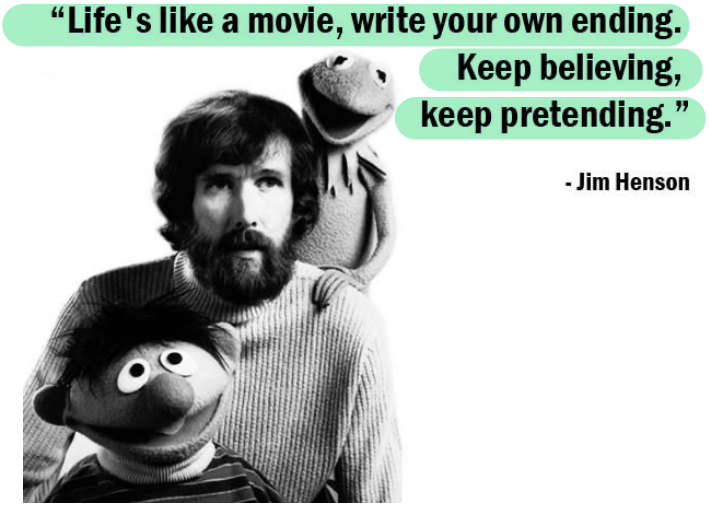
LinkedList



Good Evening

Topics for Today

- → Classes & Object
- → Multiple obj reference
- → Constructor
- → Array vs Linkedlist
- → Size of Linkedlist
- → Insertion in Linkedlist
- → Deletion in Linkedlist

→ Class → Blueprint (design)

→ Object → Real instance / of class creation

→ Class
- Attributes : Holds information
- Functionality : Action

```
class Car {
    Color
    Seater
    Body type
    Brand

    AC()
    Speed()
    Music()
}
```

Rohit's car

Color : White
Seater : 4
Body type : SUV
Brand : Audi

AC()
Speed()
Music()

Abel's car

Color : Yellow
Seater : 5
Body type : SUV
Brand : Porsche

AC()
Speed()
Music()

Obs 1 → Same class can be used for multiple objects

Obs 2 → Attributes are going to be different but the functionalities will remain same

```
class Student {
    String name;
    int rollno;
    int m1, m2, m3;

    int totalmarks() {
        return m1 + m2 + m3
    }
    void printName() {
        SOP(name);
    }
    int maxmarks() {
        return max(m1, m2, m3);
    }
}
```

Syntax for object creation

```
Student S1 = new Student();
```

$S_1 = 4k$

name : Nayan
roll no: 42
m_1 : 80
m_2 : 46
m_3 : 0

4k

To access the attributes of an object we are going to use **. operator**

$S_1.name = "Nayan"$

$S_1.rollno = 42$

$S_1.m_1 = 23$

$S_1.m_2 = 46$

$S_1.m_1 = 80$

print (S_1.totalmarks()); ⟶ 80 + 46 + 0

* **Student S_2 ;** → valid statement but if you want to use it as a reference variable, then you have to initialise it.

Student S_2 = null;

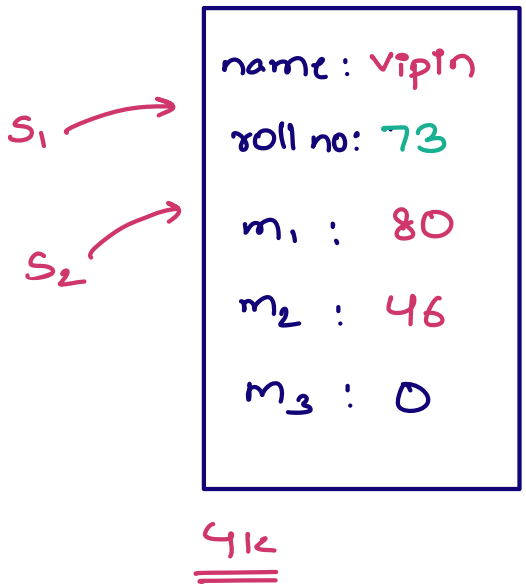
Multiple object reference

Student S_2 = S_1 (4K) → Created a shallow copy & now both S_1 & S_2 are pointing at same address

