notes

Linked list

* Internally in Java ... Array is not a continuous memory allocation.

* In General term Array as continuous memory allocation. or. ....

Let's you have a fixed size of array if that gets full then you cannot really add new items in that

* what is one of the ways to do it then something that Array list class?

Let's say ex

you have an array

3	4	1	8	9

that array contains five items and what happens is that you wanna insert a sixth item in array list.

3	4	1	8	9	7			

doubling it; it will copy all of these.

then it's not really cool. it is copying it of whatever then though it's so efficient O(1) avg. -- this limiting us.

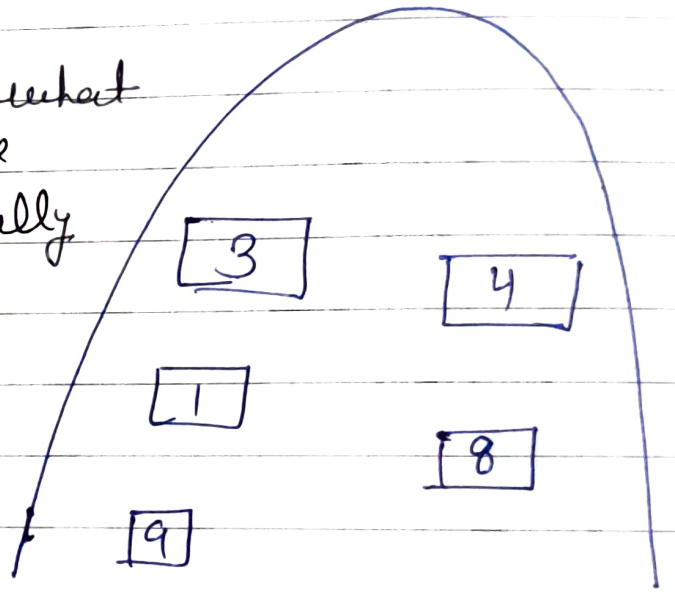
notes

So, How Linked list works.

Linked list basically says that it's not going to be like continuous memory allocation. or it will not be like fixed instead they try to break these boxes into seprate boxes.

this is what you have internally

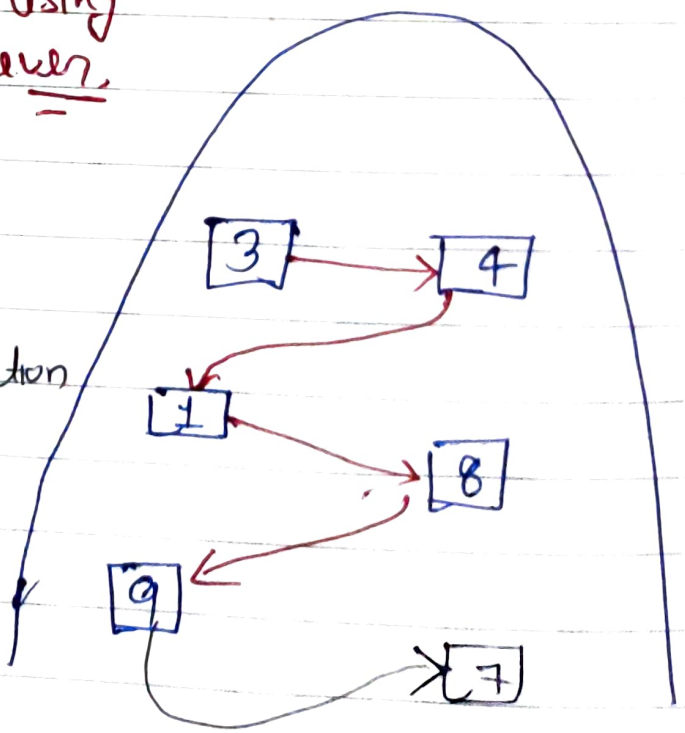
every block putting it in some random memory. and add new item just put it on some random memory.



Q how are these connected with each other.?

they are connected using these arrows & whatever.

if I wanna add a new item ... it's not really continuous memory allocation you can add new item anywhere in the memory. then you can point this over here.



this is basically what a linked list is.

