1.Doubly Linked List Insertion in java

```
class Dlist{
   Node head;
   Node tail;
   static class Node {
    int data;
    Node prev;
    Node next;
    Node(int d) {
     data = d;
     prev = null;
     next = null;
    }
   }
   public void insertBeginning(int data) {
      Node newNode = new Node(data);
      if (head == null) {
         head = newNode;
         tail = newNode;
      } else {
         newNode.next = head;
         head.prev = newNode;
         head = newNode;
      }
   }
```

```
public void insertEnd(int data) {
    Node newNode = new Node(data);
    if (head == null) {
        head = newNode;
        tail = newNode;
    } else {
        tail.next = newNode;
        newNode.prev = tail;
        tail = newNode;
    }
}
// Method to insert a node at a given position
public void insertAtPosition(int data, int position) {
    if (position < 0)
        throw new IllegalArgumentException("Position cannot be negative");

    Node newNode = new Node(data);
    if (position == 0) {
        newNode.next = head;
        if (head != null)
            head.prev = newNode;
        head = newNode;
        return;
    }
    Node current = head;
```

```
    int currentPosition = 0;
    while (currentPosition < position - 1 && current != null) {
        current = current.next;
        currentPosition++;
    }

    if (current == null && currentPosition < position - 1) {
        throw new IllegalArgumentException("Position exceeds the length of the list");
    }

    newNode.next = current.next;
    if (current.next != null)
        current.next.prev = newNode;
    current.next = newNode;
    newNode.prev = current;
}
public void display() {
    Node current = head;
    while (current != null) {
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}
public static void main(String[] args) {
```

```
        Dlist dll = new Dlist ();
        System.out.println("Insert at beggining :");
        dll.insertBeginning(1);
        dll.insertBeginning(2);
        dll.insertBeginning(3);
        dll.display();
        System.out.println("Insert at end :");
        dll.insertEnd(4);
        dll.insertEnd(5);
        dll.insertEnd(6);
        dll.display();
        dll.insertAtPosition(4,3);
        System.out.println("after insert 4 at position 3 :");
        dll.display();
    }
}
```

==========

2.Reverse a Doubly Linked List in java

```
class Rev {
    Node head;
   //creating node
    static class Node {
    int data;
    Node prev; //prev node reference
    Node next;
```

```
 Node(int d) {
    data = d;
    prev = null;
    next = null;
  }
 }
// Function to reverse a doubly linked list
 void reverse() {
    Node temp = null;
    Node current = head;
    // Swap next and prev for all nodes of
    // doubly linked list
    while (current != null) {
       temp = current.prev;
       current.prev = current.next;
       current.next = temp;
       current = current.prev;
    }
   // Before changing head, check empty list
    if (temp != null) {
       head = temp.prev;
    }
 }
// 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;
        }
    }
  // Function to insert a node at the beginning of the doubly linked list
    void push(int new_data) {
        // Allocate node
        Node new_node = new Node(new_data);
  // Make next of new node as head and previous as NULL
        new_node.next = head;
        new_node.prev = null;
  // Change prev of head node to new node
        if (head != null)
            head.prev = new_node;
// Move the head to point to the new node
        head = new_node;
    }
    public static void main(String[] args) {
    Rev list = new Rev ();
        list.push(1);
        list.push(4);
        list.push(5);
        list.push(3);
        list.push(8);
```

```
        System.out.println("Original linked list:");
        list.printList(list.head);

        list.reverse();

        System.out.println("\nReversed linked list:");
        list.printList(list.head);
    }
}
```