S_2 = 4K

→ S_2.name = "Vipin";

→ S_2.rollno = 73



Shallow copy → Only the object address will get copied in different reference

Deep copy

01. Student S_3 = new Student();

S_3.name = S_1.name

S_3.rollno = S_1.rollno

S_3.m_1 = S_1.m_1

S_3.m_2 = S_1.m_2

Constructor

```
class Pair {
    int x;
    int y;
    .
}
```

```
Pair P1 = new Pair();
P1.x = 10;
P1.y = 20
```

To modify attributes

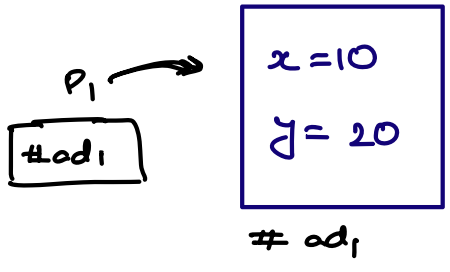
x = 10
y = 20

Constructor → Used to initialise the attributes in the class at the time of object creation

- Constructor name is going to be same as the class name.
- It is similar to function with no return type

```
class Pair {
    int x;
    int y;
    Pair(int a, int b){
        x = a;
        y = b;
    }
}
```

```
Pair P1 = new Pair(10,20);
```



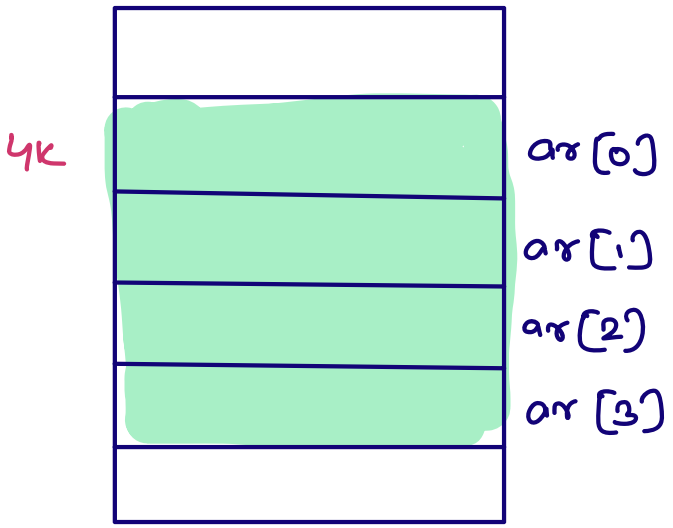
```
class Pair {
    int x;
    int y;

    Pair(int x, int y){
        this.x = x
        this.y = y
    }
}
```

this → reference of current class obj

10:25 → 10:35 pm

Arrays Vs Linked List




```
int [] ar = new int[4]
```

↓

Continous memory

TC of random accessing = O(1)

