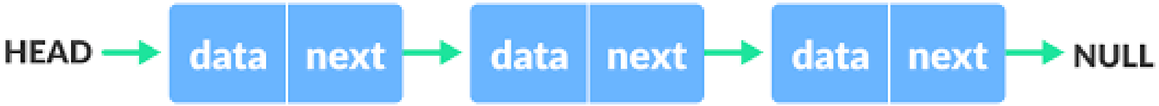


Linked List

```
import java.util.*;

public class LL {

    private Node head;
    private Node tail;
    private int size;

    public LL() {
        this.size = 0;
    }

// Code for reating the block
    private class Node {
        private int value;
        private Node next;

        public Node(int value) {
            this.value = value;
        }

        public Node(int value, Node next) {
            this.value = value;
            this.next = next;
        }
    }

    // Insert the value in the beginning of the LL
    public void insertFirst(int val) {
        Node node = new Node(val);
        node.next = head;
        head = node;

        if (tail == null) {
            tail = head;
        }
        size += 1;
    }

// Insert the value at the end of the LL
    public void insertLast(int val) {

        if (tail == null) { // empty LL so insert the Node in the beginning and don't add th enode in the
                            // last
            insertFirst(val);
            return;
        }

        Node node = new Node(val);
        tail.next = node; //adding a node to the end
        tail = node;     // assigning it as the tail
        size++;
    }

    // insert an element anywhere in the LL
    public void insert(int index, int val) {
        if (index == 0) {
            insertFirst(val);
            return;
        }

        if (index == size) {
            insertLast(val);
            return;
        }

        Node temp = head; // to iterate thru nodes, use temp,keeping the head it its place

        for (int i = 1; i < index; i++) {
            temp = temp.next;
        }

        Node node = new Node(val, temp.next);
        temp.next = node;
        size++;
    }

    // Delete the first Node
    public int deleteFirst() {
        int val = head.value;
        head = head.next;

        if (head == null) {
            tail = null;
        }

        size--;
        return val;
    }

    public int deleteLast() {
        if (size <= 1) {
            return deleteFirst();
        }

        Node secondLast = get(size - 2); // getting the 2nd last node
        int val = tail.value; // value of the current tail
        tail = secondLast;
        tail.next = null;
        return val;
    }

    // deleting node at any index
    public int delete(int index) {
        if (index == 0) {
            return deleteFirst();
        }

        if (index == size - 1) {
            return deleteLast();
        }

        Node prev = get(index - 1); // node before the index to be deleted
        int val = prev.next.value;
        prev.next = prev.next.next; // breaks the chain and links one block to the next of next block
        return val;
    }

    // Search for a node by passing its value
    public Node find(int value) {
        Node node = head;
        while (node != null) {
            if (node.value == value) {
                return node;
            }
            node = node.next;
        }
        return null;
    }

    public Node get(int index) { // required for the deleting nodes in order to fetch the required nodes needed
                                 // for deletion
        Node node = head;

        for (int i = 0; i < index; i++) {
            node = node.next;
        }

        return node;
    }

    public void display() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.value + " -> ");
            temp = temp.next;
        }
        System.out.println("END");
    }


    public static void main(String[] args) {
        LL list = new LL();
        list.insertFirst(3);
        list.insertFirst(6);
        list.insertFirst(38);
        list.insertFirst(23);
        list.insertLast(53);
        list.insert(2, 35);
        list.display();
        System.out.println(list.delete(3));
        list.display();
    }
}
```

