// **Day 5** Addtion of Two Linked List

```
import java.util.*;
class Node{
        int data;
        Node next;
   Node(){}
        Node(int val)
        {
                data=val;
           next=null;
        }
}
class addLinkedList{
        Node head;
        void addNode(Node newnode)
        {
                if(head==null)
                {
                        head=newnode;
                }
                else{
                Node last = head;
                while(last.next!=null)
                {
                        last=last.next;
                }
                last.next = newnode;
```

```
}
}
void prinlist(Node h){
	while(h!=null){
		System.out.print(h.data+"->");
		h=h.next;
	}
	System.out.print("NULL");
	System.out.println();
}
static void prinlist1(Node h){
	while(h!=null){
		System.out.print(h.data+"->");
		h=h.next;
	}
	System.out.print("NULL");
	System.out.println();
}
static Node addLists(Node l1, Node l2)
{
	Node dummy = new Node();
	Node temp=dummy;
	int carry=0;
	while(l1!=null || l2!=null || carry==1)
	{
		int sum=0;
		if(l1!=null)
		{
			sum+=l1.data;
```

```
                l1=l1.next;
            }
            if(l2!=null)
            {
                sum+=l2.data;
                l2=l2.next;
            }
            sum+=carry;
            carry=sum/10;
            Node re = new Node(sum%10);
            temp.next = re;
            temp=temp.next;
        }

        return dummy.next;
    }

public static void main(String[] args) {

    addLinkedList list1 = new addLinkedList();
    addLinkedList list2 = new addLinkedList();
    // First LinkedList
    list1.addNode(new Node(5));
    list1.addNode(new Node(6));
    list1.addNode(new Node(7));
    list1.addNode(new Node(10));
    list1.prinlist(list1.head);
    // Second LinkedList
    list2.addNode(new Node(5));
```

```
        list2.addNode(new Node(6));
        list2.addNode(new Node(7));

        list2.prinlist(list2.head);
        System.out.println("The addition is :");
        Node res = addLists(list1.head,list2.head);
        prinlist1(res);
    }
}
```

```
// Day 5  Linked List Implementation
import java.io.*;
import java.util.*;
class linkedListExamples{
        Node head;
            class Node{
                int data;
                Node next;

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

// Add element begining of the linkedList
```

```
public void push(int new_data){
        Node  new_node = new Node(new_data);
        new_node.next= head;
        head=new_node;
}

public void insertAfter(Node prev, int data){
        if(prev == null){
                System.out.println("Insertion is Not possible");
                return;
        }

        Node new_node = new  Node(data);
        new_node.next=prev.next;
        prev.next=new_node;

}
// Appending Node to the end of the LinkedList
public void append(int data){
        Node new_node = new Node(data);
        if(head== null)
        {
                head = new_node;
                return;
        }
        Node last=head;
        while(last.next!=null){
                last=last.next;
        }
```

```
        last.next=new_node;
}

public void createCycle(Node head , int pos)
{
        Node startNode=null;
        int cnt=1;
        while(head!=null && head.next!=null){

                if(cnt== pos){
                        startNode = head;
                }
                head=head.next;
                cnt++;
        }
        head.next= startNode;
}

public boolean detectCycle(Node head){
        Node slow=head, fast=head;

        while(fast!=null && fast.next!=null){
                slow=slow.next;
                fast=fast.next.next;
                if(slow== fast){
                        return true;
                }
        }
        return false;
```

```
}

public void removeCycle(Node head){
        Node slow=head, fast=head;
        do{
                slow=slow.next;
                fast=fast.next.next;
        }while(slow!=fast);

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




public Node deleteStart(Node head){
        if(head== null || head.next == null)
        {
                return null;
        }

   Node dete=head;
  head=head.next;
//  delete(dete);
  return head;
```

```
}


public Node deleteNode(Node head, int key){
        Node temp=head;
        Node prev = null;

        if(head==null)
                return null;
        if(head.next== null){
                return head;
        }

        if(temp!=null && temp.data== key){
                head=temp.next;
                return head;
        }

        while(temp!=null && temp.data!= key){
                prev=temp;
                temp=temp.next;
        }

        prev.next = temp.next;

        return head;

}
```

```
public Node deleteEnd(Node head){

  Node temp=head;
  if (head== null || head.next == null) {
    return null;

  }

  while(temp!=null && temp.next.next!=null){
            temp=temp.next;
  }
  temp.next = null;
  return head;

}

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

 static Node revereListRecursion(Node head){
```

```
        if(head== null || head.next==null){
                return head;
        }

        Node new_node = revereListRecursion(head.next);
        head.next.next=head;
        head.next=null;
        return new_node;

}

public void printReverseList(Node h){
        while(h!=null){
                System.out.print(h.data+"->");
                h=h.next;
        }
        System.out.print("NULL");
}

public void revereList(){
        Node prev= null;
        Node currPtr=head;
        Node nextPtr;
        while(currPtr!=null){
                nextPtr = currPtr.next;
                currPtr.next = prev;
                prev=currPtr;
                currPtr=nextPtr;
```

```
        }
        while(prev!=null){
                System.out.print(prev.data+"->");
    prev=prev.next;
        }
        System.out.print("NULL");
}

public Node findMiddleOfLinkedList(){
        Node slow = head , fast = head;
        while(fast!=null && fast.next!=null){
                slow=slow.next;
                fast=fast.next.next;
        }
        return slow;
}

        public static void main(String[] args)throws IOException {
                linkedListExamples li = new linkedListExamples();
                li.push(2);
                li.push(1);
                li.insertAfter(li.head.next,3);
                li.append(4);
                li.append(5);
                li.push(0);
                li.printList();
                //li.revereList();
                /*Node middle = li.findMiddleOfLinkedList();
```

```
                System.out.println("The middle of Linked List is :" + middle.data);


        *//*
