LinkedList

⇒ The LinkedList class is a Collection which can contain many objects of the same type, just like a ArrayList.

Implimentation

⇒ A linkedlist consists of the data element known as node. And each node consists of two fields:- one field has has data and second field has address of the next node.

* Use of linked list *

Linked list are often used because of their efficient insertion and delition

Time Complexity

ArrayList		LinkedList
O(n)	← Insert →	O(1)
O(1)	← Search →	O(n)

* Types of LinkedList

1. Singly linked list
2. Doubly linked list
3. Circular linked list

Properties of linked list

* Variable Size => not a fixed size

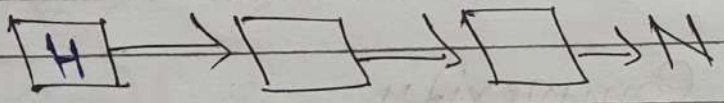
non-contiguous memory =>

Insert in O(1) =>

Search in O(n) =>

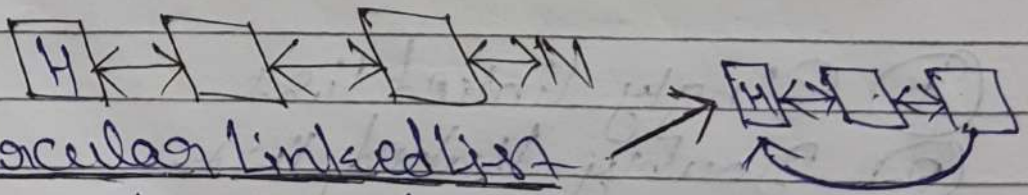
1. Singular linked list

=> A linked list traversed in only one direction from head to the last node (tail)



2. Doubly linked list

=> Doubly linked list consists of a set of sequentially linked records called nodes. Each nodes contains three fields: two link fields and one data field



3. Circular linked list

=> A circular linked list is a variation of a linked list in which the last node points to the first node, completing a full circle of nodes and it doesn't have a null element in end.

Add in linked list

Add on first position

```
class LL {
    class Node {
        String data;
        Node next;

        Node(String data) {
            this.data = data;
            this.next = null;
        }
    }

    // Add on first position
    Public void addFirst(String data) {
        Node newNode = new Node(data);
        if(head == null) {
            head = newNode;
            return;
        }
        newNode.Next = head;
        head = newNode;
    }

    // main method
    public static void main(String args[]) {
        LL list = new LL();
        list.addFirst("a");
        list.addFirst("is");
        list.printList();
    }
}
```

add on last position

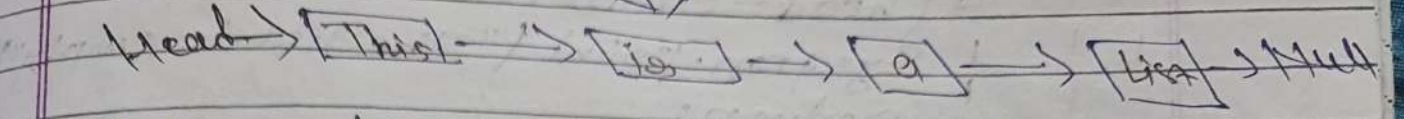
```
// Add on last position

Public void addlast(String data){
    Node newNode = new Node(data);
    If(head == null){
        head = newNode;
        return;
    }

    Node CurrNode = head;
    While(CurrNode.next != null){
        CurrNode = CurrNode.next;
    }

    CurrNode.next = newNode;
}
```

```
// Print

Public void printList(){
    If (head == null){
        System.out.println("List is empty");
        return;
    }

    Node CurrNode = head;
    While(CurrNode.next != null){
        System.out(CurrNode.data+"->");
        CurrNode = CurrNode.next;
    }
    System.out.println("null");
}
```

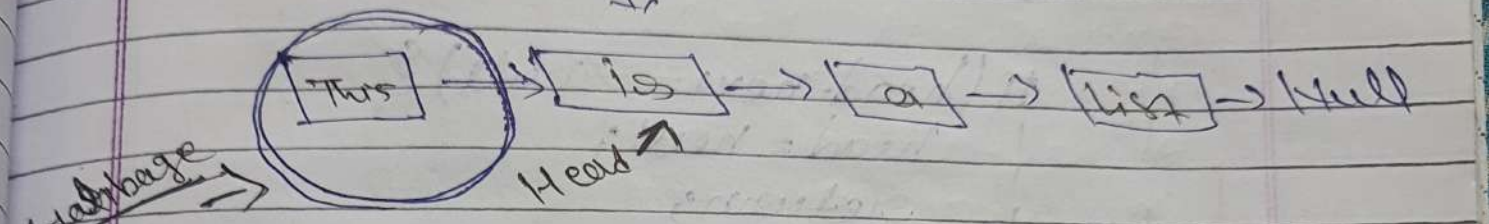
in main method Call ⇒ List.printList();