arr[x] = { base address + x * size of 1 variable }

arr[0] = 4K + 0 * 4 byts

= 4K + 0

= 4K

arr[2] = 4K + 2 integer

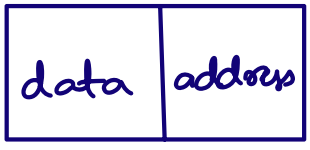
= 4K + 2 * 4 Byts

= 4000 + 8 byts = 4008 byts

Disadvantage

01. Fixed size
02. To create an array, we need continous memory.
03. Insertion & deletion

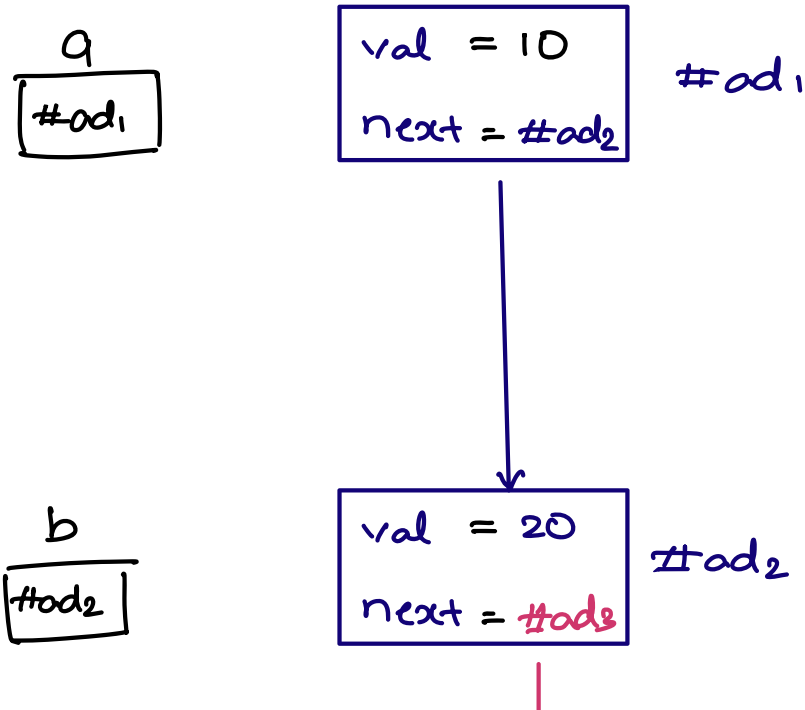
LinkedList → Linear datastructure




```
class Node{
    int val
    Node next;

    Node(val){
        this.val= val;
        .
    }
}
```

Node a = new Node(10);

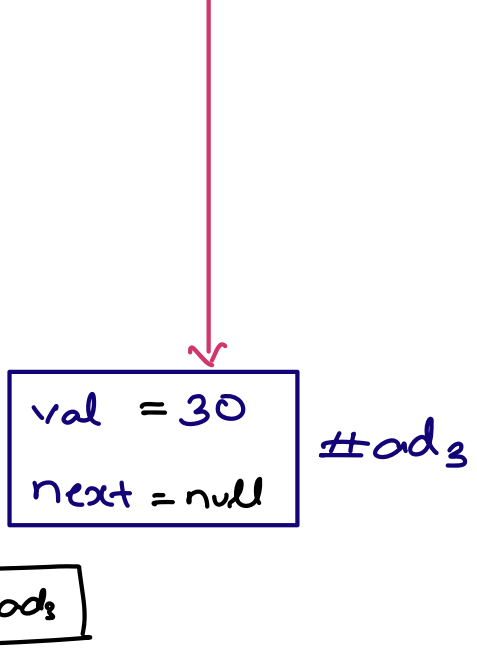


Node b = new Node (20):

a.next = b

print (b.val) = 20

print (a.next.val) = 20



Node c = new Node (30):

b.next = c:

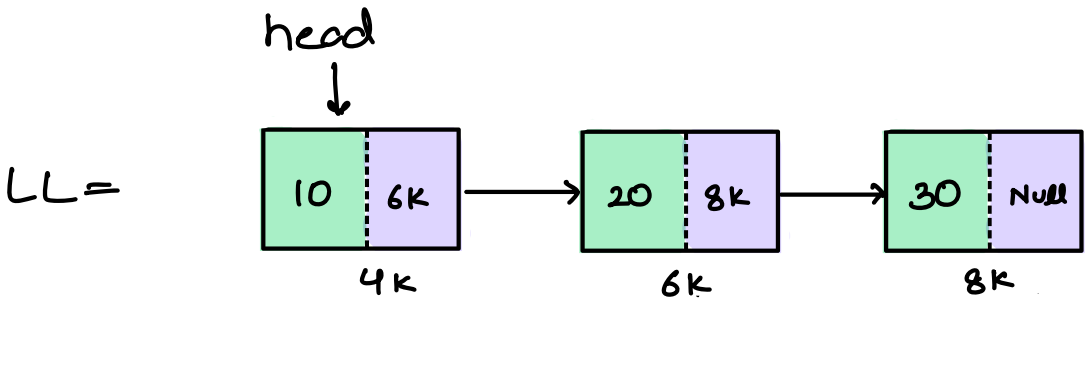
print (c.val) = 30

print (b.next.val) = 30

print (a.next.next.val); = 30

#ad2 . next

#ad3 .val :