                Node pt =li.revereListRecursion(li.head);
                li.printReverseList(pt);
*/
                /*li.createCycle(li.head,3);
                //li.printList();

            boolean isCyclePresent=li.detectCycle(li.head);
            System.out.println(isCyclePresent);
            li.removeCycle(li.head);
            boolean isCyclePresent1=li.detectCycle(li.head);
            System.out.println(isCyclePresent1);*/
 // Delete from begining;

            // Node de= li.deleteStart(li.head);
            // li.printReverseList(de);
// Detlete From a perticular Key
            /*Node de = li.deleteNode(li.head,2);
            li.printReverseList(de);
*/
            // Delete from end;

             Node de = li.deleteEnd(li.head);
            li.printReverseList(de);

        }
}
```

```
// Day 5 Merging the Intervals


import java.util.*;
class Node{
                        int data;
                        Node next;
                        Node(int val){
                                data=val;
                                next=null;
                        }
            }
class mergingTwoLL
{
            Node head;
            public void appendList(Node newnode){
               if(head == null){
                        head=newnode;
               }
        else{
               Node last=head;
               while(last.next!=null){
                        last=last.next;
               }
               last.next=newnode;
            }
            }

            static Node mergeLl(Node l1,Node l2){
```

```
            if(l1==null) return l2;
            if(l2==null)  return l1;
            if(l1.data > l2.data){
                  Node temp=l1;
                  l1=l2;
                  l2=temp;
            }
            Node res=l1;
            while(l1!=null && l2!= null){

                  Node temp=null;
                  while(l1!=null && l1.data<l2.data){
                        temp=l1;
                        l1=l1.next;
                  }
                  temp.next=l2;
//swapping lists
                  Node tmp=l1;
                  l1=l2;
                  l2=tmp;

            }

            return res;
      }

       public void printList(){
            Node h=head;
            while(h!=null){
```

```
                System.out.print(h.data+"->");
                h=h.next;
            }
            System.out.print("NULL");
            System.out.println();
    }
    public static void main(String[] args) {
            mergingTwoLL list1 = new mergingTwoLL();
            mergingTwoLL list2 = new mergingTwoLL();

            //Create a linked list1 10,15,20

            list1.appendList(new Node(10));
            list1.appendList(new Node(15));
            list1.appendList(new Node(20));
            list1.printList();

            // Now We create Linked List2 9,12,17,30
        list2.appendList(new Node(9));
        list2.appendList(new Node(12));
        list2.appendList(new Node(17));
        list2.appendList(new Node(30));
        list2.printList();

         // Without using extra space
         //list1.head=mergeLl(list1.head,list2.head);
         //using extra space
list1.head=new extraspace().mergeLl1(list1.head,list2.head);
         list1.printList();
```

```
        }
}

class extraspace{
// This  Approach is using extra Space..
public Node mergeLl1(Node l1,Node l2)
{

        Node dummy = new Node(0);
        Node tail= dummy;

        while(l1!=null && l2!=null){

                if(l1.data<l2.data){

                        tail.next=l1;
                        l1=l1.next;
                }
                else{
                        tail.next=l2;
                        l2=l2.next;
                }
                tail=tail.next;
        }
        return dummy.next;
}

}
```

```
// Check Whether linked is palidramatic
// Day5
import java.util.*;
class palindramaticLL{
// Creating the Structure of linked List
	Node head;
	class Node{
		int data;
		Node next;
		Node(int val){
			data=val;
			next=null;
		}
	}

	// Code for reversing the linked list

   Node reverseList(Node h){
		Node prev=null;
		Node currPtr=h;
		Node nextPtr;
		while(currPtr!=null){
			nextPtr = currPtr.next;
			currPtr.next = prev;
			prev=currPtr;
			currPtr=nextPtr;
		}
		return prev;
	}
```

```
// Appeding the element at last of Linked List..
        public void append(int n){
    Node new_node = new Node(n);
                if(head==null){
                        head=new_node;
                        return;
                }

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

        // Using Optimized Solution we solve this problem like this

        public boolean palidramatic_list(Node h)
        {
                // check for Only one or two Present in the Linked List

                if(h==null || h.next==null) return true;
                // Finding Middle of linked List
                Node slow=h;
                Node fast=h;
                while(fast.next!= null && fast.next.next!=null){
                        slow=slow.next;
                        fast=fast.next.next;
                }
```

```
                // Reversing the LL
    slow.next = reverseList(slow.next);
   // Moving the slow pointer
    slow=slow.next;
   while(slow!=null){
        if(head.data!=slow.data)
                return false;
        head=head.next;
        slow=slow.next;
   }

   return true;

        }
void printReverseList(Node h){
                while(h!=null){
                        System.out.print(h.data+"->");
                        h=h.next;
                }
                System.out.print("NULL");
                System.out.println();
        }

public Node deleteKthend(Node head,int n){
        Node start = new Node(0);
        start.next = head;
        Node slow=head,fast =head;
        for (int i=0;i<n ;++i) {
                fast=fast.next;
```

```
        }

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

    Node de = slow.next;
    slow.next = slow.next.next;
     return de;

  }
        public static void main(String[] args) {
                palindramaticLL list =new palindramaticLL();
                list.append(10);
                list.append(20);
                list.append(30);
                list.append(30);
                list.append(20);
                list.append(10);
                list.printReverseList(list.head);
                //System.out.println(list.palidramatic_list(list.head));


                // Delete a Kth Node from end of linked List
                 Node kth = list.deleteKthend(list.head,4);
                 System.out.println("The delete node is :" + kth.data+"\n after delection the List Will
be");
```

```
                list.printReverseList(list.head);
        }
}
```