* Logic for delete first

lets our linked list is



⇒ make head to first.next

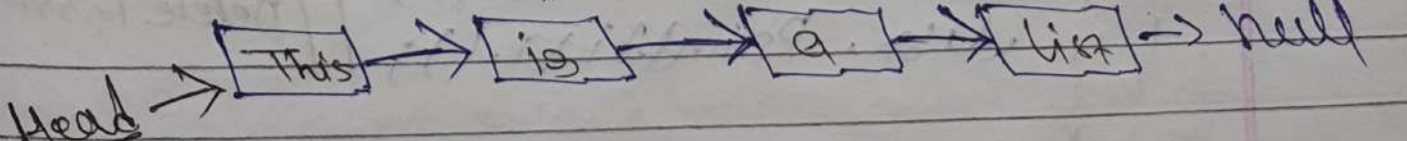


// Delete first

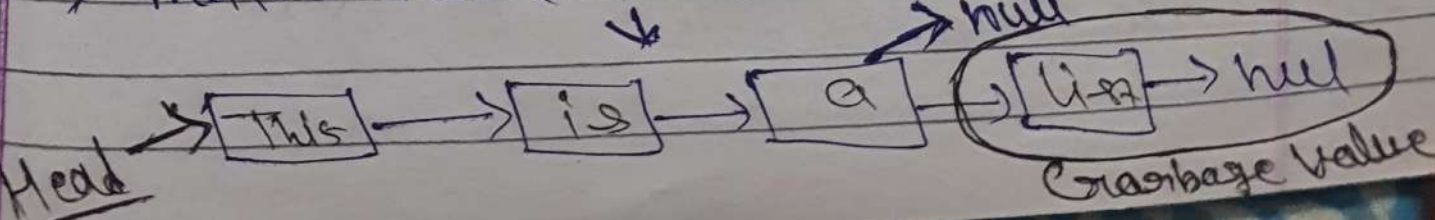
```
public void deleteFirst(){
    if(head==null){
        System.out.println("the list is Empty")
        return
    }
    head = head.next;
}
```

* Logic for delete last

lets our Linked list is



⇒ make null to the second last



// Delete last

```
Public void deleteLast() {
    if(head == null) {
        System.out.print("The list is empty");
        return;
    }

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

    Node secondLast = head;
    Node lastNode = head.next;
    while(lastNode.next != null) {
        lastNode = lastNode.next;
        secondLast = secondLast.next;
    }

    secondLast.next = null;
}
```

// main method

```
list.deleteFirst(); // Delete first
list.printList();

list.deleteLast();
list.printList(); // Delete last
```

Linked list using Collection

```
import java.util.*;
class LL {
    public static void main(String args[]) {
        LinkList<String> list = new LinkedList<String>();

        list.addFirst("a");
        list.addFirst("is");
        System.out.println(list);

        list.addFirst("this");
        list.addLast("list");
        System.out.println(list);

        System.out.println(list.size());

        for(int i=0; i<list.size(); i++) {
            System.out.println(list.get(i)+"->");
        }

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

Output

[is, a]
[This, is, a, list]
4
this -> is -> a -> list -> null

Linkedlist function

```
import java.util.LinkedList;
{
  LinkedList<String>animal = new LinkedList<>();
```

Add element

```
animal.add("dog");
animal.add(1,"horse");
animal.addfirst("Cow");
animal.addlast("Lion");
```

Access element

```
animal.get(1);
animal.getfirst();
animal.getlast();
```

Using iterator()

```
java.util.Iterator
Iterator<String> it = animals.iterator();
while(iterate.hasNext()){
  System.print(iterate.next());
}
```

⇒ hasNext() - return true if there is a next element

⇒ next() - return the next element

⇒ hasPrevious() like hasnext()

⇒ previous() like next()

Change element

Set();

Remove Element

remove()

* LinkList Vs Arrays

⇒ In Arrays elements are stored in Contiguous memory locations.

But,

In linked lists, elements are stored in non Contiguous memory location

* LinkList Vs ArrayList

① ArrayList internally used a dynamic array to store the element

Where,

LinkedList internally used a doubly linked list to store the element.

② In ArrayList element are stored in Contiguous memory locations.

Where,

In linked lists, elements are stored in non Contiguous memory location.

③ ArrayList is better for storing and accessing data.

But,

LinkedList is better for manipulating data.

Time Complexity

ArrayList		LinkedList
O(n)	← Insert →	O(1)
O(1)	← Search →	O(n)

* Why LinkedList

⇒ memory and the capacity of an array remains fixed. In case of linked list we can keep adding and removing element without any capacity constraints

* Drawback of linked lists

⇒ Extra memory space for pointers is required

⇒ Random access are not allowed as elements are not stored in contiguous memory location.

* Contiguous memory allocation

⇒ Contiguous memory allocation allocates consecutive blocks of memory to a file/process.

* non-Contiguous memory allocatn

⇒ non-Contiguous memory allocation allocates separate block of memory to a file/process.

* What is Collection in Java?

⇒ A Collection represents a single unit of object.

like - A Group

* framework:-

(I) It provide readymade architecture

(II) It represent a set of classes and interfaces

(III) It is optional

* Collection framework:-

⇒ The Collection framework represents a unified architecture for storing and manipulating a group of object. It has

(I) Interfaces and its implementations i.e - classes

(II) Algorithm