Linked Lists in Java
Assignment Questions

Q1. Given a linked list and a key ‘X‘ in, the task is to check if X is present in the linked list or not.

Examples:
input:14->21->11->30->10, X = 14
output:Yes

Explanation: 14 is present in the linked list.
Input :6->21->17->30->10->8, X = 13
Output:No
Ans:

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

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

// LinkedList class
class LinkedList {
    Node head;

    // Append node to the list
    public void append(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            return;
        }
        Node lastNode = head;
        while (lastNode.next != null) {
            lastNode = lastNode.next;
        }
        lastNode.next = newNode;
    }

    // Search for a node
    public String search(int x) {
        Node current = head;            }
        while (current != null) {
            if (current.data == x) {
                return "Yes";

            current = current.next;
        }
        return "No";
    }
}

public class Main {
    public static void main(String[] args) {
        LinkedList llist1 = new LinkedList();
        llist1.append(14);
        llist1.append(21);
        llist1.append(11);
        llist1.append(30);
        llist1.append(10);
        System.out.println(llist1.search(14));
```

Q2. Insert a node at the given position in a linked list. We are given a pointer to a node, and the new node is inserted after the given node.
Input :LL = 1 -> 2 -> 4 -> 5 -> 6 pointer = 2 value = 3.
output:1 -> 2 -> 3 -> 4 -> 5 -> 6

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

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

// LinkedList class
class LinkedList {
    Node head;

    // Append node to the list
    public void append(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            return;
        }

        Node lastNode = head;
        while (lastNode.next != null) {
            lastNode = lastNode.next;
        }
        lastNode.next = newNode;
    }

    // Insert node after a given node
    public void insertAfter(int prevNodeData, int newNodeData) {
        Node newNode = new Node(newNodeData);
        Node current = head;

        while (current != null && current.data != prevNodeData) {
            current = current.next;
        }

        if (current == null) {
            System.out.println("Previous node not found.");
            return;
        }

        newNode.next = current.next;
        current.next = newNode;
    }

    // Print linked list
    public void printList() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " -> ");
            current = current.next;
        }
        System.out.println("null");
    }
}

public class Main {
    public static void main(String[] args) {
        LinkedList llist = new LinkedList();
        llist.append(1);
        llist.append(2);
        llist.append(4);
        llist.append(5);
        llist.append(6);
        System.out.println("Original Linked List:");
        llist.printList();

        llist.insertAfter(2, 3);

        System.out.println("Linked List after insertion:");
        llist.printList();
    }
}
```

Q3. Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.
Example1:
Input:head = [1,1,2]

Output:[1,2]
Ans:

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

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

// Solution class
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode current = head;
        while (current != null && current.next != null) {
            if (current.val == current.next.val) {
                current.next = current.next.next;
            } else {
                current = current.next;
            }
        }

        return head;
    }

    // Print linked list
    public void printList(ListNode head) {
        while (head != null) {
            System.out.print(head.val + " -> ");
            head = head.next;
        }
        System.out.println("null");
    }
}

public class Main {
    public static void main(String[] args) {
        Solution solution = new Solution();

        // Example 1
        ListNode head1 = new ListNode(1);
        head1.next = new ListNode(1);
        head1.next.next = new ListNode(2);

        System.out.println("Original Linked List:");
        solution.printList(head1);

        head1 = solution.deleteDuplicates(head1);

