Answer 1)

```
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class DeleteMiddleLinkedList {
    public static ListNode deleteMiddle(ListNode head) {
        if (head == null || head.next == null) {
            return null; // Empty list or only one node
        }

        ListNode slowPtr = head;
        ListNode fastPtr = head;
        ListNode prev = null;

        while (fastPtr != null && fastPtr.next != null) {
            fastPtr = fastPtr.next.next;
            prev = slowPtr;
            slowPtr = slowPtr.next;
        }

        prev.next = slowPtr.next; // Delete the middle node(s)

        return head;
    }

    public static void displayList(ListNode head) {
        ListNode current = head;
        while (current != null) {
            System.out.print(current.val + " ");
            current = current.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        ListNode head1 = new ListNode(1);
        head1.next = new ListNode(2);
        head1.next.next = new ListNode(3);
        head1.next.next.next = new ListNode(4);
        head1.next.next.next.next = new ListNode(5);

        System.out.println("Original Linked List 1:");
        displayList(head1);
        head1 = deleteMiddle(head1);
        System.out.println("Modified Linked List 1:");
        displayList(head1);

        ListNode head2 = new ListNode(2);
        head2.next = new ListNode(4);
        head2.next.next = new ListNode(6);
        head2.next.next.next = new ListNode(7);
        head2.next.next.next.next = new ListNode(5);
        head2.next.next.next.next.next = new ListNode(1);

        System.out.println("Original Linked List 2:");
        displayList(head2);
        head2 = deleteMiddle(head2);
        System.out.println("Modified Linked List 2:");
        displayList(head2);
    }
}
```

Answer 2)

```
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class LinkedListLoopDetection {
    public static boolean hasLoop(ListNode head) {
        if (head == null || head.next == null) {
            return false; // Empty list or only one node, no loop
        }
        ListNode slowPtr = head;
        ListNode fastPtr = head;

        while (fastPtr != null && fastPtr.next != null) {
            slowPtr = slowPtr.next;
            fastPtr = fastPtr.next.next;

            if (slowPtr == fastPtr) {
                return true; // Loop detected
            }
        }

        return false; // No loop detected
    }

    public static void main(String[] args) {
        ListNode head1 = new ListNode(1);
        head1.next = new ListNode(3);
        head1.next.next = new ListNode(4);
        head1.next.next.next = head1.next; // Create a loop

        System.out.println("Linked List 1 has loop: " + hasLoop(head1));

        ListNode head2 = new ListNode(1);
        head2.next = new ListNode(8);
        head2.next.next = new ListNode(3);
        head2.next.next.next = new ListNode(4);

        System.out.println("Linked List 2 has loop: " + hasLoop(head2));
    }
}
```

Answer 3)

```
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class NthNodeFromEnd {
    public static int findNthFromEnd(ListNode head, int n) {
        if (head == null) {
            return -1; // Empty list
        }

        ListNode slowPtr = head;
        ListNode fastPtr = head;

        // Move the fast pointer n nodes ahead of the slow pointer
        for (int i = 0; i < n; i++) {
            if (fastPtr == null) {
                return -1; // n is greater than the number of nodes in the list
            }
            fastPtr = fastPtr.next;
        }

        // Move both pointers simultaneously until the fast pointer reaches the end
        while (fastPtr != null) {
            slowPtr = slowPtr.next;
            fastPtr = fastPtr.next;
        }

        // At this point, the slow pointer is at the Nth node from the end
        return (slowPtr != null) ? slowPtr.val : -1;
    }

    public static void main(String[] args) {
        ListNode head = new ListNode(1);
        head.next = new ListNode(2);
        head.next.next = new ListNode(3);
        head.next.next.next = new ListNode(4);
        head.next.next.next.next = new ListNode(5);

        int N1 = 2;
        int N2 = 5;

        System.out.println("Nth node from end with N = " + N1 + ": " + findNthFromEnd(head, N1));
        System.out.println("Nth node from end with N = " + N2 + ": " + findNthFromEnd(head, N2));
    }
}
```

Answer 5)

```
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class RemoveLoop {
    public static void removeLoop(ListNode head) {
        if (head == null || head.next == null) {
            return; // No loop or empty list
        }

        ListNode slowPtr = head;
        ListNode fastPtr = head;

        // Detect the loop using the Floyd's cycle detection algorithm
        while (fastPtr != null && fastPtr.next != null) {
            slowPtr = slowPtr.next;
            fastPtr = fastPtr.next.next;

            if (slowPtr == fastPtr) {
                break; // Loop detected
            }
        }

        // If there is no loop, return
        if (slowPtr != fastPtr) {
            return;
        }

        // Move the slow pointer to the head and move both pointers at the same pace until they
meet again
        slowPtr = head;
        while (slowPtr.next != fastPtr.next) {
            slowPtr = slowPtr.next;
            fastPtr = fastPtr.next;
        }

        // Unlink the last node from the loop
        fastPtr.next = null;
    }

    public static void printLinkedList(ListNode head) {
        ListNode curr = head;
        while (curr != null) {
            System.out.print(curr.val + " ");
            curr = curr.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        ListNode head1 = new ListNode(1);
        head1.next = new ListNode(3);
        head1.next.next = new ListNode(4);
        head1.next.next.next = head1.next; // Create a loop
        int X1 = 2;

        ListNode head2 = new ListNode(1);
        head2.next = new ListNode(8);
        head2.next.next = new ListNode(3);
        head2.next.next.next = new ListNode(4);
        int X2 = 0;

        ListNode head3 = new ListNode(1);
        head3.next = new ListNode(2);
        head3.next.next = new ListNode(3);
        head3.next.next.next = new ListNode(4);
        head3.next.next.next.next = head3; // Create a loop
        int X3 = 1;

        System.out.println("Linked List before removing loop:");
        printLinkedList(head1);
        removeLoop(head1);
        System.out.println("Linked List after removing loop:");
        printLinkedList(head1);
        System.out.println();

        System.out.println("Linked List before removing loop:");
        printLinkedList(head2);
        removeLoop(head2);
        System.out.println("Linked List after removing loop:");
        printLinkedList(head2);
        System.out.println();

        System.out.println("Linked List before removing loop:");
        printLinkedList(head3);
        removeLoop(head3);
        System.out.println("Linked List after removing loop:");
        printLinkedList(head3);
    }
}
```

