Answer 1)

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

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

public class MergeLinkedLists {
    public static Node createNewList(Node list1, Node list2) {
        // Check if either of the lists is empty
        if (list1 == null) {
            return list2;
        } else if (list2 == null) {
            return list1;
        }

        // Initialize variables
        Node newList = null;
        Node current = null;
        Node current1 = list1;
        Node current2 = list2;

        // Traverse both lists
        while (current1 != null && current2 != null) {
            // Compare values of current nodes
            if (current1.data >= current2.data) {
                // Create a new node with greater value
                Node newNode = new Node(current1.data);
                current1 = current1.next;
                if (newList == null) {
                    newList = newNode;
                    current = newNode;
                } else {
                    current.next = newNode;
                    current = newNode;
                }
            } else {
                Node newNode = new Node(current2.data);
                current2 = current2.next;
                if (newList == null) {
```

```
                newList = newNode;
                current = newNode;
            } else {
                current.next = newNode;
                current = newNode;
            }
        }
    }

    // Append the remaining nodes from either list
    if (current1 != null) {
        current.next = current1;
    }
    if (current2 != null) {
        current.next = current2;
    }

    return newList;
}

public static void printLinkedList(Node head) {
    Node current = head;
    while (current != null) {
        System.out.print(current.data + "->");
        current = current.next;
    }
    System.out.println("null");
}

public static void main(String[] args) {
    // Create the first linked list: 5->2->3->8
    Node list1 = new Node(5);
    list1.next = new Node(2);
    list1.next.next = new Node(3);
    list1.next.next.next = new Node(8);

    // Create the second linked list: 1->7->4->5
    Node list2 = new Node(1);
    list2.next = new Node(7);
    list2.next.next = new Node(4);
    list2.next.next.next = new Node(5);

    // Create the new linked list
    Node newList = createNewList(list1, list2);
```

```
        // Traverse and print the new linked list: 5->7->4->8
        printLinkedList(newList);
    }
}


Answer 2)
class Node {
    int data;
    Node next;

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

public class RemoveDuplicates {
    public static Node removeDuplicates(Node head) {
        if (head == null || head.next == null) {
            return head;
        }

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

        return head;
    }

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

```
    public static void main(String[] args) {
        // Create the linked list: 11->11->11->21->43->43->60
        Node head = new Node(11);
        head.next = new Node(11);
        head.next.next = new Node(11);
        head.next.next.next = new Node(21);
        head.next.next.next.next = new Node(43);
        head.next.next.next.next.next = new Node(43);
        head.next.next.next.next.next.next = new Node(60);

        // Remove duplicate nodes
        Node newList = removeDuplicates(head);

        // Print the updated linked list: 11->21->43->60
        printLinkedList(newList);
    }
}
```

Answer 3)

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

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

public class ReverseKNodes {
    public static Node reverseKNodes(Node head, int k) {
        if (head == null || k <= 1) {
            return head;
        }

        Node current = head;
        Node prev = null;
        Node next = null;
        int count = 0;

        // Reverse first k nodes
```

```
        while (current != null && count < k) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
            count++;
        }

        // Recursive call for the remaining list
        if (next != null) {
            head.next = reverseKNodes(next, k);
        }

        return prev;
    }

    public static void printLinkedList(Node head) {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        // Create the linked list: 1->2->2->4->5->6->7->8
        Node head = new Node(1);
        head.next = new Node(2);
        head.next.next = new Node(2);
        head.next.next.next = new Node(4);
        head.next.next.next.next = new Node(5);
        head.next.next.next.next.next = new Node(6);
        head.next.next.next.next.next.next = new Node(7);
        head.next.next.next.next.next.next.next = new Node(8);

        int k = 4;

        // Reverse every k nodes
        Node newList = reverseKNodes(head, k);

        // Print the updated linked list: 4 2 2 1 8 7 6 5
        printLinkedList(newList);
    }
```

```
}
```

Answer 4)

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

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

public class ReverseAlternateKNodes {
    public static Node reverseAlternateKNodes(Node head, int k) {
        if (head == null || k <= 1) {
            return head;
        }

        Node current = head;
        Node prev = null;
        Node next = null;
        int count = 0;

        // Reverse first k nodes
        while (current != null && count < k) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
            count++;
        }

        // Skip next k nodes
        count = 0;
        while (current != null && count < k - 1) {
            current = current.next;
            count++;
        }

        // Recursive call for the remaining list
```

```
        if (current != null) {
            current.next = reverseAlternateKNodes(current.next, k);
        }

        return prev;
    }

    public static void printLinkedList(Node head) {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        // Create the linked list: 1->2->3->4->5->6->7->8->9->null
        Node head = new Node(1);
        head.next = new Node(2);
        head.next.next = new Node(3);
        head.next.next.next = new Node(4);
        head.next.next.next.next = new Node(5);
        head.next.next.next.next.next = new Node(6);
        head.next.next.next.next.next.next = new Node(7);
        head.next.next.next.next.next.next.next = new Node(8);
        head.next.next.next.next.next.next.next.next = new Node(9);

        int k = 3;

        // Reverse every alternate k nodes
        Node newList = reverseAlternateKNodes(head, k);

        // Print the updated linked list: 3 2 1 4 5 6 9 8 7
        printLinkedList(newList);
    }
}
```

Answer 5)

