1.Doubly Linked List Insertion in java

2.Reverse a Doubly Linked List in java

3.Delete a node in a Doubly Linked List in java

4.Program to find length of Doubly Linked List in java

5.Find the largest node in Doubly linked list in java

Solution for quest 1 to 5

```
class DLL2{

	Node head;
	static class Node{
	int data;
	Node prev;
	Node next;

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

	void insert(int new_data)
	{
		Node new_node = new Node(new_data);
		new_node.next = head;
		new_node.prev = null;
		if( head != null)
			head.prev = new_node;
		head = new_node;
	}

	void insertAfter(Node prev, int new_data)
	{
```

```
        if(prev == null)
                return;
        Node new_node = new Node(new_data);
        new_node.next = prev.next;
        prev.next = new_node;
        new_node.prev = prev;
        Node p = new_node.next;
        p.prev = new_node;



}




void display(Node n)
{
        Node p=null;
        System.out.println("Forward printing:");
        while(n != null)
        {
                System.out.print(n.data+ " ");
                p=n;
                n=n.next;

        }
        System.out.println("--------");
        System.out.println("Backward printing(Reverse):");
        while(p != null)
        {
                System.out.print(p.data+ " ");
                p=p.prev;

        }


}

void deletenode(Node n)
{
        //base condition
```

```
        if(head == null || n == null)
        {
                return;
        }
        //deletion at the begining
        if(head == n)
                head=n.next;
                //head.prev=null;
        // deletion in between two elements
        if(n.next != null)
                n.next.prev = n.prev;
        if(n.prev != null)
                n.prev.next=n.next;

        return;


}

int countNodes(Node head){
        int count=0;
        Node temp=head;
        while (temp!=null){
                count++;
                temp=temp.next;
        }
        return count;
}

int largestNode(Node head){
        int max=head.data;
        Node temp=head.next;
        while(temp!=null){
                if(temp.data>max){
                        max=temp.data;
                }
                temp=temp.next;
        }
        return max;
}
public static void main(String args[])
{
```

```
        DLL2 d1 = new DLL2();
        d1.insert(5);
        d1.insert(10);
        d1.insert(15);
        d1.display(d1.head);
        System.out.println();
        d1.insertAfter(d1.head, 7);
        d1.display(d1.head);

        System.out.println();
        System.out.println("Deleting nodes");
        d1.deletenode(d1.head);
        d1.display(d1.head);

        System.out.println();
        d1.deletenode(d1.head.next);
        d1.display(d1.head);
        System.out.println();
        System.out.println("Length of Doubly Linked list is "+d1.countNodes(d1.head));

        System.out.println();
        System.out.println("Largest node in list is "+d1.largestNode(d1.head));
    }
}
```

```
D:\ARUSA CDAC\ADS\Day3\DLL>java DLL2
Forward printing:
15 10 5 --------
Backward printing(Reverse):
5 10 15
Forward printing:
15 7 10 5 --------
Backward printing(Reverse):
5 10 7 15
Deleting nodes
Forward printing:
7 10 5 --------
Backward printing(Reverse):
5 10 7
Forward printing:
7 5 --------
Backward printing(Reverse):
5 7
Length of Doubly Linked list is 2

Largest node in list is 7

D:\ARUSA CDAC\ADS\Day3\DLL>
```

6.Insert value in sorted way in a sorted doubly linked list in java

```
class Arusa{

   Node head;

   class Node {

   int data;

   Node prev;

   Node next;


   Node(int d) {

      data = d;

      prev = null;
```

```
    next = null;
  }
}
```

```
// Function to insert a node with given data in sorted way
void sortedInsert(int new_data) {
    Node new_node = new Node(new_data);
    Node current;

    // If list is empty or new node is to be inserted before the head node
    if (head == null || head.data >= new_node.data) {
        new_node.next = head;
        new_node.prev = null;
        if (head != null)
            head.prev = new_node;
        head = new_node;
        return;
    }

    // Find the node after which new node to be inserted
    current = head;
    while (current.next != null && current.next.data < new_node.data)
```

```
        current = current.next;

    // Insert the new_node after current
    new_node.next = current.next;
    if (current.next != null)
        current.next.prev = new_node;
    current.next = new_node;
    new_node.prev = current;
}

// Function to print nodes in a given doubly linked list
void printList(Node node) {
    while (node != null) {
        System.out.print(node.data + " ");
        node = node.next;
    }
}

public static void main(String[] args) {
    Arusa list = new Arusa();

    // Insert 10, 20, 30, 40, 50 in sorted order
    list.sortedInsert(40);
    list.sortedInsert(10);
    list.sortedInsert(30);
```

```
        list.sortedInsert(50);
        list.sortedInsert(20);

        System.out.println("Sorted Doubly Linked List:");
        list.printList(list.head);
    }
}
```

