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

Program:

```
package LinkedList;

import java.util.Scanner;

public class FindElementAss1 {
   Node head;
   class Node{
      int data;
      Node next;
      Node(int data){
         this.data = data;
         next = null;
      }
   }
   public void push(int newData){
      Node newNode = new Node(newData);
      newNode.next = head;
      head = newNode;
   }

   public boolean search(int target){
      Node temp = head;
      while(temp!=null){
         if(temp.data == target){
            return true;
         }
         temp = temp.next;
      }
      return false;
   }
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      FindElementAss1 fe = new FindElementAss1();
      fe.push(14);
      fe.push(21);
      fe.push(11);
```

```
        fe.push(30);
        fe.push(10);

        System.out.println("Enter the number you want to search: ");
        int key = sc.nextInt();

        if(fe.search(key)){
            System.out.println(key+" is present in the list or not? ="+"Yes");
        }
        else{
            System.out.println(key+" is present in the list or not? ="+"No");
        }
    }
}
```

Input2: 6->21->17->30->10->8, X = 13
Output: No
Explanation: 13 is not present in the linked list

Program:

```
package LinkedList;

import java.util.Scanner;

public class FindElementAss1 {
    Node head;
    class Node{
        int data;
        Node next;
        Node(int data){
            this.data = data;
            next = null;
        }
    }
    public void push(int newData){
        Node newNode = new Node(newData);
        newNode.next = head;
        head = newNode;
    }

    public boolean search(int target){
        Node temp = head;
        while(temp!=null){
            if(temp.data == target){
                return true;
            }
```

```
            temp = temp.next;
        }
        return false;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        FindElementAss1 fe = new FindElementAss1();
        fe.push(6);
        fe.push(21);
        fe.push(17);
        fe.push(30);
        fe.push(10);
        fe.push(8);

        System.out.println("Enter the number you want to search: ");
        int key = sc.nextInt();

        if(fe.search(key)){
            System.out.println(key+" is present in the list or not? = "+"Yes");
        }
        else{
            System.out.println(key+" is present in the list or not? = "+"No");
        }
    }
}
```

Output:

```
Enter the number you want to search:
13
13 is present in the list or not? = No

Process finished with exit code 0
```

Approach:

- We have simply traversed the linked list and checked for every node whether its data is equal to the target.
- If found we returned true else if the iterator reached the end of the linked list we returned false.

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.

Input1: LL = 1 -> 2 -> 4 -> 5 -> 6
pointer = 2 value = 3.
Output1: 1 -> 2 -> 3 -> 4 -> 5 -> 6

Approach:

- First, a new node with a given element is created. If the insert position is 1, then the new node is made to head.
- Otherwise, traverse to the node that is previous to the insert position and check if it is null or not. In case of null, the specified position does not exist.
- In other case, assign the next of the new node as the next of the previous node and the next of the previous node as new node.

Program:

```
package LinkedList;
import java.util.Scanner;
public class InsertNodeAss2 {
   Node head;
   class Node {
      int data;
      Node next;
      Node(int data) {
         this.data = data;
         next = null;
      }
   }
   public void push(int newData) {
      Node newNode = new Node(newData);
      newNode.next = head;
      head  = newNode;
   }

   public void insertAtPos(int pos, int key){
      Node temp = head;
      int count = 0;
      while(temp!=null){
         count++;
         if(count == pos){
            Node newNode = new Node(key);
            newNode.next = temp.next;
            temp.next = newNode;
            return;
         }
         temp = temp.next;
      }
   }
   public void display(){
      Node temp = head;
      while(temp!=null){
         System.out.print(temp.data+"->");
```

```
            temp = temp.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        InsertNodeAss2 node = new InsertNodeAss2();
//        inserting nodes from beginning
        node.push(6);
        node.push(5);
        node.push(4);
        node.push(2);
        node.push(1);

        System.out.println("Before inserting the node elements are: ");
        node.display();


        System.out.print("Enter the position: ");
        int pos = sc.nextInt();
        System.out.print("Enter the key for insert: ");
        int key = sc.nextInt();

        node.insertAtPos(pos, key);
        System.out.println("After inserting the node elements are: ");
        node.display();
    }
}
```

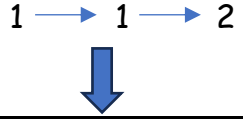
Output:

```
"C:\Users\rajan\Desktop\PW Skill\JAVA-With-DSA\out\production\JAVA-With-DSA" LinkedList.InsertNodeAss2
Before inserting the node elements are:
1->2->4->5->6->
Enter the position: 2
Enter the key for insert: 3
After inserting the node elements are:
1->2->3->4->5->6->

Process finished with exit code 0
```

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.

