LINKEDLIST

Adding First and Last in linkedlist:

```java
public class addfirstinLL {// class creation
    public static class Node{
        int data;
        Node next;
    public Node(int data){
        this.data=data;
        this.next=null;
    }
    }
  public static Node head;
  public static Node tail;
  // addfirst function
  public void addfirst(int data){

    // step1-> create  new node
    Node newNode=new Node(data);
    if(head==null){
        head= tail=newNode;
        return;
      }
    // step2 newNodenext= head
    newNode.next=head;
    // step 3 head = new node
    head=newNode;
    // addlast function

  }
  public void addlast(int data){
    // step1 create new node
    Node newNode= new Node(data);
    if(head==null){
        head= tail= newNode;
        return;
    }
    // step 2 newnode.next=tail
    tail.next=newNode;
    // tail= new node
    tail=newNode;
}
// print linklist using function
public void print(){
    Node temp=head;
    if(head==null){
        System.out.println("null");
        return;
    }
        while(temp!=null){
            System.out.print(temp.data+"->");
        temp=temp.next;
    }
    System.out.println("null");

}

  public static void main(String[] args) {
    addfirstinLL ll= new addfirstinLL();
    ll.addfirst(2);

    ll.addfirst(1);

    ll.addlast(3);

    ll.addlast(4);
    ll.print();

  }
}
Output:
1->2->3->4->null
```

Adding in the middle:

```
public class addinmiddeleLL {
    //create a claas node
    public static class Node{
        int data;
        Node next;
        // calling values using constructer
        public Node( int data){
         this.data=data;
         this.next=null;
        }
    }
```

```
     // create head and tail
    public static Node head;
    public static Node tail;

     // methods
        //method 1 to add data in ll
    public void addfirst(int data){
        // create a new node
        Node newnode=new Node(data);
        //if  ll  is empty
        if(head==null){
            head=tail=newnode;
            return;
        }
        newnode.next=head;
        head=newnode;
    }
    // method 2 to add last in ll
    public void addlast(int data){
        // create new node
        Node newnode=new Node(data);
        // if ll is empty
        if(head==null){
            head=tail=newnode;
            return;
        }
        tail.next=newnode;
        tail=newnode;

    }
    // method 3 to insert in the middle
    public void addmiddle( int idx ,int data){
    Node newnode=new Node(data);
    // base case
    if(idx==0){
        addfirst(data);
    }
    Node temp=head;
    int i=0;
    while(i<idx-1){
        temp=temp.next;
        i++;
    }
    // now we get the previous element idx-1==prev
    newnode.next=temp.next;
    temp.next=newnode;

    }

    //method 4 to print the Linked list
    public void print(){
        Node temp=head;
        if(head==null){
            System.out.println("ll is empty");
            return;
        }
        while(temp!=null){
        System.out.print(temp.data +"->");
        temp= temp.next;
      }
      System.out.println("null");
    }

    public static void main(String[] args) {
        addinmiddeleLL LL= new addinmiddeleLL();
         LL.addfirst(2);
         LL.addfirst(1);
         LL.addlast(3);
         LL.addlast(4);
         LL.addmiddle(2, 10);
         LL.print();

    }
}
Output:
1->2->10->3->4->null
```

