1. Doubly Linked List Insertion in java:-

```
public class DoublyLinkedList {
    Node head;

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

        Node() {
        }

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

    public void insertData(int d) {
        Node new_node = new Node(d);
        new_node.next = head;
        new_node.prev = null;
        if (head != null)
            head.prev = new_node;
        head = new_node;
    }
```

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

    public static void main(String args[]) {
        DoublyLinkedList list = new DoublyLinkedList();
        list.insertData(21);
        list.insertData(30);
        list.insertData(12);
        list.insertData(201);
        list.insertData(2001);
        list.insertData(102);
        list.printData();
    }
}
```

2. Reverse a Doubly Linked List in java

```
public class ReverseDoublyLinkedList {
    Node head;

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

        Node() {
        }

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

    public void insertData(int d) {
        Node new_node = new Node(d);
        new_node.next = head;
        new_node.prev = null;
        if (head != null)
            head.prev = new_node;
        head = new_node;
    }
```

```
public void reverseList() {
    Node temp = null;
    Node current = head;
    while (current != null) {
        temp = current.prev;
        current.prev = current.next;
        current.next = temp;
        current = current.prev;
    }
    if (temp != null)
        head = temp.prev;
}

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

public static void main(String args[]) {
    ReverseDoublyLinkedList list = new ReverseDoublyLinkedList();
    list.insertData(21);
    list.insertData(30);
    list.insertData(12);
```

```
        list.insertData(201);
        list.insertData(2001);
        list.insertData(102);
        System.out.println("Original List:");
        list.printData();
        list.reverseList();
        System.out.println("\nReversed List:");
        list.printData();
    }
}
```

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

```
public class DeleteDoublyLinkedList {
    Node head;

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

        Node() {
        }

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

    public void insertData(int d) {
        Node new_node = new Node(d);
        new_node.next = head;
        new_node.prev = null;
        if (head != null)
            head.prev = new_node;
        head = new_node;
    }
```

```
public void deleteNode(Node del) {
    if (head == null || del == null)
        return;
    if (head == del)
        head = del.next;
    if (del.next != null)
        del.next.prev = del.prev;
    if (del.prev != null)
        del.prev.next = del.next;
}

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

public static void main(String args[]) {
    DeleteDoublyLinkedList list = new DeleteDoublyLinkedList();
    list.insertData(21);
    list.insertData(30);
    list.insertData(12);
    list.insertData(201);
    list.insertData(2001);
```

```
        list.insertData(102);
        System.out.println("Original List:");
        list.printData();
        Node delNode = list.head.next.next; // Delete node with data 30
        list.deleteNode(delNode);
        System.out.println("\nList after deleting node with data 30:");
        list.printData();
    }
}
```

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

```
public class DoublyLinkedListLength {
    Node head;

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

        Node() {
        }

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

    public void insertData(int d) {
        Node new_node = new Node(d);
        new_node.next = head;
        new_node.prev = null;
        if (head != null)
            head.prev = new_node;
        head = new_node;
    }
```

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

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

public static void main(String args[]) {
    DoublyLinkedListLength list = new DoublyLinkedListLength();
    list.insertData(21);
    list.insertData(30);
    list.insertData(12);
    list.insertData(201);
    list.insertData(2001);
    list.insertData(102);
```

```
        System.out.println("Original List:");
        list.printData();
        System.out.println("\nLength of Doubly Linked List: " + list.length());
    }
}
```

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

```
public class LargestNodeDoublyLinkedList {
    Node head;

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

        Node() {
        }

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

    public void insertData(int d) {
        Node new_node = new Node(d);
        new_node.next = head;
        new_node.prev = null;
        if (head != null)
            head.prev = new_node;
        head = new_node;
    }
```

```
public int findLargest() {
    if (head == null)
        return Integer.MIN_VALUE;

    int largest = head.data;
    Node temp = head.next;
    while (temp != null) {
        if (temp.data > largest) {
            largest = temp.data;
        }
        temp = temp.next;
    }
    return largest;
}

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

public static void main(String args[]) {
    LargestNodeDoublyLinkedList list = new LargestNodeDoublyLinkedList();
    list.insertData(21);
```

```
        list.insertData(30);
        list.insertData(12);
        list.insertData(201);
        list.insertData(2001);
        list.insertData(102);
        System.out.println("Original List:");
        list.printData();
        System.out.println("\nLargest Node in Doubly Linked List: " + list.findLargest());
    }
}
```