DOUBLY LINKED LIST -

```
import java.util.*;

public class LL {

    private Node head;

    public void insertFirst(int val) {
        Node node = new Node(val);
        node.next = head;
        node.prev = null;
        if (head != null) { // if DLL is empty, it will give a null pointer exception
            head.prev = node;
        }
        head = node;
    }

    // Code for creating the block
    private class Node {
        int val;
        Node next;
        Node prev;

        public Node(int val) {
            this.val = val;
        }

        public Node(int val, Node next, Node prev) {
            this.val = val;
            this.next = next;
            this.prev = prev;
        }
    }

    public void insertLast(int val) {
        Node node = new Node(val);
        Node last = head;
        node.next = null; // last node's next pointer will be null
        if (head == null) {
            node.prev = null;
            head = node;
            return;
        }

        while (last.next != null) {
            last = last.next;
        }

        last.next = node;
        node.prev = last;
    }

    // Search for a node by passing its value
    public Node find(int val) {
        Node node = head;
        while (node != null) {
            if (node.val == val) {
                return node;
            }
            node = node.next;
        }
        return null;
    }

    // function that adds a block after the mentioned value
    public void insertAfter(int after, int val) {
        Node p = find(after);

        if (p == null) {
            System.out.println("Does not exist");
            return;
        }

        Node node = new Node(val);
        node.next = p.next;
        p.next = node;
        node.prev = p;

        if (node.next != null) { // In case p is the last element of the DLL
            node.next.prev = node; // i.e. the previous of the block after p (node.next).prev
        }
    }

    public void display() {
        Node node = head;
        Node last = null;
        while (node != null) {
            System.out.print(node.val + "->");
            last = node; // used for reverse printing operation to save th evalue of the last node
            node = node.next;
        }
        System.out.println("END");

        System.out.println("Reverse Printing - ");
        while (last != null) {
            System.out.print(last.val + "->");
            last = last.prev;
        }

        System.out.println("START");
    }


    public static void main(String[] args) {
        LL list = new LL();
        list.insertFirst(3);
        list.insertFirst(6);
        list.insertFirst(38);
        list.insertFirst(23);
        list.insertLast(51);
        list.insertAfter(6, 86);
        list.insertAfter(51, 86);
        list.display();
    }
}
```

CIRCULAR LINKED LIST -

```
import java.util.*;

public class LL {

    private Node head;
    private Node tail;

    public LL() {
        this.head = head;
        this.tail = tail;
    }

    public void insert(int val) {
        Node node = new Node(val);

        if (head == null) {
            head = node;
            tail = node;
            return;
        }

        tail.next = node;
        node.next = head;
        tail = node;
    }
    public void delete(int val) {
        Node node = head;
        if (node == null) {
            return;
        }

        if (node.val == val) { // if u want to delet the head
            head = head.next;
            tail.next = head;
        }

        do {
            Node n = node.next;
            if (n.val == val) {
                node.next = n.next;
                break;
            }

            node = node.next;
        } while (node != head);
    }

    public void display() {
        Node node = head;

        if (head != null) {
            do {
                System.out.print(node.val + "->");
                node = node.next;
            } while (node != head);
        }

        System.out.println("HEAD");
    }

    // Code for creating the block
    private class Node {
        int val;
        Node next;

        public Node(int val) {
            this.val = val;
        }

        public Node(int val, Node next) {
            this.val = val;
            this.next = next;
        }
    }

    public static void main(String[] args) {
        LL list = new LL();
        list.insert(53);
        list.insert(46);
        list.insert(71);
        list.insert(86);
        list.delete(71);
        list.display();

    }
}
```

> Insert a node in a singly LL using recursion

```
// insert using recursion

    public void insertRecursion(int value, int index) {
        head = insertRec(value, index, head);
    }

private Node insertRec(int value, int index, Node node) {
        if (index == 0) {
            Node temp = new Node(value, node);
            size++;
            return temp;
        }

        node.next = insertRec(value, index--, node.next);
        return node;
    }
```

83. Remove Duplicates from Sorted List

```
class Solution {
    public ListNode deleteDuplicates(ListNode node) {
       if(node==null){
           return node ;
       }
       ListNode head = node ;
        while(node.next!=null){
            if(node.val==node.next.val){
                node.next=node.next.next;
            }else{
                node=node.next ;
            }
        }

return head ;
    }
}
```

1290. Convert Binary Number in a Linked List to Integer

```
class Solution {
    public int getDecimalValue(ListNode head) {
        int number = 0 ;
        ListNode temp=head;
        ListNode temp1=head;
        int size = 0;
        while(temp!=null){

            size++ ;
            temp = temp.next;
        }
        size--;
        while(temp1!=null){
        number+= temp1.val*(int)Math.pow(2,size--);
        temp1=temp1.next  ;
        }

        return number ;
    }
}
```