Size of a linklist:

```
public class addinmiddeleLL {
    //create a claas node
    public static class Node{
        int data;
        Node next;
        // calling values using constructer
        public Node( int data){
         this.data=data;
         this.next=null;
        }
    }
     // create head and tail
    public static Node head;
    public static Node tail;
    public static int size;

     // methods
        //method 1 to add data in ll
    public void addfirst(int data){
        // create a new node
        Node newnode=new Node(data);
        size++;
        //if  ll  is empty
        if(head==null){
            head=tail=newnode;
            return;
        }
        newnode.next=head;
        head=newnode;

    }
    // method 2 to add last in ll
    public void addlast(int data){
        // create new node
        Node newnode=new Node(data);
        size++;
        // if ll is empty
        if(head==null){
            head=tail=newnode;
            return;
        }
        tail.next=newnode;
        tail=newnode;


    }
    // method 3 to insert in the middle
    public void addmiddle( int idx ,int data){
    Node newnode=new Node(data);
    // base case
    if(idx==0){
        addfirst(data);
    }
    size++;
```

```
    Node temp=head;
    int i=0;
    while(i<idx-1){
        temp=temp.next;
        i++;
    }
    // now we get the previous element idx-1==prev
    newnode.next=temp.next;
    temp.next=newnode;

    }

    //method 4 to print the Linked list
    public void print(){
        Node temp=head;
        if(head==null){
            System.out.println("ll is empty");
            return;
        }
        while(temp!=null){
        System.out.print(temp.data +"->");
        temp= temp.next;
      }
      System.out.println("null");
    }

    public static void main(String[] args) {
        addinmiddeleLL LL= new addinmiddeleLL();
         LL.addfirst(2);
         LL.addfirst(1);
         LL.addlast(3);
         LL.addlast(4);
         LL.addmiddle(2, 10);
         //LL.print();
         System.out.println(LL.size);

    }

}
Output:5
```

Remove first and last in LL

```
public class removefirtandlast {
    public static class Node{
```

```java
        int data;
        Node next;
        public Node(int data){
            this.data=data;
            this.next=null;
        }
    }
    public static Node head;
    public static Node tail;
    public static int size;

    // METHOD1-> add first
    public  void addfirst(int data){
        Node newnode=new Node(data);
        if(head==null){
            head=tail=newnode;
            return;
        }
        size++;
        newnode.next=head;
        head=newnode;
    }
    //METHOD2 to add last
    public  void addlast(int data){
        Node newnode=new Node(data);
        if(head==null){
            head=tail=newnode;
            return;
        }
        size++;
        tail.next=newnode;
        tail=newnode;

    }
    //Method3 to add in middle
    public  void addmiddle(int idx, int data){
        Node newnode=new Node(data);
        if(idx==0){
            head=tail=newnode;
            return;
        }
        size++;
        Node temp=head;
        int i=0;
        while(i<idx-1){
            temp=temp.next;
            i++;
        }
        newnode=temp.next;
        temp.next=newnode;

    }
   // Method 4 to remove first
   public  int removefirst(){
    if(size==0){
     System.out.println("empty");
     return Integer.MIN_VALUE;
    }
    if(size==1){
        int val=head.data;
      head.next=null;
      return val;
   }
    size--;
    int val=head.data;
    head=head.next;
    return val;
   }
  // method 4to remove lastelement
  public int removelast(){
    if(size==0){
        System.out.println("empty");
        return -1;
    }
   else if(size==1){
    int val=head.data;
    head=tail=null;
    size=0;
    return val;
   }
   Node prev=head;
   for(int i=0;i<size-2;i++){
    prev=prev.next;
   }
   int val=prev.next.data;//tail.data
   prev.next=null;
   tail=prev;
   size--;
   return val;
  }


    //method 5 to print the Linked list
    public void print(){
        Node temp=head;
        if(head==null){
            System.out.println("ll is empty");
            return;
        }
        while(temp!=null){
        System.out.print(temp.data +"->");
        temp= temp.next;
       }
       System.out.println("null");
    }
   public static void main(String[] args) {
    removefirtandlast ll = new removefirtandlast();
    ll.addfirst(2);
    ll.addfirst(1);
    ll.addlast(4);
    ll.addlast(5);
    ll.addmiddle(2, 3);
    ll.print();
    System.out.println(ll.size);
    ll.removefirst();
    ll.print();
    System.out.println(ll.size);
    ll.removelast();
    ll.print();
    System.out.println(ll.size);
}}
Output:
1->2->4->5->null
4
2->4->5->null
3
2->4->null
2
```

