                slow_ptr = slow_ptr.next;
            }
            System.out.println("The middle element is [" + slow_ptr.data + "]
\n");

        }

        public static void main(String[] args) {
            Program5 p5 = new Program5();

            p5.add(1);
            p5.add(2);
            p5.add(3);
            p5.add(4);
            p5.add(5);
            p5.add(6);
            p5.add(7);

            System.out.println("The list is : ");
            p5.print();

            System.out.println();
            p5.printMiddle();
        }
}
```

Output Snapshot: