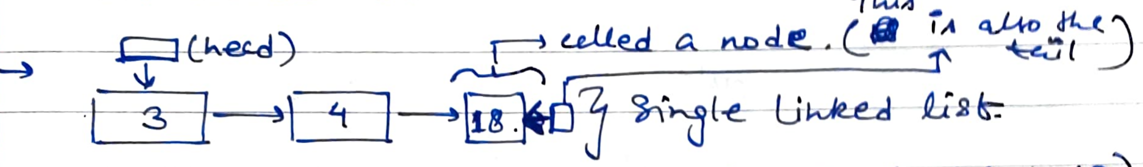
Linked List

① Why? → In general, arrays are continuous (structures) data storing, & are fixed → Limitation.

② → Linked List is not continuous, but dynamic [not fixed]. That's why they also need addresses to point where next element is.



→ 4 doesn't know about 3, just about 18 (points to 18).

→ head has only address.

→ tail doesn't have address (points to null).

③ Syntax → class Node{ } → All are private.

```
class Node{
    int val;          → data of LL Node.
    Node next;        → address of next. LL node.
}
```

* temp val is created everytime, i.e for anything to be done on linked list, temp val is created.

* tail → generally found by an item having next = null. if not given, same used for finding size().

④ * In Java direct usage (using Collections).

→ LinkedList <Integer> list = new LinkedList<>();

has many functions →

a). add(data);
b). add (index, data);
c). addFirst (data);
d). addLast (data);
e) get, getFirst, getLast(); → indexOf also.
f) clear();, clone();
g) peek() → shows, doesn't remove
h) poll() → shows & removes.

i) remove(), remove(object o), remove(index), removeFirstOccurrence();

j) set(index, element)   k) size()   l) toArray(); toString();

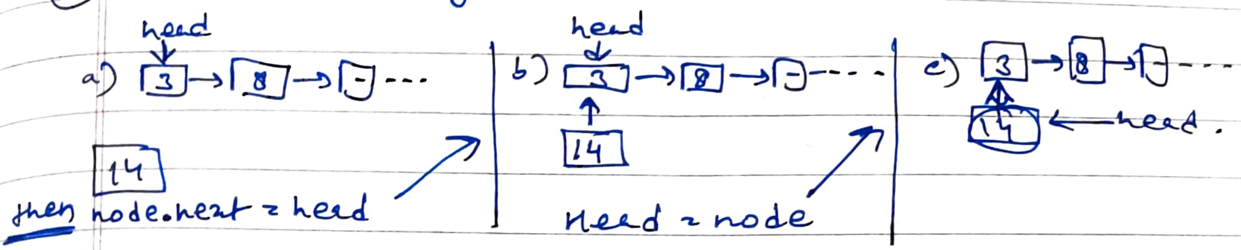
(5) Code & Internal working.

```
package com.Tyagi;

line (1) public class LL{
line (2)     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;
                 }
         }
         private Node head;
         private Node tail;

         private int size;

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

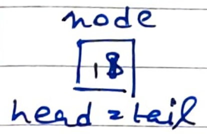
Constructor for LL.

```
package com.Tyagi

public class Main{
    public static void Main(String[] args){
        LL list = new LL();

    }
}
```

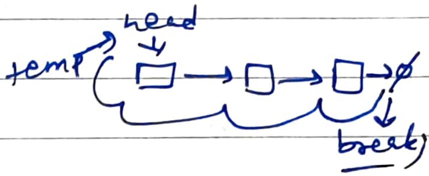
⑥ Insertion & Display.



d) Inserting the first element.



e) Display.



Code [B/n Line ① & ②].

(for inserting elems at start.)

```
public void insertFirst(int val){
        Node node = new Node(val); //adding ~~new~~ new node.
        node.next = head;   //i.e null.
        head = node;   // head equial to address of new node.
        if(tail == null){
            tail = head;
        }
        size += 1;
}
```

`if(tail == null){ tail = head; }` → tells us if it is the first ele to be added.

`size += 1;` → keeps track of size

```
public void display(){
      Node temp = head;  // makes a new temp node.
      while(temp != null){
          system.out.println(temp.value + " -> ");
          temp = temp.next;
      }
      System.out.print("END");
}
```

//ln not needed.

f) Insert at last

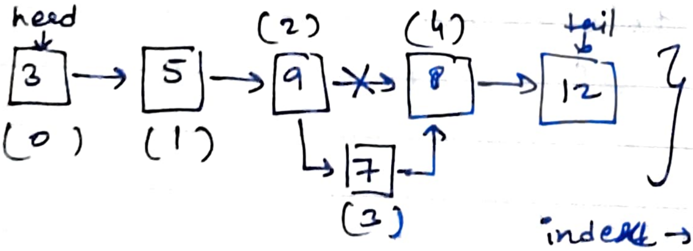
```
public void insertLast(int val) {
        if(tail == null){
            insertFirst(val);
            return;
        }

