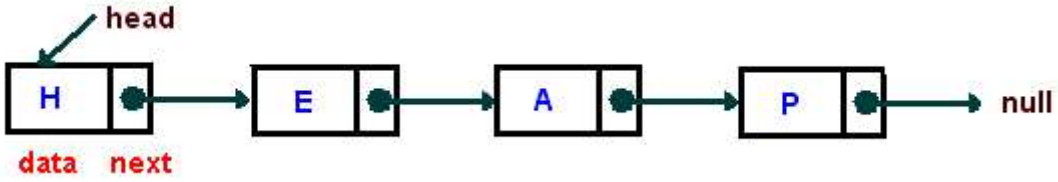
Lecture

NOTE: FOR FURTHER DETAILS AND MORE COMPREHENSIVE STUDY, PLEASE SEE RECOMMENDED BOOKS OR INTERNET.

Linked List

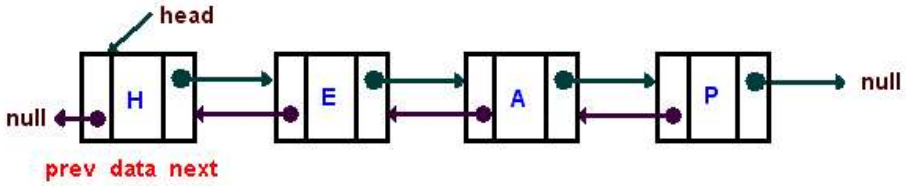
A linked list is a linear data structure where each element is a separate object or ***node***.



Linked List is an Abstract Data Type (ADT) that holds a collection of ***Nodes***, the nodes can be accessed in a sequential way. Linked List doesn't provide a random access to a ***Node***. Usually, those ***Nodes*** are connected to the next node and/or with the previous one, this gives the ***linked*** effect. When the Nodes are connected with only the ***next*** pointer the list is called Singly Linked List and when it's connected by the ***next*** and ***previous*** the list is called Doubly Linked List.

Doubly Linked List

A ***doubly linked list*** is a list that has two references, one to the ***next*** node and another to ***previous*** node.



Insertion and Deletion Operations

Doubly linked list guarantees the ***insertion*** and ***deletion*** of node at ***head*** or ***tail*** with the computational cost of ***O(1)***. However the insertion of deletion in somewhere between head or tail increases the computation cost by adding the cost of searching a specific node such that to insert a node after a certain node n or delete a node n will increase the cost of search by O(n). Today we will implement a doubly linked list with insertion and deletion at both ends with the computational cost of ***O(1).***

Your Home task will be to add insertion and deletion in between such that ***insertAfterKey(int key, int value)*** *and* ***deleteNodeWithKey(int key).***

Code for Doubly Linked List

```
public class DoublyLL <T>
{
    class Node <T>
    {
        T data;
        Node pre = null;
        Node next = null;
    }

    Node head;
    Node tail;

    public void insertAtStart(T value)
    {
        Node n = new Node();
        n.data = value;

        if(head == null)
        {
            head = n;
            tail = n;
        }
        else
        {
            n.next = head;
            head.pre = n;
            head = n;
        }
    }

    public T deleteAtStart()
    {
        if(head == null)
        {
            System.out.println("Linked List is Empty");
            return null;
        }
        else if(head == tail)
        {
            T value = (T) head.data;
            head =  null;
            tail = null;
            return value;
        }
        else
        {
            T value = (T) head.data;
            head = head.next;
            head.pre = null;
            return value;
```

```
        }
    }

    public void insertAtEnd(T value)
    {
        Node n = new Node();
        n.data = value;

        if(head == null)
        {
            head = n;
            tail = n;
        }
        else
        {
            tail.next = n;
            n.pre = tail;
            tail = n;
        }
    }

    public T deleteAtEnd()
    {
        if(tail == null)
        {
            System.out.println("Linked List is Empty");
            return null;
        }
        else if(head == tail)
        {
            T value = (T) tail.data;
            head =  null;
            tail = null;
            return value;
        }
        else
        {
            T value = (T) tail.data;
            tail = tail.pre;
            tail.next = null;
            return value;
        }
    }

    public void traverseForward()
    {
        Node temp = head;
        System.out.print("Forward Traversal: ");
        while(temp != null)
        {
            System.out.print(temp.data + "    ");
            temp = temp.next;
```