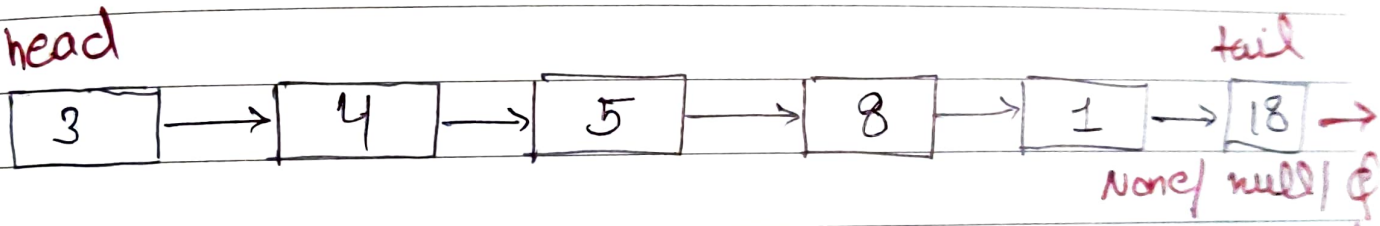
How it works internally?

* [8],

this particular box is known as a Node.

these boxes are somewhat like this in memory:

[8]

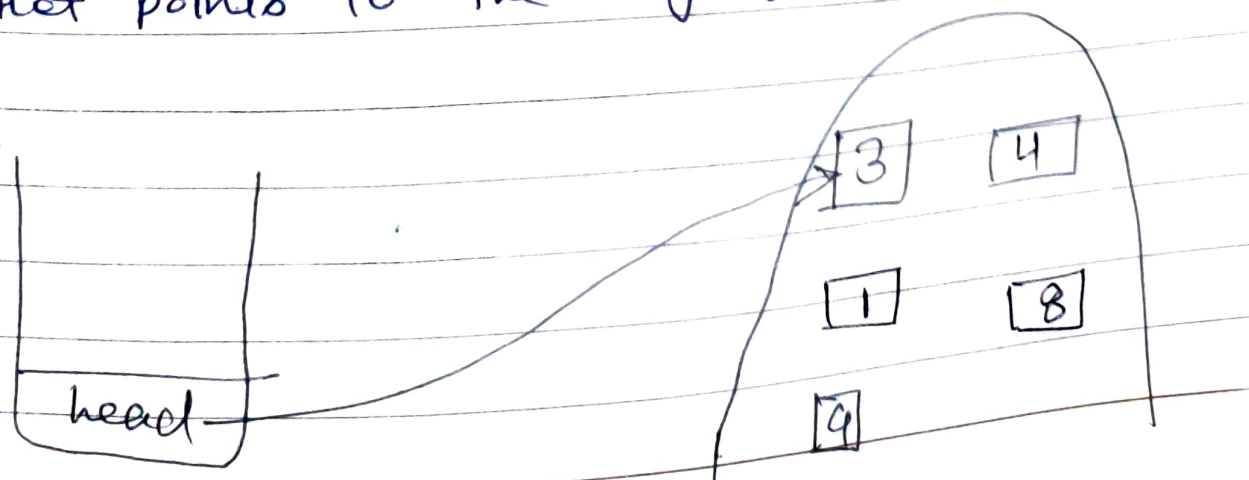


this is the normal representation of the linked list.

* In the list every single item knows about the next item. It is a singly linked list.

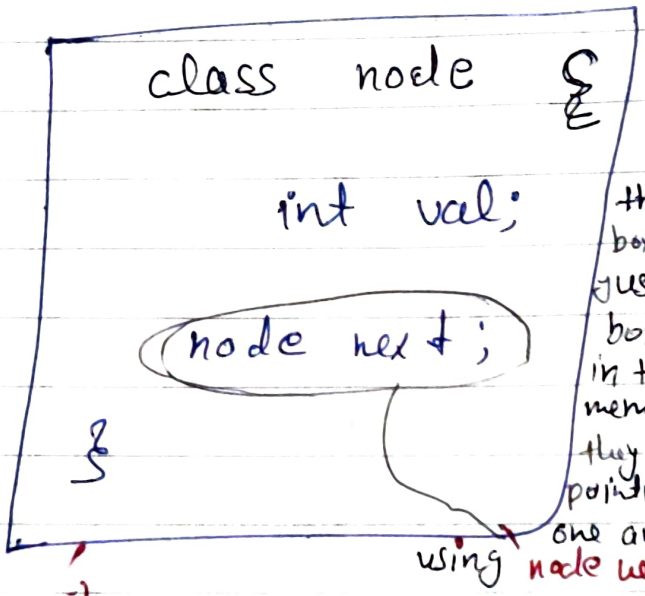
head :- head is actually just a reference variable that points to the very first node.

tail :-



tail :- tail will be pointing to the last one

* this individual box have its own type. node itself is a type.



① this is what the representation is going to be every single box.

② Every single box known as a node.

this is a structured of linked list

Q. what does all the properties this every single box have.?

① a box has its own value like an integer value.

② to whom it is pointing to whom it loves ... the next pointer.

its pointing to some other node of type node only

Ex value is 3, type node its pointing to another node ... another object of type node value is 4 ... 4 is pointing to another object of type node value is 5. So every single object is going to have this arrow that is pointing to which it is pointing to node type.

notes

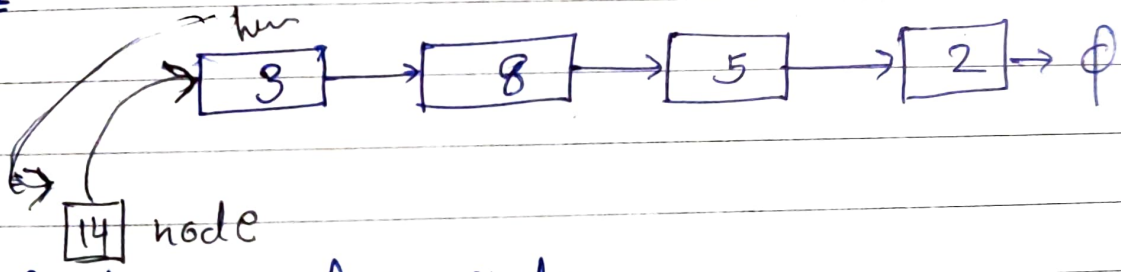
(*) Every single item knows about the next item in the list.

(*) No one idea about how many total elements in the list.

(*) they had only idea about two things
(1) who is the person that I love.
(2) what is my integer value.

insertion in a singly linked list :-

Eg.



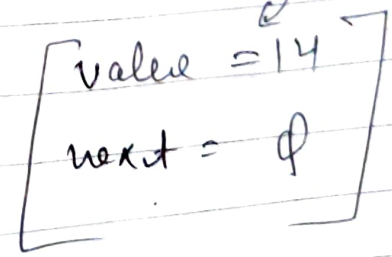
first create a Node

So head points to 3. but head should be first node

```
node.next = head;
head = node.
```

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

every node has two values:



head
18 node
tail

notes

How gonna display those things. :-

e.g:-

wrong

```
while ( head != ϕ )
    print( head.value )
```

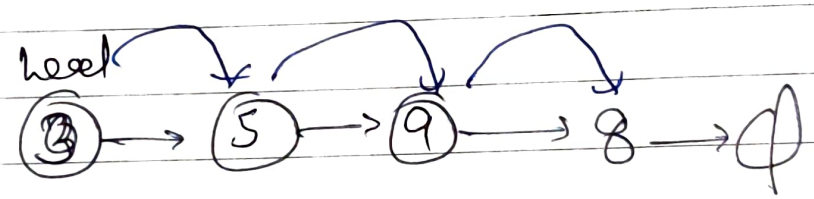
keep printing head.value...

but

there is a problem:

if you print head.value and then head updated

like



if check

head == null NO

print (3)

head is going to be forward

head move over (5) and check.