876. Middle of the Linked List

```
class Solution {
    public ListNode middleNode(ListNode head) {
        ListNode temp = head ;
        ListNode temp1 = head ;
        int size =0 ;
        while(temp!=null){
            size++ ;
            temp=temp.next;
        }

       for(int i =0 ;i<size/2 ; i++){
           temp1=temp1.next;
       }
return temp1;

    }
}
```

206. Reverse Linked List

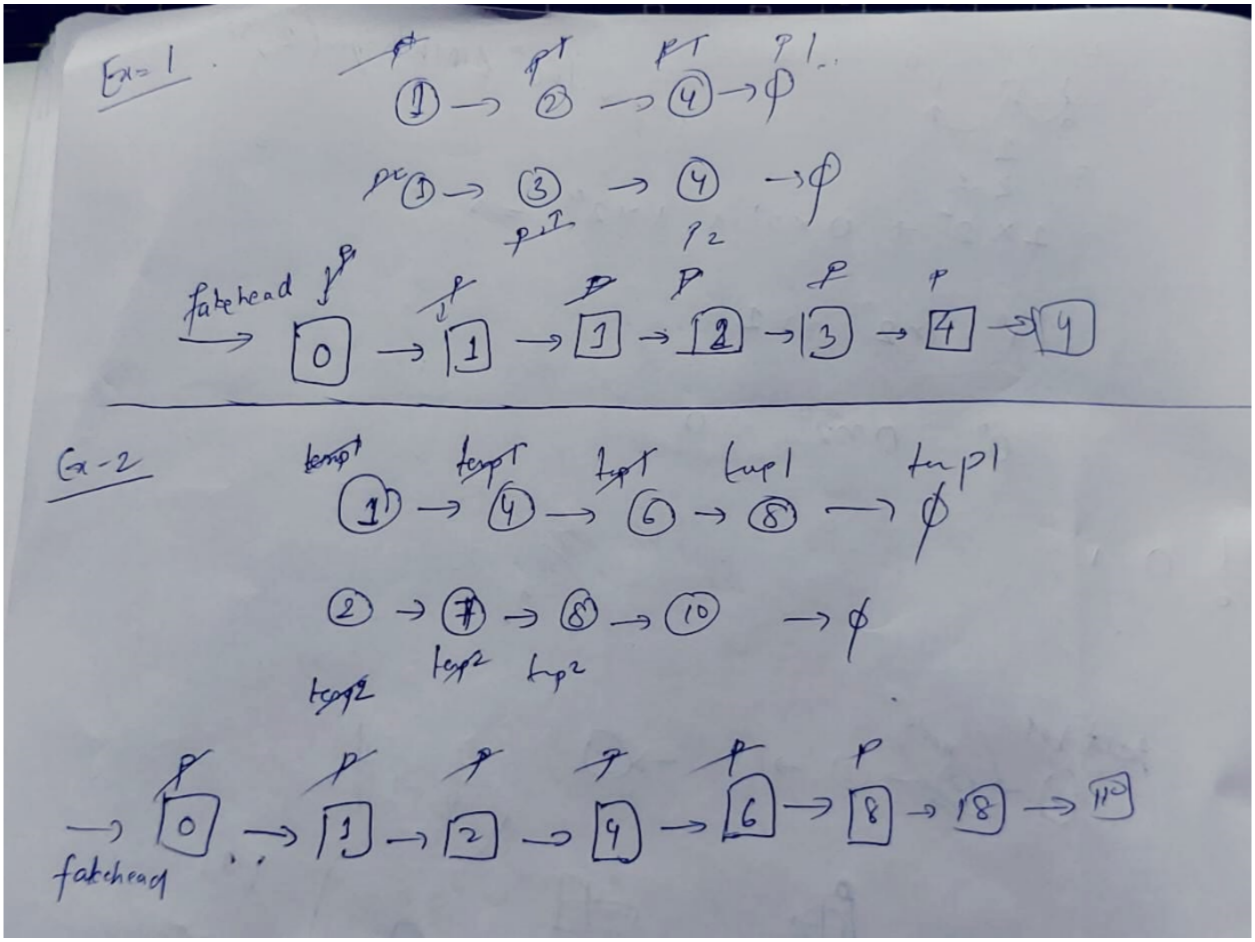
Reverse a Singly Linked List in Java | Leetcode #206 | Data Structures & Algorithms