Search for a key in LL using iterative search:

```java
public class searchinLL {
    public static class Node{
        int data;
        Node next;
        public Node(int data){
            this.data=data;
            this.next=null;
        }
    }
    public static Node head;
    public static Node tail;
    public static int size;

    // METHOD1-> add first
    public  void addfirst(int data){
        Node newnode=new Node(data);
        if(head==null){
            head=tail=newnode;
            return;
        }
        size++;
        newnode.next=head;
        head=newnode;
    }
    //METHOD2 to add last
    public  void addlast(int data){
        Node newnode=new Node(data);
        if(head==null){
            head=tail=newnode;
            return;
        }
        size++;
        tail.next=newnode;
        tail=newnode;

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

    }

    // method for searching an key
    public int search(int key){
        int idx=0;
        Node temp=head;
        while(temp!=null){
            if(temp.data==key){
                return idx;
            }
            temp=temp.next;
            idx++;
        }
        return -1;

    }
    public static void main(String[] args) {
        searchinLL ll=new searchinLL();
        ll.addfirst(5);
        ll.addfirst(4);
        ll.addfirst(3);
        ll.addfirst(2);
        ll.addfirst(1);
        ll.print();
       System.out.println( ll.search(3));


    }
}
Output:
1->2->3->4->5->null
2
```

Search for a key in LL using recursion:

```
public class recursivesearchinll {

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

        // METHOD1-> add first
        public  void addfirst(int data){
            Node newnode=new Node(data);
            if(head==null){
                head=tail=newnode;
                return;
            }
            size++;
            newnode.next=head;
            head=newnode;
        }
        //METHOD2 to add last
        public  void addlast(int data){
            Node newnode=new Node(data);
            if(head==null){
                head=tail=newnode;
                return;
            }
            size++;
            tail.next=newnode;
            tail=newnode;

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

        }
```

```java
        // method for searching an key
        public int helper(Node head,int key){
            if(head==null){
                return -1;
            }
            if(head.data==key){
                return 0;
            }
            int idx=helper(head.next, key);
            if(idx==-1){
                return -1;
            }
            return idx+1;
        }
        public int recursivesearch(int key){
            return helper(head,key);
        }
        public static void main(String[] args) {
            recursivesearchinll ll=new  recursivesearchinll();
            ll.addfirst(5);
            ll.addfirst(4);
            ll.addfirst(3);
            ll.addfirst(2);
            ll.addfirst(1);
            ll.print();
           System.out.println( ll.recursivesearch(3));


        }

    }
Output:
1->2->3->4->5->null
2
```

Reverse a LL:

```java
public class ReversealL {
```

```java
public static class Node{
    int data;
    Node next;
    public Node(int data){
        this.data=data;
        this.next=null;
    }
}
public static Node head;
public static Node tail;
public static int size;

// METHOD1-> add first
public  void addfirst(int data){
    Node newnode=new Node(data);
    if(head==null){
        head=tail=newnode;
        return;
    }
    size++;
    newnode.next=head;
    head=newnode;
}
//METHOD2 to add last
public  void addlast(int data){
    Node newnode=new Node(data);
    if(head==null){
        head=tail=newnode;
        return;
    }
    size++;
    tail.next=newnode;
    tail=newnode;

}
public void print(){
    Node temp=head;
    if(head==null){
        System.out.println("null");
        return;
    }
        while(temp!=null){
            System.out.print(temp.data+"->");
        temp=temp.next;
    }
```

```
        System.out.println("null");

    }
     // method for removing a Linked list
     public void reverse(){
        Node prev=null;
        Node curr=tail=head;
        Node next;
        while(curr!=null){
            next=curr.next;
            curr.next=prev;
            prev=curr;
            curr=next;
        }
        head=prev;
     }

    public static void main(String[] args) {
        ReverseaLL ll=new  ReverseaLL();
        ll.addfirst(5);
        ll.addfirst(4);
        ll.addfirst(3);
        ll.addfirst(2);
        ll.addfirst(1);
        ll.print();
        ll.reverse();
        ll.print();
    }

}
Output:
1->2->3->4->5->null
5->4->3->2->1->null
```