:

and after print 

and move over ϕ

because head is already null

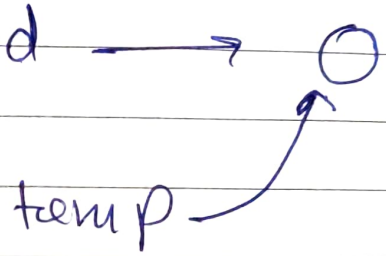
only I have to printed cancel

and it's actually ~~then~~ change the entire

notes

Structure of linked list. head always suppose to be first element. you never make head like this until or unless you changed the structure of the linked list.

instead of this..... you can create a temporary Node which is actually equal to head.

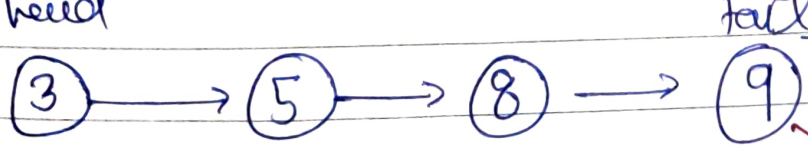


head is pointing to this Node and temp also pointing to this Node.

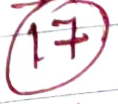
display function you will have a temp.

(*) #insert a node in the last.

E.g



if (tail == null) {insertfirst(val);}



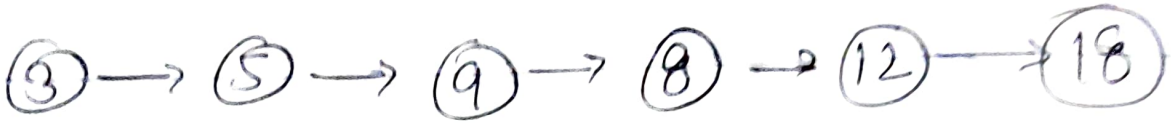
~~tail~~ =

tail.next = node

tail = node size++;

- If insert a node ~~in the~~ at a particular position -

you have a linked.

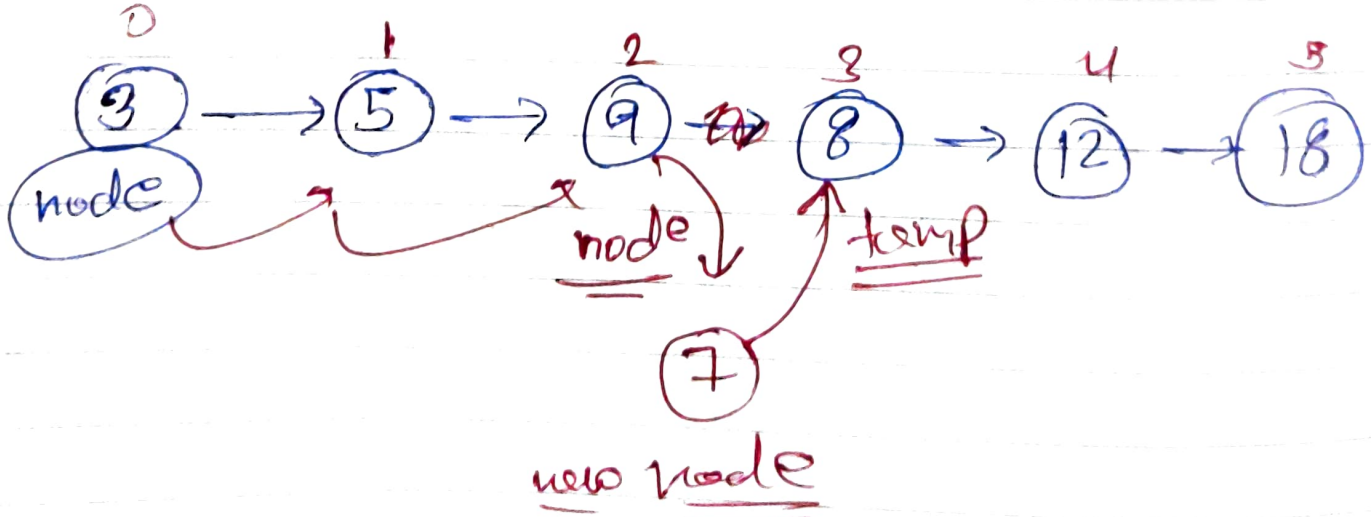


how do you do that:-
first things first given to 0
zero, you just call insertFirst()
method that we just created if
the index equal to the size then you
just call the insertLast() method
we have created