Example 1:



Input: head = [1,1,2]

Output: [1,2]

Program:

```
package LinkedList;

class Node{
   int data;
   Node next;
   Node(int newData){
      data = newData;
      next = null;
   }
}
public class DeleteDupAss3 {
   Node head;

   public void insertAtEnd(int newData){
      Node newNode = new Node(newData);
      if(head == null){
         head = new Node(newData);
         return;
      }
      // List is not empty
      newNode.next = null;
      Node current = head;

      while(current.next != null){
         current = current.next;
      }
      current.next = newNode;
      return;
   }
   public void deleteDuplicates() {
      if(head == null || head.next == null)
         return ;
      Node curr = head;

      while( curr != null && curr.next != null){

         if(curr.data == curr.next.data){
```

```
            curr.next = curr.next.next;
        }
        else{
            curr = curr.next;
        }
    }
    return;
  }
  void PrintList() {
    Node temp = head;
    if(temp != null) {
        System.out.print("The list contains: ");
        while(temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    } else {
        System.out.println("The list is empty.");
    }
  }

  public static void main(String[] args){
    DeleteDupAss3 dda = new DeleteDupAss3();
    dda.insertAtEnd(1);
    dda.insertAtEnd(1);
    dda.insertAtEnd(2);

    System.out.println("List before deletion: ");
    dda.PrintList();


    dda.deleteDuplicates();
    System.out.println("List after Deletion");
    dda.PrintList();
  }
}
```

Output:

```
List before deletion:
The list contains: 1 1 2
List after Deletion
The list contains: 1 2
```

Approach:

- Traverse all element through a while loop if curr node and the next node of curr node are present.
- If the value of curr is equal to the value of prev.
- It means the value is present in the linked list.
- So we can skip the 'curr' pointer and prev.next can point to curr.next. By this we ensure that prev now doesn't points to curr anymore. The node curr is not pointed anymore, so it would be h=garbage collected automatically by JVM.
- Hence, we do not need to include curr again in the linked list, so we increment the value of curr otherwise, we increment the curr pointer.

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

Example1:

Input: head = [1,2,2,1]

Output: true

```
package LinkedList;

import java.util.ArrayList;

public class PalindromeAss4 {
   Node head;
   class Node{
      int data;
      Node next;
      Node(int data){
         this.data = data;
         next = null;
      }
   }

//    Inserting an element.
   public void insertAtEnd(int newData){
      Node newNode = new Node(newData);
      if(head == null){
         head = new Node(newData);
         return;
      }
      // List is not empty
      newNode.next = null;
      Node current = head;

      while(current.next != null){
```

```
            current = current.next;
        }
        current.next = newNode;
        return;
    }

    public boolean isPalindrome(Node head){
        ArrayList<Integer> arrayList = new ArrayList<>();
        Node temp = head;
        while(temp!=null){
            arrayList.add(temp.data);
            temp = temp.next;
        }

        int low = 0;
        int high = arrayList.size()-1;

        while(low<high){
            if(!arrayList.get(low).equals(arrayList.get(high))){
                return false;
            }
            low = low+1;
            high = high-1;
        }
        return true;
    }

    public static void main(String[] args){
        PalindromeAss4 pa = new PalindromeAss4();

        pa.insertAtEnd(1);
        pa.insertAtEnd(2);
        pa.insertAtEnd(2);
        pa.insertAtEnd(1);

        System.out.println(pa.isPalindrome(pa.head));

    }
}
```

Output:

```
"C:\Users\rajan\Desktop\PW Skill\JAVA-With-DSA\out\production\JAVA-With-DSA" LinkedList.PalindromeAss4
true

Process finished with exit code 0
```

Example2:

Input: head = [1,2]