► ►Personal queries? - Follow me on LinkedIn - https://www.linkedin.com/in/dinesh-varyani/ ► In this video we will learn how to reverse a singly linked list in Java. The Leetcode problem states - Given the head of a singly linked list, reverse the list, and return the reversed list.

https://www.youtube.com/watch?v=jY-EUKXYT20

```
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode prev = null;
        ListNode current = head;


        while(current != null) {
            ListNode next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }
       return prev;
    }
}
```

21. Merge Two Sorted Lists



```
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
      ListNode temp1= list1;
      ListNode temp2= list2 ;
      //Initializing new LL with a node consiting 0
      ListNode fakeHead= new ListNode(0) ;
      // p is used for moving forward and adding new nodes to the list
      ListNode p = fakeHead ;

      while(temp1!=null && temp2!=null){
          if(temp1.val<=temp2.val){
              p.next=temp1 ; // next node of the LL will be temp1
              temp1=temp1.next ; // traversing forward in list1
          }
          else{
              p.next=temp2 ;  // next node of the LL will be temp2
              temp2=temp2.next; // traversing forward in list2
          }

         p= p.next; // moving to the next node in the final LL
      }

      if(temp1!=null){  // In case list2 ends before list1
        // add the remaining elements of list1 to the final LL
       p.next= temp1 ;
      }

      if(temp2!=null){ // In case, list1 ends before list2
         // add the remaining elements of the list2 to the final LL
          p.next=temp2 ;
      }
 // fakeHead points to 0, sorted list starts from fakeHead.next
      return fakeHead.next;
    }
}
```

160. Intersection of Two Linked Lists

Intersection point of two Linked Lists | Amazon | Microsoft | Brute | Better | Optimal1 | Optimal2

Check our Website: https://www.takeuforward.org/

In case you are thinking to buy courses, please check below:

▶ https://www.youtube.com/watch?v=u4FWXfgS8jw

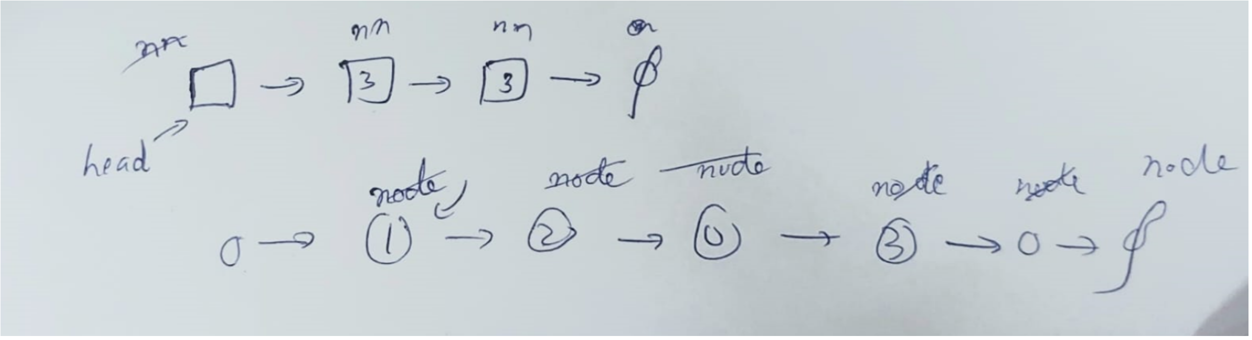
```
public class Solution {
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        int sizeA = 0 ;
        int sizeB = 0 ;
        ListNode tempA =headA ;
        ListNode tempB= headB ;
        int diff =0 ;

        while(tempA!=null){
             sizeA++ ;
             tempA=tempA.next;
        }

        while(tempB!=null){
             sizeB++ ;
             tempB=tempB.next;
        }

      tempA=headA ; // currently tempA and tempB points to null
      tempB=headB;
        if(sizeA>sizeB){
           diff = sizeA-sizeB ;
// if size of (listA>listB),shifting the pointer in listA to a position equivalent to the pointer in listB
            for( int i =0;i<diff; i++){
               tempA=tempA.next;
           }
        }

// if size of (listB>listA),shifting the pointer in listB to a position equivalent to the pointer in listA

        if(sizeA<sizeB){
           diff = sizeB-sizeA ;

           for( int i =0;i<diff; i++){
               tempB=tempB.next;
           }
        }
// Checking for intersection of nodes
        while(tempA!=null && tempB!=null){
            if(tempA==tempB){
                return tempA;
            }

            else{
                tempA=tempA.next ;
                tempB=tempB.next ;
            }
        }
// if no intersection found, retun null
  return null ;
    }
}
```

234. Palindrome Linked List

not a good solution

```
class Solution {
    public boolean isPalindrome(ListNode head) {

    ListNode temp = head ;
    int size = 0 ;
    while(temp!=null){
        size++ ;
        temp=temp.next;
    }
      int[] arr = new int[size] ;
      temp =head ;
       int i =0 ;
      while(temp!=null && i<size){
        arr[i]=temp.val;
        temp=temp.next;
        i++ ;
      }

      int j = 0;
      int k =size-1;
      while(j<k){
          if(arr[j]==arr[k]){
              j++ ;
              k--;
          }

          else{
              return false ;
          }
      }

      return true ;
    }
}
```

2181. Merge Nodes in Between Zeros



```
class Solution {
    public ListNode mergeNodes(ListNode head) {
        ListNode node = head.next;
        ListNode nn = new ListNode();
        head = nn;

        int sum = 0;
        while(node != null) {
            if(node.val == 0) {
                nn.next = new ListNode(sum);
                nn = nn.next;
                sum = 0;
            } else {
                sum += node.val;
            }
            node = node.next;
        }
        return head.next;

    }
}
```

237. Delete Node in a Linked List

```
class Solution {
    public void deleteNode(ListNode node) {
     // The value of the node to be deleted is replaced by the value of the next node
      node.val= node.next.val;
    // since the value of next node is copied, the next node has to be removed
        node.next=node.next.next  ;
    }
}
```

234. Palindrome Linked List

```
class Solution {
    public boolean isPalindrome(ListNode head) {
       ListNode slow = head ;
       ListNode fast =head ;

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

      slow = reverseList(slow) ; // Points to the middle of the LL being reversed
       fast =head ;  // fast becomes teh head
       while(slow!=null){ // slow will proceed towards the end and fast will proceed towards middle
           if(fast.val!=slow.val){ //
               return false;
           }
           fast= fast.next ;
           slow=slow.next ;
       }
       return true ;
    }

       public ListNode reverseList(ListNode head) {
        ListNode prev = null;
        ListNode current = head;

        while(current != null) {
            ListNode next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }
       return prev;
       }
}
```

2326. Spiral Matrix IV

```
class Solution {
    public int[][] spiralMatrix(int m, int n, ListNode head) {
        int[][] arr = new int[m][n];
        for(int[] row: arr)
          Arrays.fill(row,-1); // fill all the values of the matrix by 1
        int top = 0, left = 0, right = n-1, bottom = m-1;
        while(head != null){
            for(int i=left; i<=right && head != null; i++){
                arr[top][i] = head.val;
                head = head.next;
            }
            top++;
            for(int i=top; i<=bottom && head != null; i++){
                arr[i][right] = head.val;
                head = head.next;
            }
            right--;
            for(int i=right; i>=left && head != null; i--){
                arr[bottom][i] = head.val;
                head = head.next;
            }
            bottom--;
            for(int i=bottom; i>=top && head != null; i--){
                arr[i][left] = head.val;
                head = head.next;
            }
            left++;
        }
        return arr;
```