what if:- if let's say index (3)

index = 3

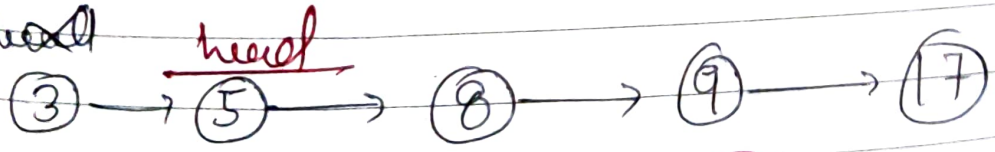
you gonna start from here



notes

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

deletion from first.



head will come over 5

```
head = head.next
```

if we have only one item

head, tail (both)

(3) → ∅

head.next = head

So

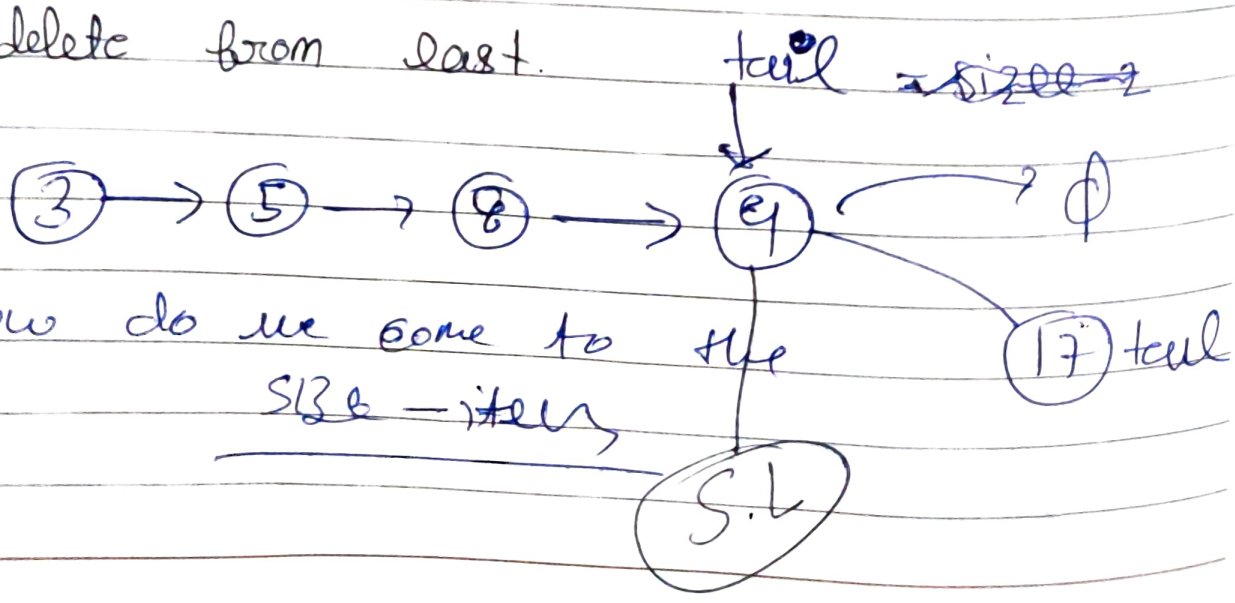
what about tail.

tail also be null

// delete first

```
public int deleteFirst() {
    int val = head.value;
    head = head.next;
    if( head == null) {
        tail = null;
    }
    size--;
    return val;
}
```

// delete from last.



How do we come to the size - item

So you wanna get the value of this node you can create a get function.

∴ Give me the reference of that node.

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

it will actually return reference pointer to that node

delete last element -

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

notes

delete last element:-

```
public int deleteLast() {
    if(size <= 1) {
        return deleteFirst();
    }
    Node secondLast = get(size-2);
    int val = tail.value;
    tail = secondLast;
    tail.next = null;
    return val;
}
```

// remove from a particular index

you need to remove 8

first you have to go one step behind.