==========

7.Write tree traversals in java

```
class Arusa_BinaryTree {
    Node root;
// Node class representing a node in the binary tree
    static class Node {
    int data;
    Node left, right;
    public Node(int item) {
        data = item;
        left = right = null;
     }
   }
   public Arusa_BinaryTree() {
        root = null;
    }
   // Inorder traversal: Left -> Root -> Right
    public void inorderTraversal(Node node) {
```

```
    if (node == null)
        return;

    inorderTraversal(node.left);
    System.out.print(node.data + " ");
    inorderTraversal(node.right);
  }
 // Preorder traversal: Root -> Left -> Right
  public void preorderTraversal(Node node) {
    if (node == null)
        return;

    System.out.print(node.data + " ");
    preorderTraversal(node.left);
    preorderTraversal(node.right);
  }
 // Postorder traversal: Left -> Right -> Root
  public void postorderTraversal(Node node) {
    if (node == null)
        return;

    postorderTraversal(node.left);
    postorderTraversal(node.right);
    System.out.print(node.data + " ");
  }
```

```
    // Driver method to test traversal methods
    public static void main(String[] args) {
        Arusa_BinaryTree tree = new Arusa_BinaryTree();
        tree.root = new Node(10);
        tree.root.left = new Node(20);
        tree.root.right = new Node(30);
        tree.root.left.left = new Node(40);
        tree.root.left.right = new Node(50);

        System.out.println("Inorder traversal:");
        tree.inorderTraversal(tree.root);
        System.out.println("\nPreorder traversal:");
        tree.preorderTraversal(tree.root);
        System.out.println("\nPostorder traversal:");
        tree.postorderTraversal(tree.root);
    }
}
```

==========

8.Search a node in Binary Tree

```
class Arusa_BinaryTree {
    Node root;
// Node class representing a node in the binary tree
    static class Node {
    int data;
    Node left, right;
```

```
 public Node(int item) {
    data = item;
    left = right = null;
  }
}
public Arusa_BinaryTree() {
    root = null;
 }
 // Search for a node with given key in the binary tree
 public boolean search(Node node, int key) {
    // Base Cases: root is null or key is present at root
    if (node == null)
       return false;
    if (node.data == key)
       return true;

    // Recur for left and right subtrees
    return search(node.left, key) || search(node.right, key);
 }
 public static void main(String[] args) {
    Arusa_BinaryTree tree = new Arusa_BinaryTree();
    tree.root = new Node(10);
    tree.root.left = new Node(20);
    tree.root.right = new Node(30);
    tree.root.left.left = new Node(40);
```

```
        tree.root.left.right = new Node(50);


        int key = 40;
        if (tree.search(tree.root, key))
            System.out.println( key + " found in the tree");
        else
            System.out.println( key + " not found in the tree");


    }
}
```

==========

9.Inorder Successor of a node in Binary Tree

```
class Arusa_BinaryTree {
    Node root;
// Node class representing a node in the binary tree
    static class Node {
    int data;
    Node left, right;
    public Node(int item) {
        data = item;
        left = right = null;
     }
   }
   public Arusa_BinaryTree() {
```

```
    root = null;
}
// Function to find the leftmost node in the subtree rooted at given node
public Node findLeftmostNode(Node node) {
    if (node == null)
        return null;

    while (node.left != null)
        node = node.left;
    return node;
}
// Function to find the inorder successor of a given node
public Node inorderSuccessor(Node root, Node node) {
    // If right subtree of node is not null, then the inorder successor
    // is the leftmost node in the right subtree
    if (node.right != null)
        return findLeftmostNode(node.right);

    // Otherwise, we need to find the ancestor of the node for which
    // the given node is in the left subtree
    Node successor = null;
    Node current = root;
    while (current != null) {
        if (node.data < current.data) {
            successor = current;
```

```
            current = current.left;
        } else if (node.data > current.data) {
            current = current.right;
        } else {
            break; // Node found, exit loop
        }
    }
    return successor;
}
public static void main(String[] args) {
    Arusa_BinaryTree tree = new Arusa_BinaryTree();
    tree.root = new Node(10);
    tree.root.left = new Node(20);
    tree.root.right = new Node(30);
    tree.root.left.left = new Node(40);
    tree.root.left.right = new Node(50);

    Node node = tree.root.left.right; // Node for which we want to find the successor
    Node successor = tree.inorderSuccessor(tree.root, node);
    if (successor != null)
        System.out.println("Inorder successor of " + node.data + " is " + successor.data);
    else
        System.out.println("No inorder successor found for " + node.data);

}
```

```
}
```

==========

10.Print Head node of every node in Binary Tree

```
class Arusa_BinaryTree {
   Node root;
// Node class representing a node in the binary tree
   static class Node {
   int data;
   Node left, right;
   public Node(int item) {
      data = item;
      left = right = null;
    }
  }
  public Arusa_BinaryTree() {
      root = null;
   }
   // Function to find the head node
   public Node findHeadNode(Node root, Node node) {
      if (root == null || root == node) {
         return root;
      }
      Node left = findHeadNode(root.left, node);
      Node right = findHeadNode(root.right, node);
// If the node is found in the left subtree, return the root of the left subtree
```

```
        if (left != null) {
            return left;
        }
// If the node is found in the right subtree, return the root of the right subtree
        if (right != null) {
            return right;
        }
        // Otherwise, the node is not found in the current subtree
        return null;
    }
  // Function to print the head node
    public void printHeadNodes(Node root) {
        if (root == null) {
            return;
        }
// Traverse each node and print its head node
        printHeadNodes(root.left);
        System.out.println("Head node of " + root.data + " is " + findHeadNode(this.root, root).data);
        printHeadNodes(root.right);
    }
    public static void main(String[] args) {
        Arusa_BinaryTree tree = new Arusa_BinaryTree();
        tree.root = new Node(10);
        tree.root.left = new Node(20);
        tree.root.right = new Node(30);
```

```
        tree.root.left.left = new Node(40);
        tree.root.left.right = new Node(50);


        tree.printHeadNodes(tree.root);


    }
}
```