Answer 1)

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

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

class LinkedList {
    Node head;

    void addNode(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            Node currentNode = head;
            while (currentNode.next != null) {
                currentNode = currentNode.next;
            }
            currentNode.next = newNode;
        }
    }

    void detectAndRemoveLoop() {
        Node slowPtr = head;
        Node fastPtr = head;
        Node prev = null;

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

            if (slowPtr == fastPtr) {
                break;
            }
        }

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

        // Move slowPtr to the head
        slowPtr = head;

        // Find the node where the loop starts
        while (slowPtr != fastPtr) {
            prev = fastPtr;
            slowPtr = slowPtr.next;
            fastPtr = fastPtr.next;
        }

        // Remove the loop by setting the next pointer of the last node to null
        prev.next = null;
    }

    void printList() {
        Node currentNode = head;
        while (currentNode != null) {
            System.out.print(currentNode.data + " ");
            currentNode = currentNode.next;
        }
        System.out.println();
    }
}

public class Main {
    public static void main(String[] args) {
        LinkedList linkedList = new LinkedList();
        linkedList.addNode(1);
        linkedList.addNode(3);
        linkedList.addNode(4);

        // Create a loop by connecting the last node to the second node
        linkedList.head.next.next.next = linkedList.head.next;

        System.out.println("Original List:");
        linkedList.printList();

        linkedList.detectAndRemoveLoop();

        System.out.println("List after removing the loop:");
        linkedList.printList();
    }
```

```
}
```

Answer 2)

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

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

class LinkedList {
    Node head;

    void addNode(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            Node currentNode = head;
            while (currentNode.next != null) {
                currentNode = currentNode.next;
            }
            currentNode.next = newNode;
        }
    }

    void reverse() {
        Node prev = null;
        Node current = head;
        Node next = null;

        while (current != null) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }
        head = prev;
    }

    void addOne() {
        reverse();

        Node currentNode = head;
        int carry = 1;

        while (currentNode != null) {
            int sum = currentNode.data + carry;
            carry = sum / 10;
            currentNode.data = sum % 10;
            currentNode = currentNode.next;
        }

        if (carry > 0) {
            Node newNode = new Node(carry);
            newNode.next = head;
            head = newNode;
        }

        reverse();
    }

    void printList() {
        Node currentNode = head;
        while (currentNode != null) {
            System.out.print(currentNode.data);
            currentNode = currentNode.next;
        }
        System.out.println();
    }
}

public class Main {
    public static void main(String[] args) {
        LinkedList linkedList = new LinkedList();
        linkedList.addNode(4);
        linkedList.addNode(5);
        linkedList.addNode(6);

        System.out.println("Original List:");
        linkedList.printList();
        linkedList.addOne();

        System.out.println("List after adding 1:");
        linkedList.printList();
    }
}
```

Answer 3)

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

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

class LinkedList {
    Node head;

    void addNode(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            Node currentNode = head;
            while (currentNode.next != null) {
                currentNode = currentNode.next;
            }
            currentNode.next = newNode;
        }
    }

    Node mergeLists(Node list1, Node list2) {
        if (list1 == null)
            return list2;
        if (list2 == null)
```

```
            return list1;

        Node mergedList;

        if (list1.data < list2.data) {
            mergedList = list1;
            mergedList.bottom = mergeLists(list1.bottom, list2);
        } else {
            mergedList = list2;
            mergedList.bottom = mergeLists(list1, list2.bottom);
        }

        mergedList.next = null;
        return mergedList;
    }

    Node flattenList(Node node) {
        if (node == null || node.next == null) {
            return node;
        }

        // Recursively flatten the next pointer
        node.next = flattenList(node.next);

        // Merge the current list with the flattened next list
        node = mergeLists(node, node.next);