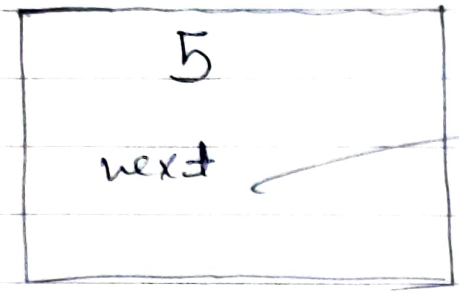
code :-

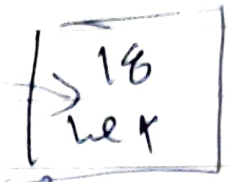
```
public int delete (int index) {
    if( index == 0) {
        return deleteFirst ();
    }
    if ( index == size -1) {
        return deleteFirst ();
    }
    Node prev = get ( index -1);
    int val = prev.next.value;
    prev.next = prev.next.next;

    return val;
}
```

Si Similarly you can find the value ? I will give you particular value give me the node of.

```
public Node find (int value) {
    Node node = head;
    while ( node != null) {
        if( node.value == value) {
            return node;
        }
        node = node.next;
    }
    return null;
}
```

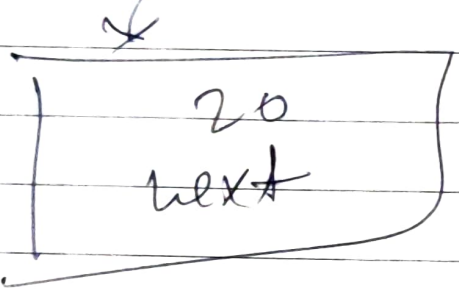
O(n)





ref = obj
does it not mean

ref variable pointing to an object in heap.



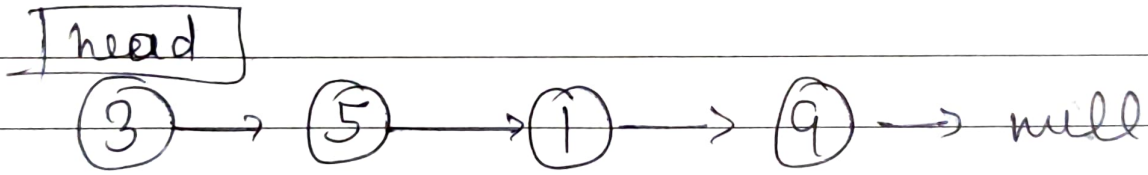
this is a reference variable pointing to an object so internally it will also be something like this
this is a reference variable in this object it will pointing to some other object in heap ... this object ~~itself~~ itself will have its identity (98) it will also have a next pointer next pointer pointing to some other object . . . . you can see these are not like index values that you can directly jump on. it will may random places in the memory.

You can't directly jump on to it.
So you wanna go over here from 5 to 18 I can say
this node = name.next

So, this is basic understanding of what this thing represent next.

This is a reference variable of type Node because it's pointing to something that is type node. that's why it's a type node here.

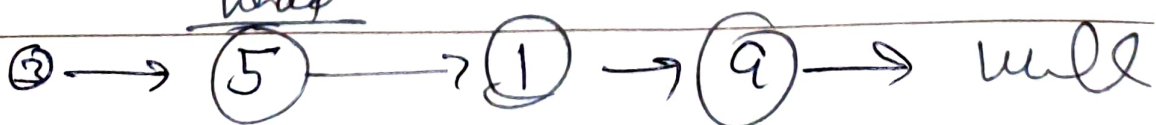
What is head:-



node = head.

Scope will be in function only it will not change structure of linked list.

if you were just saying print head
so head = head.next .next itself is a node type head will now



notes

he will move over here

Now entire structure of LL is changed
So, head will be at null.

head should always be over here

(X) remember

node.something = something
this is actually saying making a change.

the only thing we did was

node = node.next

this is not changing the pointer.
(head pointer)

this is reassigning.

you can think as a one-sided love