        System.out.println("Linked List after deleting duplicates:");
        solution.printList(head1);
    }
}
```

Q4. Given the head of a singly linked list, return true if it is a palindrome or false otherwise.

Example 1:
Input:head = [1,2,2,1]
Output:true

Example 2:
Input:head = [1,2]
Output:false
Ans:

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

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

// Solution class
class Solution {
    public boolean isPalindrome(ListNode head) {
        // Find the end of first half and reverse second half.
        ListNode firstHalfEnd = endOfFirstHalf(head);
        ListNode secondHalfStart = reverseList(firstHalfEnd.next);
        // Check whether or not there's a palindrome.
        ListNode p1 = head;
        ListNode p2 = secondHalfStart;
        boolean result = true;
        while (result && p2 != null) {
            if (p1.val != p2.val) result = false;
            p1 = p1.next;
            p2 = p2.next;
        }

        // Restore the list and return the result.
        firstHalfEnd.next = reverseList(secondHalfStart);
        return result;
    }

    // Reverse the linked list
    private ListNode reverseList(ListNode head) {
        ListNode previous = null;
        ListNode current = head;
        while (current != null) {
            ListNode nextTemp = current.next;
            current.next = previous;
            previous = current;
            current = nextTemp;
        }
        return previous;
    }

    // Find the end of the first half
    private ListNode endOfFirstHalf(ListNode head) {
        ListNode slow = head;
        ListNode fast = head;
        while (fast.next != null && fast.next.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }

    // Print linked list
    public void printList(ListNode head) {
        while (head != null) {
            System.out.print(head.val + " -> ");
            head = head.next;
        }
        System.out.println("null");
    }
}

public class Main {
    public static void main(String[] args) {
        Solution solution = new Solution();

        // Example 1
        ListNode head1 = new ListNode(1);
        head1.next = new ListNode(2);
        head1.next.next = new ListNode(2);
        head1.next.next.next = new ListNode(1);

        System.out.println("Linked List:");
        solution.printList(head1);

        System.out.println("Is Palindrome? " +
solution.isPalindrome(head1));

        // Example 2
        ListNode head2 = new ListNode(1);
        head2.next = new ListNode(2);

        System.out.println("Linked List:");
        solution.printList(head2);

        System.out.println("Is Palindrome? " +
solution.isPalindrome(head2));
```

```
    }
}
```

Q5. Given two numbers represented by two lists, write a function that returns the sum list. The sum list is a list representation of the addition of two input numbers.
Example:
Input:

List1: 5->6->3
List2: 8->4->2
Output:

Resultant list: 1->4->0->5
Explanation: 563 + 842 = 1405

Example2:
Input
List1: 7->5->9->4->6
List2: 8->4
Output:

Resultant list: 7->6->0->3->0
Explanation: 75946+84=76030

Ans:

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

// Solution class
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode dummyHead = new ListNode(0);
        ListNode current = dummyHead;
        int carry = 0;

        while (l1 != null || l2 != null) {
            int x = (l1 != null) ? l1.val : 0;
            int y = (l2 != null) ? l2.val : 0;
            int sum = carry + x + y;
            carry = sum / 10;

            current.next = new ListNode(sum % 10);
            current = current.next;

            if (l1 != null) l1 = l1.next;
            if (l2 != null) l2 = l2.next;
        }

        if (carry > 0) {
            current.next = new ListNode(carry);
        }

        return dummyHead.next;
    }

    // Print linked list
    public void printList(ListNode head) {
        while (head != null) {
            System.out.print(head.val + " -> ");
            head = head.next;
        }
        System.out.println("null");
    }
}

public class Main {
    public static void main(String[] args) {
        Solution solution = new Solution();

        // Example 1
        ListNode l1 = new ListNode(5);
        l1.next = new ListNode(6);
        l1.next.next = new ListNode(3);

        ListNode l2 = new ListNode(8);
        l2.next = new ListNode(4);
        l2.next.next = new ListNode(2);

        System.out.println("List1:");
        solution.printList(l1);
        System.out.println("List2:");
        solution.printList(l2);

        ListNode result = solution.addTwoNumbers(l1, l2);
        System.out.println("Resultant list:");
        solution.printList(result);

        // Example 2
        ListNode l3 = new ListNode(7);
        l3.next = new ListNode(5);
        l3.next.next = new ListNode(9);
        l3.next.next.next = new ListNode(4);
        l3.next.next.next.next = new ListNode(6);

        ListNode l4 = new ListNode(8);
        l4.next = new ListNode(4);

        System.out.println("List1:");
        solution.printList(l3);
        System.out.println("List2:");
        solution.printList(l4);

        ListNode result2 = solution.addTwoNumbers(l3, l4);
        System.out.println("Resultant list:");
        solution.printList(result2);
    }
}
```