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

```
    Node next;

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

public class DeleteLastOccurrence {
    public static Node deleteLastOccurrence(Node head, int key) {
        if (head == null) {
            return null;
        }

        Node prev = null;
        Node current = head;
        Node lastOccurrence = null;
        Node lastOccurrencePrev = null;

        while (current != null) {
            if (current.data == key) {
                lastOccurrence = current;
                lastOccurrencePrev = prev;
            }
            prev = current;
            current = current.next;
        }

        if (lastOccurrence == null) {
            return head; // Key not found, return the original list
        }

        if (lastOccurrencePrev == null) {
            head = head.next; // Last occurrence is at the head
        } else {
            lastOccurrencePrev.next = lastOccurrence.next; // Skip the last occurrence
        }

        return head;
    }

    public static void printLinkedList(Node head) {
        Node current = head;
        while (current != null) {
```

```
            System.out.print(current.data + "->");
            current = current.next;
        }
        System.out.println("null");
    }

    public static void main(String[] args) {
        // Create the linked list: 1->2->3->5->2->10
        Node head = new Node(1);
        head.next = new Node(2);
        head.next.next = new Node(3);
        head.next.next.next = new Node(5);
        head.next.next.next.next = new Node(2);
        head.next.next.next.next.next = new Node(10);

        int key = 2;

        // Delete the last occurrence of the key
        Node newList = deleteLastOccurrence(head, key);

        // Print the updated linked list: 1->2->3->5->10
        printLinkedList(newList);
    }
}
```

Answer 6)

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

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

public class MergeSortedLists {
    public static Node mergeSortedLists(Node a, Node b) {
        // Check if any of the lists is empty
        if (a == null) {
            return b;
        }
```

```
    if (b == null) {
        return a;
    }

    Node head;
    Node tail;

    // Determine the head of the merged list
    if (a.data <= b.data) {
        head = a;
        tail = a;
        a = a.next;
    } else {
        head = b;
        tail = b;
        b = b.next;
    }

    // Merge the lists
    while (a != null && b != null) {
        if (a.data <= b.data) {
            tail.next = a;
            tail = a;
            a = a.next;
        } else {
            tail.next = b;
            tail = b;
            b = b.next;
        }
    }

    // Append remaining nodes, if any
    if (a != null) {
        tail.next = a;
    }
    if (b != null) {
        tail.next = b;
    }

    return head;
}

public static void printLinkedList(Node head) {
    Node current = head;
```

```
        while (current != null) {
            System.out.print(current.data + "->");
            current = current.next;
        }
        System.out.println("null");
    }

    public static void main(String[] args) {
        // Create the linked lists: a: 5->10->15, b: 2->3->20
        Node a = new Node(5);
        a.next = new Node(10);
        a.next.next = new Node(15);

        Node b = new Node(2);
        b.next = new Node(3);
        b.next.next = new Node(20);

        // Merge the sorted lists
        Node mergedList = mergeSortedLists(a, b);

        // Print the merged list: 2->3->5->10->15->20
        printLinkedList(mergedList);
    }
}
```

Answer 7)

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

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

public class ReverseDoublyLinkedList {
    public static Node reverseDoublyLinkedList(Node head) {
        if (head == null || head.next == null) {
```

```
        return head;
    }

    Node current = head;
    Node temp = null;

    // Swap prev and next pointers for all nodes
    while (current != null) {
        temp = current.prev;
        current.prev = current.next;
        current.next = temp;
        current = current.prev; // Move to the next node
    }

    // The new head will be the last node (original tail)
    return temp.prev;
}

public static void printLinkedList(Node head) {
    Node current = head;
    while (current != null) {
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}

public static void main(String[] args) {
    // Create the doubly linked list: 10 <-> 8 <-> 4 <-> 2
    Node head = new Node(10);
    Node node1 = new Node(8);
    Node node2 = new Node(4);
    Node node3 = new Node(2);

    head.next = node1;
    node1.prev = head;
    node1.next = node2;
    node2.prev = node1;
    node2.next = node3;
    node3.prev = node2;

    System.out.print("Original Linked list: ");
    printLinkedList(head);
```

```
        // Reverse the doubly linked list
        Node reversedList = reverseDoublyLinkedList(head);

        System.out.print("Reversed Linked list: ");
        printLinkedList(reversedList);
    }
}
```

Answer 8)

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

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

public class DeleteNodeFromPosition {
    public static Node deleteNode(Node head, int position) {
        if (head == null) {
            return null;
        }

        if (position == 1) {
            // If the node to be deleted is the head node
            head = head.next;
            if (head != null) {
                head.prev = null;
            }
            return head;
        }

        Node current = head;
        int count = 1;

        // Traverse to the node at the given position
        while (count < position && current != null) {
            current = current.next;
```

```
            count++;
        }

        if (current == null) {
            // If the position is greater than the size of the list
            return head;
        }

        // Update the prev and next pointers of the adjacent nodes
        current.prev.next = current.next;
        if (current.next != null) {
            current.next.prev = current.prev;
        }

        return head;
    }

    public static void printLinkedList(Node head) {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        // Create the doubly linked list: 1 <-> 3 <-> 4
        Node head = new Node(1);
        Node node1 = new Node(3);
        Node node2 = new Node(4);

        head.next = node1;
        node1.prev = head;
        node1.next = node2;
        node2.prev = node1;

        int position = 3;

        System.out.print("Original Linked list: ");
        printLinkedList(head);

        // Delete the node at the given position
        head = deleteNode(head, position);
```

```
        System.out.print("Linked list after deletion: ");
        printLinkedList(head);
    }
}
```