Node head = 4K;

Q1. Find the size of LinkedList

size = 0 ✗ 2

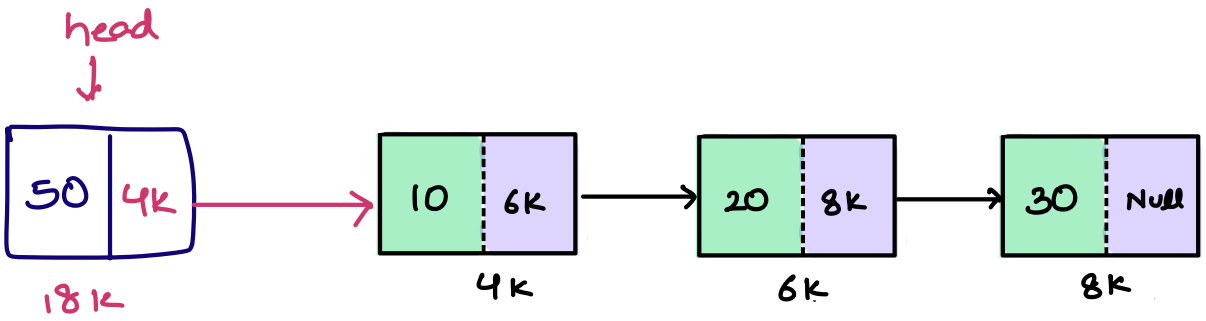
```
int size(Node head)
    Node temp = head;
     size=0
    while (temp.!=null){
        size = size+1
        temp= temp.next
    }
    return size:
}
```

* Insertion

01. Insertion at the beginning

// Create a node

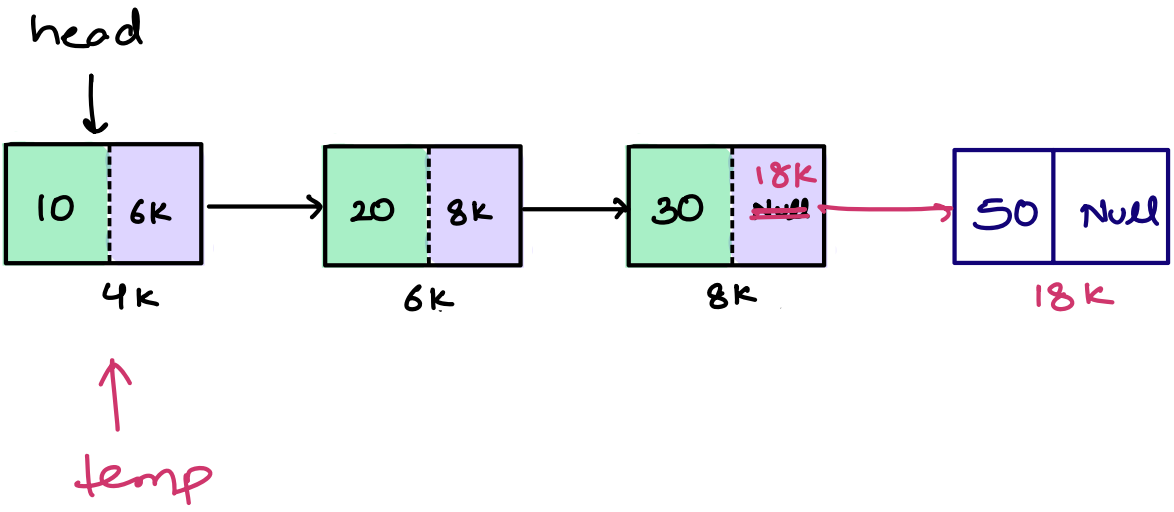
// Make the certain links




```
Node insertAtStart (Node head, int val)        <-- 50
    Node nn = new Node(val);
    nn.next = head
    head = nn;
```