Answer 6)

```
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class RetainDeleteLinkedList {
    public static void retainDeleteLinkedList(ListNode head, int M, int N) {
        if (head == null || M <= 0 || N <= 0) {
            return;
        }

        ListNode curr = head;
        ListNode prev = null;

        while (curr != null) {
            // Retain M nodes
            for (int i = 1; i < M && curr != null; i++) {
                curr = curr.next;
            }

            // If there are no more nodes to process, break
            if (curr == null) {
```

```
            break;
        }

        // Delete N nodes
        ListNode next = curr.next;
        for (int i = 0; i < N && next != null; i++) {
            next = next.next;
        }

        curr.next = next;
        curr = next;
    }
}

public static void printLinkedList(ListNode head) {
    ListNode curr = head;
    while (curr != null) {
        System.out.print(curr.val + " ");
        curr = curr.next;
    }
    System.out.println();
}

public static void main(String[] args) {
    ListNode head1 = new ListNode(1);
    head1.next = new ListNode(2);
    head1.next.next = new ListNode(3);
    head1.next.next.next = new ListNode(4);
    head1.next.next.next.next = new ListNode(5);
    head1.next.next.next.next.next = new ListNode(6);
    head1.next.next.next.next.next.next = new ListNode(7);
    head1.next.next.next.next.next.next.next = new ListNode(8);
    int M1 = 2;
    int N1 = 2;

    ListNode head2 = new ListNode(1);
    head2.next = new ListNode(2);
    head2.next.next = new ListNode(3);
    head2.next.next.next = new ListNode(4);
    head2.next.next.next.next = new ListNode(5);
    head2.next.next.next.next.next = new ListNode(6);
    head2.next.next.next.next.next.next = new ListNode(7);
    head2.next.next.next.next.next.next.next = new ListNode(8);
    head2.next.next.next.next.next.next.next.next = new ListNode(9);
```

```
        head2.next.next.next.next.next.next.next.next.next = new ListNode(10);
        int M2 = 3;
        int N2 = 2;

        ListNode head3 = new ListNode(1);
        head3.next = new ListNode(2);
        head3.next.next = new ListNode(3);
        head3.next.next.next = new ListNode(4);
        head3.next.next.next.next = new ListNode(5);
        head3.next.next.next.next.next = new ListNode(6);
        head3.next.next.next.next.next.next = new ListNode(7);
        head3.next.next.next.next.next.next.next = new ListNode(8);
        head3.next.next.next.next.next.next.next.next = new ListNode(9);
        head3.next.next.next.next.next.next.next.next.next = new ListNode(10);
        int M3 = 1;
        int N3 = 1;

        System.out.println("Linked List before operation:");
        printLinkedList(head1);
        retainDeleteLinkedList(head1, M1, N1);
        System.out.println("Linked List after operation:");
        printLinkedList(head1);
        System.out.println();
```

Answer 7)

```
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class InsertAlternate {
    public static void insertAlternate(ListNode first, ListNode second) {
        if (first == null || second == null) {
            return;
        }
        ListNode firstCurr = first;
        ListNode secondCurr = second;

        while (firstCurr != null && secondCurr != null) {
            ListNode firstNext = firstCurr.next;
            ListNode secondNext = secondCurr.next;

            firstCurr.next = secondCurr;
            secondCurr.next = firstNext;

            firstCurr = firstNext;
            secondCurr = secondNext;
        }

        second = secondCurr;
    }

    public static void printLinkedList(ListNode head) {
        ListNode curr = head;
        while (curr != null) {
            System.out.print(curr.val + " ");
            curr = curr.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        ListNode firstHead = new ListNode(5);
        firstHead.next = new ListNode(7);
        firstHead.next.next = new ListNode(17);
        firstHead.next.next.next = new ListNode(13);
        firstHead.next.next.next.next = new ListNode(11);

        ListNode secondHead = new ListNode(12);
        secondHead.next = new ListNode(10);
        secondHead.next.next = new ListNode(2);
        secondHead.next.next.next = new ListNode(4);
        secondHead.next.next.next.next = new ListNode(6);

        System.out.println("First Linked List before operation:");
        printLinkedList(firstHead);
        System.out.println("Second Linked List before operation:");
        printLinkedList(secondHead);
        insertAlternate(firstHead, secondHead);

        System.out.println("First Linked List after operation:");
        printLinkedList(firstHead);
        System.out.println("Second Linked List after operation:");
        printLinkedList(secondHead);
    }
}
```

Answer 8)

```
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class CircularLinkedList {
    public static boolean isCircular(ListNode head) {
        if (head == null) {
            return false;
        }

        ListNode slow = head;
        ListNode fast = head;

        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;

            if (slow == fast) {
                return true;
            }
        }

        return false;
    }
    public static void main(String[] args) {
        ListNode head = new ListNode(1);
        head.next = new ListNode(2);
        head.next.next = new ListNode(3);
        head.next.next.next = new ListNode(4);
        head.next.next.next.next = head;  // Make it circular

        boolean isCircular = isCircular(head);
        System.out.println("Is the linked list circular? " + isCircular);
    }
}
```