Find and remove nth node from end(flip,amazon,adobe)

```
public class removefend{
    public class Node{
        int data;
        Node next;
        public Node(int data){
            this.data=data;
            this.next=null;
        }
    }
    //head and tail creation
    public static Node head;
    public static Node tail;


    public static int size;

    //methods to add first or last
    public void addfirst(int data){
        Node newnode= new Node(data);
        size++;
        if(head==null){
            head=tail=newnode;
            return;
        }
        newnode.next=head;
        head=newnode;
    }

    public void addlast(int data){
        Node newnode= new Node(data);
        size++;
        if(head==null){
            head= tail= newnode;
            return;
        }
        tail.next=newnode;
       tail=newnode;
    }

    public static void print(){
        Node temp=head;
        if(head==null){
            System.out.println("empty");
            return;
        }
        while(temp!=null){
            System.out.print(temp.data+"->");
            temp=temp.next;
        }
        System.out.println("null");
    }


public void removenthfromend(int n){
    int size=0;
    Node temp=head;
    while(temp!=null){
        temp=temp.next;
        size++;
    }
    if(n==size){
        head=head.next;
        return;
    }

    int i=1;
    int idxtofind=size-n;
    Node prev=head;
    while(i<idxtofind){
        prev=prev.next;
        i++;

    }
    prev.next=prev.next.next;
    return;
}

    //main
    public static void main(String[] args) {


        removefend ll = new removefend();


        ll.addfirst(2);
        ll.addfirst(1);
        ll.addlast(2);
        ll.addlast(1);
        ll.print();
        ll.removenthfromend(2);
        ll.print();
    //
    }

}
Output:
1->2->2->1->null
1->2->1->null
```

Check weather the LL is palindrome or not:

```
public class palindromeornot {

    public class Node{
        int data;
        Node next;

        public Node(int data){
            this.data=data;
            this.next=null;
        }
    }
    //head and tail creation
    public static Node head;
    public static Node tail;


    public static int size;

    //methods to add first or last
    public void addfirst(int data){
        Node newnode= new Node(data);
        size++;
        if(head==null){
            head=tail=newnode;
            return;
        }
        newnode.next=head;
        head=newnode;
    }

    public void addlast(int data){
        Node newnode= new Node(data);
        size++;
        if(head==null){
            head= tail= newnode;
            return;
        }
        tail.next=newnode;
       tail=newnode;
    }


    public static void print(){
        Node temp=head;
        if(head==null){
            System.out.println("empty");
            return;
        }
        while(temp!=null){
            System.out.print(temp.data+"->");
            temp=temp.next;
        }
        System.out.println("null");
    }



public Node findmid(Node head){
    Node slow=head;
    Node fast=head;
    while(fast !=null && fast.next!=null){
        slow=slow.next;
        fast= fast.next.next;

    }
    return slow;

}
public boolean findpdrome(){
    if(head==null && head.next==null){
        return false;
    }
    //1 find mid
    Node mid=findmid(head);
    //2 reverse 2nd half
    Node prev=null;
    Node curr=mid;
    Node next;
    while(curr!=null){
        next=curr.next;
        curr.next=prev;
        prev=curr;
        curr=next;
    }
    Node right=prev;// 2nd half lastpart
    Node left=head;// 1st half firstpart

    //3 check 1st half==2nd half
    while(right!=null){
        if(left.data!=right.data){
            return false;
        }
        right=right.next;
        left=left.next;
    }
    return true;
}

public boolean isdprome(){
    if(head==null && head.next==null){
        return false;
    }
    //1 find mid
    Node mid=findmid(head);

    //2 rev 2nd half
    Node prev=null;
    Node curr=mid;
    Node next;
    while(curr!=null){
    next=curr.next;
    curr.next=prev;
    prev=curr;
    curr=next;}
    Node left=head;
    Node right=prev;
    //3 left=right

    while(right!=null){
        if(left.data!=right.data){
            return false;}
            left=left.next;
            right=right.next;
    }
    return true;
}
    //main
    public static void main(String[] args) {
       palindromeornot ll=new palindromeornot();
        ll.addfirst(2);
        ll.addfirst(1);
        ll.addlast(2);
        ll.addlast(1);
        ll.print();
      System.out.println(ll.isdprome());
    }

}
Output:
1->2->2->1->null
true
```