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

```
public class SortedInsertDoublyLinkedList {
    Node head;

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

        Node() {
        }

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

    public void sortedInsert(int newData) {
        Node newNode = new Node(newData);
        if (head == null) {
            head = newNode;
        } else if (newNode.data <= head.data) {
            newNode.next = head;
            head.prev = newNode;
            head = newNode;
```

```
    } else {
        Node current = head;
        while (current.next != null && current.next.data < newNode.data) {
            current = current.next;
        }
        newNode.next = current.next;
        if (current.next != null) {
            current.next.prev = newNode;
        }
        current.next = newNode;
        newNode.prev = current;
    }
}

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

public static void main(String args[]) {
    SortedInsertDoublyLinkedList list = new SortedInsertDoublyLinkedList();
    list.sortedInsert(21);
    list.sortedInsert(30);
    list.sortedInsert(12);
```

```
        list.sortedInsert(201);
        list.sortedInsert(2001);
        list.sortedInsert(102);
        System.out.println("Original List:");
        list.printData();
    }
}
```

7. Write tree traversals in java

```
class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

public class BinaryTree {
    Node root;

    public BinaryTree() {
        root = null;
    }

    /* Preorder traversal of a binary tree */
    public void preorderTraversal(Node node) {
        if (node == null)
            return;

        // Print the data of the node
        System.out.print(node.data + " ");

        // Traverse the left subtree
```

```
    preorderTraversal(node.left);

    // Traverse the right subtree
    preorderTraversal(node.right);
}

/* Inorder traversal of a binary tree */
public void inorderTraversal(Node node) {
    if (node == null)
        return;

    // Traverse the left subtree
    inorderTraversal(node.left);

    // Print the data of the node
    System.out.print(node.data + " ");

    // Traverse the right subtree
    inorderTraversal(node.right);
}

/* Postorder traversal of a binary tree */
public void postorderTraversal(Node node) {
    if (node == null)
        return;

    // Traverse the left subtree
```

```
        postorderTraversal(node.left);

        // Traverse the right subtree
        postorderTraversal(node.right);

        // Print the data of the node
        System.out.print(node.data + " ");
    }

    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

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

8. Search a node in Binary Tree

```
class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

public class BinaryTree {
    Node root;

    public BinaryTree() {
        root = null;
    }

    /* Search for a node with a given key */
    public Node search(Node root, int key) {
        // Base Cases: root is null or key is present at root
        if (root == null || root.data == key)
            return root;

        // Key is greater than root's key
        if (root.data < key)
            return search(root.right, key);
```

```
        // Key is smaller than root's key
        return search(root.left, key);
    }

    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(4);
        tree.root.left = new Node(2);
        tree.root.right = new Node(5);
        tree.root.left.left = new Node(1);
        tree.root.left.right = new Node(3);

        int key = 3;
        Node result = tree.search(tree.root, key);
        if (result != null)
            System.out.println("Node with key " + key + " found in the binary tree.");
        else
            System.out.println("Node with key " + key + " not found in the binary tree.");
    }
}
```

9. Inorder Successor of a node in Binary Tree

```
class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

public class BinaryTree {
    Node root;

    public BinaryTree() {
        root = null;
    }

    /* Function to find the node with minimum value in a subtree rooted at given node */
    public Node minValueNode(Node node) {
        Node current = node;
        // Loop down to find the leftmost leaf
        while (current.left != null)
            current = current.left;
        return current;
    }
```

```
    /* 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 successor is the leftmost node in the right subtree
        if (node.right != null)
            return minValueNode(node.right);

        Node successor = null;
        // Start from root and search for the successor down the tree
        while (root != null) {
            if (node.data < root.data) {
                successor = root;
                root = root.left;
            } else if (node.data > root.data) {
                root = root.right;
            } else
                break; // If node is found, break the loop
        }
        return successor;
    }

    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(20);
        tree.root.left = new Node(8);
        tree.root.right = new Node(22);
        tree.root.left.left = new Node(4);
```

```
        tree.root.left.right = new Node(12);
        tree.root.left.right.left = new Node(10);
        tree.root.left.right.right = new Node(14);

        Node node = tree.root.left.right; // Node with data 12
        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("Inorder successor of " + node.data + " is not found");
    }
}
```

10. Print Head node of every node in Binary Tree

```
class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

public class BinaryTree {
    Node root;

    public BinaryTree() {
        root = null;
    }

    /* Function to print the head node of every node in the binary tree */
    public void printHeadNodes(Node node) {
        if (node == null)
            return;

        System.out.println("Head node of " + node.data + " is " + findHeadNode(node));

        // Recursively print head nodes of left and right subtrees
        printHeadNodes(node.left);
```

```
        printHeadNodes(node.right);
    }

    /* Function to find the head node of a given node */
    public int findHeadNode(Node node) {
        while (node.left != null)
            node = node.left;
        return node.data;
    }

    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);

        tree.printHeadNodes(tree.root);
    }
}
```