2130. Maximum Twin Sum of a Linked List

```
class Solution {
    public int pairSum(ListNode head) {
       ListNode slow = head ;
       ListNode fast =head  ;
         int maxSum =0 ;
       while(fast!=null && fast.next!=null){
           slow=slow.next ;
           fast=fast.next.next ;
       }
// slow pointer will be at the middle
       slow= reverse(slow) ;
       fast =head ;
       while(slow!=null){
          int sum = slow.val+fast.val ;
          maxSum= Math.max(sum,maxSum) ;
          slow=slow.next;
          fast=fast.next ;
       }

     return maxSum ;
    }

//reversing the list from slow till end
    public ListNode  reverse(ListNode head){
     ListNode prev =null ;
     ListNode current = head ;
     while(current!=null){
         ListNode temp = current.next ;
         current.next  =prev;
         prev = current ;
         current =temp ;
     }
     return prev;
    }
}
```

19. Remove Nth Node From End of List

```
  //    // ex =class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        ListNode temp = head;
        int size =0 ;
        while(temp!=null){
          size++ ;
          temp=temp.next ;
        }
        temp=head ;
    if(size==1 && n==1){ //  ex- LL=[1] , n=1
        return null;
    }

        int remove = size-n ; // index from the start

        if(remove==0){   // ex - LL=[1,2] , n=2
            head=head.next ;
            return head ;
        }

        for( int i =0 ;i<=remove ;i++){
//link the node before the node to be removed to the next node
            if(i==(remove-1)){
                temp.next=temp.next.next ;
            }
            else{
                temp=temp.next ;
            }
        }

        return head;
    }
}
```

2. Add Two Numbers

Add Two Numbers Given as LinkedLists | Amazon | Microsoft | Facebook | Qualcomm

Check our Website: https://www.takeuforward.org/

In case you are thinking to buy courses, please check below:

▶ https://www.youtube.com/watch?v=LBVsXSMOIk4

```
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(0); // creating an dummy list
        ListNode curr = dummy; // intialising an pointer
        int carry = 0; // intialising our carry with 0 intiall
        // while loop will run, until l1 OR l2 not reaches null OR if they both reaches null. But our carry has some
value in it.
    // We will add that as well into our list
        while(l1 != null || l2 != null || carry == 1){
            int sum = 0; // intialising our sum
            if(l1 != null){ // adding l1 to our sum & moving l1
                sum += l1.val;
                l1 = l1.next;
            }
            if(l2 != null){ // adding l2 to our sum & moving l2
                sum += l2.val;
                l2 = l2.next;
            }
            sum += carry; // if we have carry then add it into our sum
            carry = sum/10; // if we get carry, then divide it by 10 to get the carry
            ListNode node = new ListNode(sum % 10); // the value we'll get by moduloing it, will become as new node
 so. add it to our list
            curr.next = node; // curr will point to that new node if we get
            curr = curr.next; // update the current every time
        }
        return dummy.next; // return dummy.next bcz, we don't want the value we have consider in it intially!!
    }
}
```

141. Linked List Cycle

```
public class Solution {
    public boolean hasCycle(ListNode head) {
      ListNode slow =head ;
      ListNode fast = head ;

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

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

142. Linked List Cycle II

```
public class Solution {
    public ListNode detectCycle(ListNode head) {

         ListNode fast = head, slow = head;
        while (fast != null && fast.next != null) {
            fast = fast.next.next;
            slow = slow.next;
            if (fast == slow) {
                break;
            }
        }
        if (fast == null || fast.next == null) {
            return null;
        }
        fast = head;

// The head of the cycle LL will always be equidistant as the fast/slow pointers from the Cycle Node
        while (fast != slow) {
            fast = fast.next;
            slow = slow.next;
        }
        return fast;
    }

}
```

Extended solution of 141 for 142 -

```
public class Solution {
    public ListNode detectCycle(ListNode head) {
     boolean b = false ;
      ListNode slow =head ;
      ListNode fast = head ;

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

          if(fast==slow){
             b=true  ;
             break ;
          }
      }
      fast =head;
      if(b){
          while(fast!=slow){
              fast=fast.next ;
              slow=slow.next;
          }

          return fast ;
      }

      return null ;


    }
}
```