Output: false

Program:

```
package LinkedList;

import java.util.ArrayList;

public class PalindromeAss4 {
   Node head;
   class Node{
      int data;
      Node next;
      Node(int data){
         this.data = data;
         next = null;
      }
   }

//    Inserting an element.
   public void insertAtEnd(int newData){
      Node newNode = new Node(newData);
      if(head == null){
         head = new Node(newData);
         return;
      }
      // List is not empty
      newNode.next = null;
      Node current = head;

      while(current.next != null){
         current = current.next;
      }
      current.next = newNode;
      return;
   }

   public boolean isPalindrome(Node head){
      ArrayList<Integer> arrayList = new ArrayList<>();
      Node temp = head;
      while(temp!=null){
         arrayList.add(temp.data);
         temp = temp.next;
      }

      int low = 0;
```

```
        int high = arrayList.size()-1;

        while(low<high){
            if(!arrayList.get(low).equals(arrayList.get(high))){
                return false;
            }
            low = low+1;
            high = high-1;
        }
        return true;
    }

    public static void main(String[] args){
        PalindromeAss4 pa = new PalindromeAss4();

        pa.insertAtEnd(1);
        pa.insertAtEnd(2);

        System.out.println(pa.isPalindrome(pa.head));

    }
}
```

Output:

```
"C:\Users\rajan\Desktop\PW Skill\JAVA-With-DSA\out\production\JAVA-With-DSA" LinkedList.PalindromeAss4
false

Process finished with exit code 0
```

Approach:

- The idea is to first reverse the second half part of the linked list and then check whether the list is palindrome or not.
- Get the middle of the linked list and reverse the second half of the linked list.
- Check if the first half and second half are identical.
- Construct the original linked list by reversing the second half again and attaching it back to the first half.

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.

Input: List1: 5->6->3 // Represent number 563
List2: 8->4->2 // Represent number 842

Output:
Resultant list: 1->4->0->5 // Represent number 1405
Explanation: 563 +842 = 1405

Input:
List1: 7->5->9->4->6 // Represent number 75946
List2: 8->4 // Represent number 84
Output:
Resultant list: 7 -> 6 -> 0 -> 3 -> 0 // Represent number 76030
Explanation: 75946 + 84 76030

Program:

```
package LinkedList;

public class AdditionOfTwoList {
   static Node head1, head2;

   static class Node {

      int data;
      Node next;
      Node(int d) {
         data = d;
         next = null;
      }
   }

   void addTwoLists(Node first, Node second) {
      Node start1 = new Node(0);
      start1.next = first;
      Node start2 = new Node(0);
      start2.next = second;

      addPrecedingZeros(start1, start2);
      Node result = new Node(0);
      if (sumTwoNodes(start1.next, start2.next, result) == 1) {
         Node node = new Node(1);
         node.next = result.next;
         result.next = node;
      }
      printList(result.next);
   }

   private int sumTwoNodes(Node first, Node second, Node result) {
      if (first == null) {
         return 0;
```

```
    }
    int number = first.data + second.data + sumTwoNodes(first.next, second.next, result);
    Node node = new Node(number % 10);
    node.next = result.next;
    result.next = node;
    return number / 10;
}
private void addPrecedingZeros(Node start1, Node start2) {
    Node next1 = start1.next;
    Node next2 = start2.next;
    while (next1 != null && next2 != null) {
        next1 = next1.next;
        next2 = next2.next;
    }
    if (next1 == null && next2 != null) {
        while (next2 != null) {
            Node node = new Node(0);
            node.next = start1.next;
            start1.next = node;
            next2 = next2.next;
        }
    } else if (next2 == null && next1 != null) {
        while (next1 != null) {
            Node node = new Node(0);
            node.next = start2.next;
            start2.next = node;
            next1 = next1.next;
        }
    }
}

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

public static void main(String[] args) {
    AdditionOfTwoList list = new AdditionOfTwoList();

    // creating first list
    list.head1 = new Node(7);
    list.head1.next = new Node(5);
    list.head1.next.next = new Node(9);
```

```
        list.head1.next.next.next = new Node(4);
        list.head1.next.next.next.next = new Node(6);
        System.out.print("First List is ");
        list.printList(head1);

        // creating second list
        list.head2 = new Node(8);
        list.head2.next = new Node(4);
        System.out.print("Second List is ");
        list.printList(head2);

        System.out.print("Resultant List is ");
        // add the two lists and see the result
        list.addTwoLists(head1, head2);
    }
}
```

Approach:

- Traverse both lists to the end and add preceding zeros in the list with lesser digits.
- Then call a recursive function on the start nodes of both lists which calls itself for the next nodes of both lists till it sets to the end.
- This function creates a node for the sum of the current digits and returns the carry.
- Traverse the two linked lists in order to add preceding zeros in case a list is having lesser digits than the other one.
- Start from the head node of both lists and call a recursive function for the next nodes.
- Continue it till the end of the lists.
- Creates a node for current digits sum and returns the carry