Doubly Linked List:

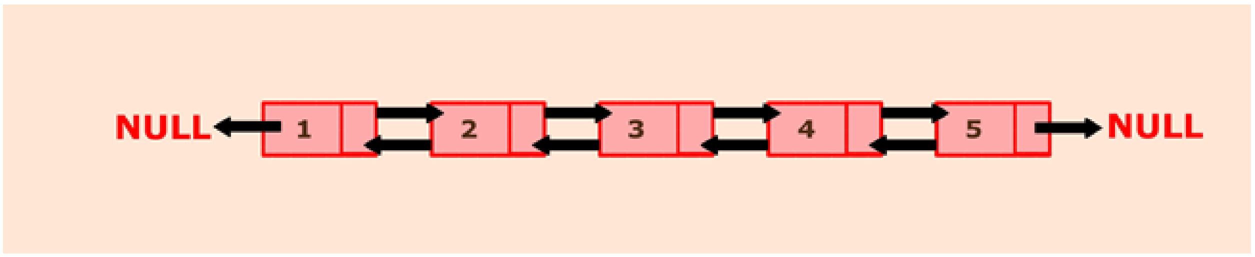
Doubly Linked List is a variation of the linked list. The linked list is a linear data structure which can be described as the collection of nodes. Nodes are connected through pointers. Each node contains two fields: data and pointer to the next field. The first node of the linked list is called the head, and the last node of the list is called the tail of the list.

One of the limitations of the singly linked list is that it can be traversed in only one direction that is forward. The doubly linked list has overcome this limitation by providing an additional pointer that points to the previous node. With the help of the previous pointer, the doubly linked list can be traversed in a backward direction thus making insertion and deletion operation easier to perform. So, a typical node in the doubly linked list consists of three fields:

Data represents the data value stored in the node.

Previous represents a pointer that points to the previous node.

Next represents a pointer that points to next node in the list.



Above picture represents a doubly linked list in which each node has two pointers to point to previous and next node respectively. Here, node 1 represents the head of the list. The previous pointer of the head node will always point to NULL. Next pointer of node one will point to node 2. Node 5 represents the tail of the list whose previous pointer will point to node 4, and next will point to **NULL**..

Algorithm

- Define a Node class which represents a node in the list. It will have three properties: data, previous which will point to the previous node and next which will point to the next node.
- Define another class for creating a doubly linked list, and it has two nodes: head and tail. Initially, head and tail will point to null.
- addNode() will add node to the list:
    - It first checks whether the head is null, then it will insert the node as the head.
    - Both head and tail will point to a newly added node.
    - Head's previous pointer will point to null and tail's next pointer will point to null.
    - If the head is not null, the new node will be inserted at the end of the list such that new node's previous pointer will point to tail.
    - The new node will become the new tail. Tail's next pointer will point to null.

a. display() will show all the nodes present in the list.

- Define a new node 'current' that will point to the head.
- Print current.data till current points to null.
- Current will point to the next node in the list in each iteration.

Program:

```
public class DoublyLinkedList {

    //Represent a node of the doubly linked list

```

```java
class Node{
    int data;
    Node previous;
    Node next;

    public Node(int data) {
        this.data = data;
    }
}

//Represent the head and tail of the doubly linked list
Node head, tail = null;

//addNode() will add a node to the list
public void addNode(int data) {
    //Create a new node
    Node newNode = new Node(data);

    //If list is empty
    if(head == null) {
        //Both head and tail will point to newNode
        head = tail = newNode;
        //head's previous will point to null
        head.previous = null;
        //tail's next will point to null, as it is the last node of the list
        tail.next = null;
    }
    else {
//newNode will be added after tail such that tail's next will point to newNode
```

```
            tail.next = newNode;
            //newNode's previous will point to tail
            newNode.previous = tail;
            //newNode will become new tail
            tail = newNode;
            //As it is last node, tail's next will point to null
            tail.next = null;
        }
    }

    //display() will print out the nodes of the list
    public void display() {
        //Node current will point to head
        Node current = head;
        if(head == null) {
            System.out.println("List is empty");
            return;
        }
        System.out.println("Nodes of doubly linked list: ");
        while(current != null) {
            //Prints each node by incrementing the pointer.

            System.out.print(current.data + " ");
            current = current.next;
        }
    }

    public static void main(String[] args) {

        DoublyLinkedList dList = new DoublyLinkedList();
        //Add nodes to the list
        dList.addNode(1);
        dList.addNode(2);
        dList.addNode(3);
        dList.addNode(4);
        dList.addNode(5);

        //Displays the nodes present in the list
        dList.display();
    }
}
```

Output:

```
1 2 3 4 5
```