        Node node = new Node(val);
        tail.next = node;
        tail = node;
        size++;
}
```

⇒ If tail not there then use fn in display() to go to last and insert the element. → O(n).

⇒ If tail there → O(1)

---

g) Insert at a particular Index



2 connections to be made.

1 connection to be removed.

index → 3.

Code

```
public void insert(int val, int index) {
    if (index == 0){
        insertFirst(val);
        return;
    }
    if( index == size) {
        insertLast(val);
        return;
    }
    Node temp = head;
    for (int i=1; i<index; i++) {
        temp = temp.next;  // [9] will be temp finelly.
    }
                                      **
    Node node = new Node (val, temp.next);
    temp.next = node;

    size++;

}
```

** Remember $\boxed{9}$ is temp at that stage, so new node $\boxed{7}$ created will point to temp.next i.e $\boxed{8}$.

& then temp.next = node ($\boxed{9} \rightarrow \boxed{7}$)

⑦ Deletion (first, last & index).

```
public int deleteFirst() {
    int val = head.value;  // imagine as (node.next).value
        head = head.next;
        if( head == null) {
            tail = null;
        }
        size--;
        return val; // returning deleted element.
}
```

```
public int deletelast() {
```

↳ thinking here is we iterate to n-2 and then make it's address point to null.

Last element will go into garbage value.

P.T.O.

Getting value at the index given [prereq for delete last & for any value also.

```
public Node get(int index){
    Node node = head;
    for (int i=0; i<index; i++){
            node = node.next;
    }
    return node;
}
```

```
public int deletelast(){
    if(size <= 1){
        deleteFirst();
    }

    Node secondLast = get(size-2);
    int val = tail.value;
    tail = secondlast;
    tail.next = null;
    return val;
}
```

} if tail not there use a temp node.

(optional step). storing value to be deleted to return

// 2 temp values c
also be u

Deleting any element.

```
public int delete(index){
        if(index==0){ deletFirst();}
        if(index == size-1){ deletelast();}

        Node.prev = get(index-1);
        int val = prev.next.value;
        prev.next = prev.next.next;
        return val;
}
```

Bonus fn

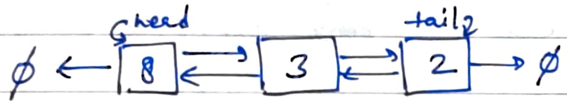
```
public Node find (int value) {
        Node node = head
        for (int i = 0; i<size; i++) {
            ~~if (node != null) {~~ if( node.value == value){
                return node;
            }
            node = node.next;
        }
        return null; // node not found.
}
```

→ while(node != null) can also be used.

→ (* write node instead of i here.)

Doubly LL & Circular LL

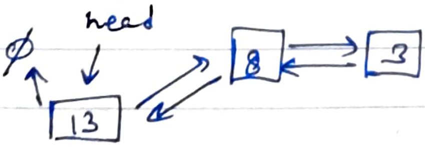
① →



Syntax →

```
class Node {
    int val;
    Node next;
    Node prev; → Extra added.
}
```

② → Insertion.



from ① to ②

Create new node.

node.next = head

node.prev = null.

Check for null pointer exception { Head.prev = node

head = node

```
if(head != null) {
    head.prev = node;
}
```

③ Insert at last & Display remains the same. Deletion remains same.

④ To reverse linked list, start from tail & node.prev, use a temp variable. till temp ≠ ∅.

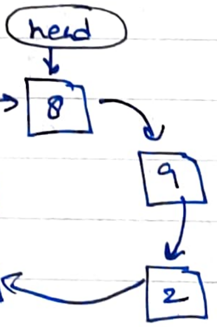
⑤ **Insertion in b/n**

- Same as SLL if index given.
- If Question is insert node after a given value. i.e Insert after 5.

Sol. create temp reach at 5. Create the new node.

```
→ node.next = temp.next;
  temp.next = node;        node.next.prev = node.
  etc.
```

Circular LL →



Only difference:

- things are not null, until list is empty.

```
→ tail.next = node       }              |  if(head == ∅){
  node.next = head       } insertion    |     head = node.
  tail = node.           }              |     tail = node.
                                        |  }
```

---

do while loop for display →

```
do {
    System.out.println(node.val)
    node = node.next;
} while (node != head);
```