3

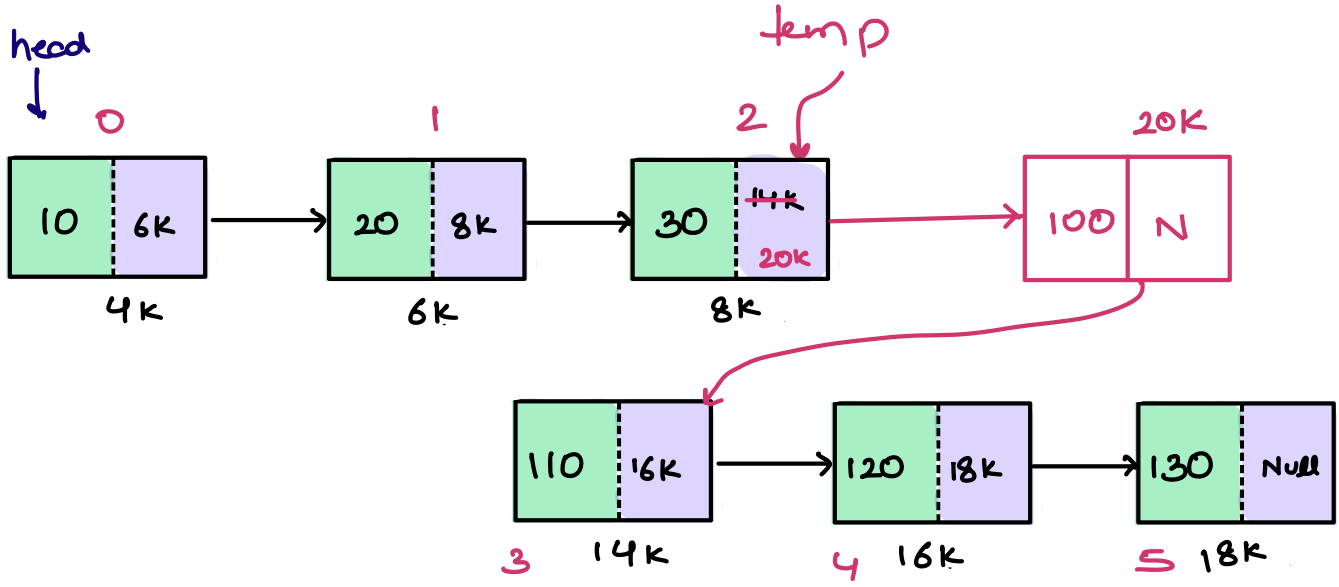
02 Insert at the Last with val 50



```
Node insertatlast ( Node head, int val)
    Node nn = new Node (val);
    if (head == null) { head = nn);
    else { Node temp = head
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = nn
    return head;
}
```

03. Insert at k^{th} index

Q = Insert a node with value as 100 at 3rd index



```
Node insertatKthindex (Node head, int val, int K)
{
    Node nn = new Node(val);
    if (K==0) return insertatstart(head, val);
    else {
        Node temp = head;
        for (i=1; i<K; i++){
            temp = temp.next;
        }
        nn.next = temp.next
        temp.next = nn;
    }
    return head;
}
```

K-1 iteration

Deletion

Delete at first = head = head.next

Delete at last = Move to second last

secondlast.next = null

Delete at k^{th} index = Move to $(k-1)^{th}$ node

$(k-1)^{th}$ node.next = $(k-1)^{th}$ node.next.next