        // Return the merged list
        return node;
    }

    void printList() {
        Node currentNode = head;
        while (currentNode != null) {
            System.out.print(currentNode.data + " ");
            currentNode = currentNode.bottom;
        }
        System.out.println();
    }
}

public class Main {
    public static void main(String[] args) {
        LinkedList linkedList = new LinkedList();
        linkedList.addNode(5);
        linkedList.addNode(10);
        linkedList.addNode(19);
        linkedList.addNode(28);

        linkedList.head.bottom = new Node(7);
        linkedList.head.bottom.bottom = new Node(8);
        linkedList.head.bottom.bottom.bottom = new Node(30);

        linkedList.head.next.bottom = new Node(20);

        linkedList.head.next.next.bottom = new Node(22);
        linkedList.head.next.next.next.bottom = new Node(50);

        linkedList.head.next.next.next.next.bottom = new Node(35);
        linkedList.head.next.next.next.next.bottom.bottom = new Node(40);
        linkedList.head.next.next.next.next.bottom.bottom.bottom = new Node(45);

        System.out.println("Original List:");
        linkedList.printList();

        linkedList.head = linkedList.flattenList(linkedList.head);

        System.out.println("Flattened List:");
        linkedList.printList();
    }
}
```

Answer 4)

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

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

class LinkedList {
    Node head;
```

```
void addNode(int data) {
   Node newNode = new Node(data);
   if (head == null) {
      head = newNode;
   } else {
      Node currentNode = head;
      while (currentNode.next != null) {
         currentNode = currentNode.next;
      }
      currentNode.next = newNode;
   }
}

void printList(Node node) {
   Node currentNode = node;
   while (currentNode != null) {
      int arbData = (currentNode.arb != null) ? currentNode.arb.data : -1;
      System.out.println("Data: " + currentNode.data + ", Random: " + arbData);
      currentNode = currentNode.next;
   }
}

Node copyRandomList(Node head) {
   if (head == null)
      return null;

   // Create a mapping between original nodes and their copies
   HashMap<Node, Node> map = new HashMap<>();

   // Create copies of all nodes
   Node current = head;
   while (current != null) {
      map.put(current, new Node(current.data));
      current = current.next;
   }

   // Assign next and random pointers for the copy nodes
   current = head;
   while (current != null) {
      Node copyNode = map.get(current);
      copyNode.next = map.get(current.next);
      copyNode.arb = map.get(current.arb);
      current = current.next;
```

```
        }

        // Return the head of the copied list
        return map.get(head);
    }
}

public class Main {
    public static void main(String[] args) {
        LinkedList linkedList = new LinkedList();
        linkedList.addNode(1);
        linkedList.addNode(2);
        linkedList.addNode(3);
        linkedList.addNode(4);

        linkedList.head.arb = linkedList.head.next;
        linkedList.head.next.arb = linkedList.head.next.next.next;

        System.out.println("Original List:");
        linkedList.printList(linkedList.head);

        Node copiedList = linkedList.copyRandomList(linkedList.head);

        System.out.println("Copied List:");
        linkedList.printList(copiedList);
    }
}
```

Answer 5)

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

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

class LinkedList {
    ListNode head;
    void addNode(int val) {
        ListNode newNode = new ListNode(val);
        if (head == null) {
            head = newNode;
        } else {
            ListNode currentNode = head;
            while (currentNode.next != null) {
                currentNode = currentNode.next;
            }
            currentNode.next = newNode;
        }
    }

    ListNode oddEvenList(ListNode head) {
        if (head == null || head.next == null)
            return head;

        ListNode odd = head;
        ListNode even = head.next;
        ListNode evenHead = even;

        while (even != null && even.next != null) {
            odd.next = even.next;
            odd = odd.next;
            even.next = odd.next;
            even = even.next;
        }

        odd.next = evenHead;
        return head;
    }

    void printList(ListNode head) {
        ListNode currentNode = head;
        while (currentNode != null) {
            System.out.print(currentNode.val + " ");
            currentNode = currentNode.next;
        }
        System.out.println();
    }
}

public class Main {
    public static void main(String[] args) {
        LinkedList linkedList = new LinkedList();
        linkedList.addNode(1);
        linkedList.addNode(2);
        linkedList.addNode(3);
        linkedList.addNode(4);
        linkedList.addNode(5);

        System.out.println("Original List:");
        linkedList.printList(linkedList.head);