==========

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

```
class D {
    Node head;
   //creating node
    static class Node {
    int data;
    Node prev; //prev node reference
    Node next;
    Node(int d) {
        data = d;
        prev = null;
        next = null;
      }
    }
```

```
 void deleteNode(Node del){
    if (head == null || del == null) {
        return;
    }
   // If node to be deleted is head node
    if (head == del) {
        head = del.next;
    }
   // Change next only if node to be deleted is NOT the last node
    if (del.next != null) {
        del.next.prev = del.prev;
    }
    // Change prev only if node to be deleted is NOT the first node
    if (del.prev != null) {
        del.prev.next = del.next;
    }
 }
 // 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;
    }
 }
// Function to insert a node at the beginning of the doubly linked list
```

```
  void insert(int new_data) {
      // Allocate node
      Node new_node = new Node(new_data);
 // Make next of new node as head and previous as NULL
      new_node.next = head;
      new_node.prev = null;
 // Change prev of head node to new node
      if (head != null)
         head.prev = new_node;
// Move the head to point to the new node
      head = new_node;
  }
  public static void main(String[] args) {
      D list = new D ();
      list.insert(1);
      list.insert(4);
      list.insert(5);
      list.insert(3);
      list.insert(8);

      System.out.println("Original linked list:");
      list.printList(list.head);
      // Delete node with value 5
      Node delNode = list.head.next.next; // Node with value 5
      list.deleteNode(delNode);
```

```
        System.out.println("\nLinked list after deleting node with value 5:");
        list.printList(list.head);
    }
}
```

==========

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

```
class ins {
    Node head;
   //creating node
    static class Node {
    int data;
    Node prev; //prev node reference
    Node next;
    Node(int d) {
        data = d;
        prev = null;
        next = null;
      }
    }
    int length() {
        int count = 0;
        Node current = head;
        while (current != null) {
            count++;
            current = current.next;
```

```
        }
        return count;
    }
    // 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;
        }
    }
  // Function to insert a node at the beginning of the doubly linked list
    void insert(int new_data) {
        // Allocate node
        Node new_node = new Node(new_data);
  // Make next of new node as head and previous as NULL
        new_node.next = head;
        new_node.prev = null;
  // Change prev of head node to new node
        if (head != null)
            head.prev = new_node;
// Move the head to point to the new node
        head = new_node;
    }
    public static void main(String[] args) {
        Ins list = new ins ();
```

```
        list.insert(1);
        list.insert(4);
        list.insert(5);
        list.insert(3);
        list.insert(8);

      System.out.println("Length of the linked list: " + list.length());
    }
}
```

==========

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

```
class Cr {
    Node head;
   //creating node
    static class Node {
    int data;
    Node prev; //prev node reference
    Node next;
    Node(int d) {
        data = d;
        prev = null;
        next = null;
      }
    }
```

```
 int findLargest() {
    if (head == null) {
       return Integer.MIN_VALUE; // Return minimum value if the list is empty
    }
    int max = head.data;
     Node current = head.next;
    // Iterate through the list and update the max value
    while (current != null) {
       if (current.data > max) {
          max = current.data;
       }
       current = current.next;
    }
     return max;
 }
 // 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;
    }
 }
// Function to insert a node at the beginning of the doubly linked list
 void insert(int new_data) {
    // Allocate node
```

```
        Node new_node = new Node(new_data);
    // Make next of new node as head and previous as NULL
        new_node.next = head;
        new_node.prev = null;
    // Change prev of head node to new node
        if (head != null)
            head.prev = new_node;
   // Move the head to point to the new node
        head = new_node;
      }
      public static void main(String[] args) {
        Cr  list = new Cr ();
        list.insert(1);
        list.insert(4);
        list.insert(5);
        list.insert(3);
        list.insert(8);

       int largest = list.findLargest();

        if (largest != Integer.MIN_VALUE) {
            System.out.println("Largest element in the linked list: " + largest);
        } else {
            System.out.println("The linked list is empty.");
        }
```

```
    }
}
```

==========

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

```
class srt {
    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) {
        Srt  list = new Srt ();

        // 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 bt _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 bt _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) {
    bt _BinaryTree tree = new bt _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 bt _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 bt _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) {
        bt _BinaryTree tree = new bt _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 bt _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 bt _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) {
    bt _BinaryTree tree = new bt _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 bt _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 bt _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) {
        bt _BinaryTree tree = new bt _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);

    }
}
```

==========