        ListNode reorderedList = linkedList.oddEvenList(linkedList.head);

        System.out.println("Reordered List:");
        linkedList.printList(reorderedList);
    }
}
```

Answer 6)

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

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

class LinkedList {
    Node head;

    void addNode(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            Node currentNode = head;
            while (currentNode.next != null) {
                currentNode = currentNode.next;
            }
            currentNode.next = newNode;
        }
    }

    void leftShift(Node head, int k) {
        if (head == null || k == 0)
            return;

        Node current = head;
        int count = 1;

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

        if (current == null)
            return;

        Node kthNode = current;

        while (current.next != null)
            current = current.next;

        current.next = head;
        head = kthNode.next;
        kthNode.next = null;

        this.head = head;
    }

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

public class Main {
    public static void main(String[] args) {
        LinkedList linkedList = new LinkedList();
        linkedList.addNode(2);
        linkedList.addNode(4);
        linkedList.addNode(7);
        linkedList.addNode(8);
        linkedList.addNode(9);

        System.out.println("Original List:");
        linkedList.printList(linkedList.head);

        int k = 3;
        linkedList.leftShift(linkedList.head, k);

        System.out.println("Left-shifted List:");
        linkedList.printList(linkedList.head);
    }
}
```

Answer 7)

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

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

class LinkedList {
    ListNode head;

    void addNode(int val) {
        ListNode newNode = new ListNode(val);
        if (head == null) {
            head = newNode;
        } else {
            ListNode current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newNode;
        }
    }

    int[] nextLargerNodes(ListNode head) {
        int length = getLength(head);
        int[] result = new int[length];
        int index = 0;
        ListNode current = head;

        while (current != null) {
            ListNode nextGreaterNode = getNextGreaterNode(current);
            if (nextGreaterNode != null) {
                result[index] = nextGreaterNode.val;
            }
            index++;
            current = current.next;
        }

        return result;
    }

    int getLength(ListNode head) {
        int length = 0;
        ListNode current = head;
        while (current != null) {
            length++;
            current = current.next;
        }
        return length;
    }

    ListNode getNextGreaterNode(ListNode node) {
        int target = node.val;
        ListNode current = node.next;
        while (current != null) {
            if (current.val > target) {
                return current;
            }
            current = current.next;
        }
        return null;
    }
}

public class Main {
    public static void main(String[] args) {
        LinkedList linkedList = new LinkedList();
        linkedList.addNode(2);
        linkedList.addNode(1);
        linkedList.addNode(5);

        System.out.println("Original List:");
        linkedList.printList(linkedList.head);

        int[] result = linkedList.nextLargerNodes(linkedList.head);

        System.out.println("Next Greater Nodes:");
        for (int value : result) {
            System.out.print(value + " ");
        }
        System.out.println();
    }
}
```

Answer 8)

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

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

class LinkedList {
    ListNode head;

    void addNode(int val) {
        ListNode newNode = new ListNode(val);
        if (head == null) {
            head = newNode;
        } else {
            ListNode current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newNode;
        }
    }

    ListNode removeZeroSumSublists(ListNode head) {
        ListNode dummy = new ListNode(0);
        dummy.next = head;

        ListNode current = dummy;
        while (current != null) {
            int sum = 0;
            ListNode runner = current.next;
            while (runner != null) {
                sum += runner.val;
                if (sum == 0) {
                    current.next = runner.next;
                    break;
                }
                runner = runner.next;
            }
            if (runner == null) {
                current = current.next;
            }
        }

        return dummy.next;
    }

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

public class Main {
    public static void main(String[] args) {
        LinkedList linkedList = new LinkedList();
        linkedList.addNode(1);
        linkedList.addNode(2);
        linkedList.addNode(-3);
        linkedList.addNode(3);
        linkedList.addNode(1);

        System.out.println("Original List:");
        linkedList.printList(linkedList.head);

        ListNode result = linkedList.removeZeroSumSublists(linkedList.head);

        System.out.println("Modified List:");
        linkedList.printList(result);
    }
}
```