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

    public void traverseBackward()
    {
        Node temp = tail;
        System.out.print("Backward Traversal: ");
        while(temp != null)
        {
            System.out.print(temp.data + "   ");
            temp = temp.pre;
        }
        System.out.println("");
    }

    public void search(T key)
    {
        int count = 1;
        Node temp = head;
        while(temp != null)
        {
            if(temp.data == key)
            {
                System.out.println("Found at Node No. " + count);
                return;
            }
            temp = temp.next;
            count++;
        }
        System.out.println("Value Not Found");
    }
}
```

Code above is the code for Doubly Linked List class. However, to demonstrate the working of Doubly Linked List class we will use the following code in the main class.

```
public class DoublyLinkedListDemo
{
    public static void main(String[] args)
    {
        DoublyLL<Integer> dll = new DoublyLL<Integer>();

        dll.deleteAtStart();
        dll.deleteAtEnd();

        dll.insertAtStart(10);
        dll.insertAtStart(20);
        dll.insertAtStart(30);
        dll.insertAtStart(40);
        dll.traverseForward();
```

```
        dll.traverseBackward();
        dll.search(30);
        dll.search(50);
        System.out.println("Deleted At End: " + dll.deleteAtEnd());
        System.out.println("Deleted At End: " + dll.deleteAtEnd());
        dll.traverseForward();
        System.out.println("Deleted At End: " + dll.deleteAtEnd());
        System.out.println("Deleted At End: " + dll.deleteAtEnd());

        dll.deleteAtStart();
        dll.deleteAtEnd();

        dll.insertAtEnd(10);
        dll.insertAtEnd(20);
        dll.insertAtEnd(30);
        dll.insertAtEnd(40);
        dll.traverseForward();
        dll.traverseBackward();
        dll.search(30);
        dll.search(50);
        System.out.println("Deleted At Start: " + dll.deleteAtStart());
        System.out.println("Deleted At Start: " + dll.deleteAtStart());
        dll.traverseBackward();
        System.out.println("Deleted At Start: " + dll.deleteAtStart());
        System.out.println("Deleted At Start: " + dll.deleteAtStart());
    }
}
```

Output:

```
Linked List is Empty
Linked List is Empty
Forward Traversal: 40   30   20   10
Backward Traversal: 10   20   30   40
Found at Node No. 2
Value Not Found
Deleted At End: 10
Deleted At End: 20
Forward Traversal: 40   30
Deleted At End: 30
Deleted At End: 40
Linked List is Empty
Linked List is Empty
Forward Traversal: 10   20   30   40
Backward Traversal: 40   30   20   10
Found at Node No. 3
Value Not Found
Deleted At Start: 10
Deleted At Start: 20
Backward Traversal: 40   30
```

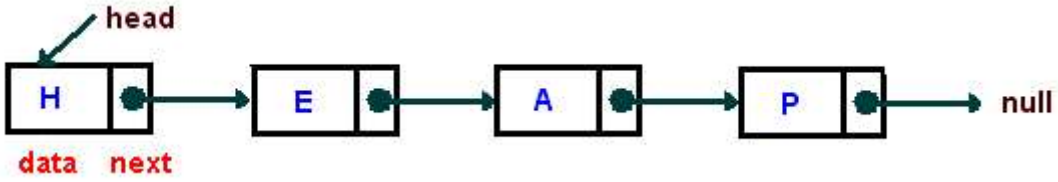
```
Deleted At Start: 30
Deleted At Start: 40
```

Lecture

NOTE: FOR FURTHER DETAILS AND MORE COMPREHENSIVE STUDY, PLEASE SEE RECOMMENDED BOOKS OR INTERNET.

Linked List

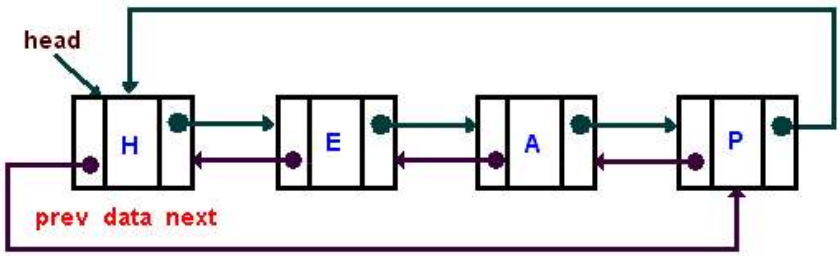
A linked list is a linear data structure where each element is a separate object or ***node***.



Linked List is an Abstract Data Type (ADT) that holds a collection of ***Nodes***, the nodes can be accessed in a sequential way. Linked List doesn't provide a random access to a ***Node***. Usually, those ***Nodes*** are connected to the next node and/or with the previous one, this gives the ***linked*** effect. When the Nodes are connected with only the ***next*** pointer the list is called Singly Linked List and when it's connected by the ***next*** and ***previous*** the list is called Doubly Linked List.

Circular Linked List

A ***circular linked list*** is where last node of the ***singly linked list*** points back to the first node (or the head) of the list. If the list is ***doubly linked list*** then the last node points next to the first node (or the head) of the list and first node points previous to the last node (or the tail) of the list as shown below.



Code for Circular Linked List (Doubly)

```
public class CircularLL <T>
{
    class Node <T>
    {
        T data;
        Node pre = null;
        Node next = null;
    }
```

```
Node head;
Node tail;
int count = 0;

public void insertAtStart(T value)
{
    Node n = new Node();
    n.data = value;

    if(head == null)
    {
        head = n;
        tail = n;
        head.pre = tail;
        tail.next = head;
        count++;
    }
    else
    {
        n.next = head;
        head.pre = n;
        head = n;
        tail.next = head;
        head.pre = tail;
        count++;
    }
}

public T deleteAtStart()
{
    if(head == null)
    {
        System.out.println("Linked List is Empty");
        return null;
    }
    else if(head == tail)
    {
        T value = (T) head.data;
        head =  null;
        tail = null;
        count--;
        return value;
    }
    else
    {
        T value = (T) head.data;
        head = head.next;
        head.pre = tail;
        tail.next = head;
        count--;
        return value;
    }
```

```
}

public void insertAtEnd(T value)
{
    Node n = new Node();
    n.data = value;

    if(head == null)
    {
        head = n;
        tail = n;
        head.pre = tail;
        tail.next = head;
        count++;
    }
    else
    {
        n.pre = tail;
        tail.next = n;
        tail = n;
        head.pre = tail;
        tail.next = head;
        count++;
    }
}

public T deleteAtEnd()
{
    if(tail == null)
    {
        System.out.println("Linked List is Empty");
        return null;
    }
    else if(head == tail)
    {
        T value = (T) tail.data;
        head =  null;
        tail = null;
        count--;
        return value;
    }
    else
    {
        T value = (T) tail.data;
        tail = tail.pre;
        tail.next = head;
        head.pre = tail;
        count--;
        return value;
    }
}
```

```
    public void traverse()
    {
        Node temp = head;
        System.out.print("Items in List: ");
        for(int i = 1; i<=count ;i++)
        {
            System.out.print(temp.data + "   ");
            temp = temp.next;
        }
        System.out.println("");
    }

    // This is test method to check weather
    // the list is circular if it is then
    // it will print the list twice.
    public void traverse2()
    {
        Node temp = head;
        System.out.print("Items in List: ");
        for(int i = 1; i<=2*count ;i++)
        {
            System.out.print(temp.data + "   ");
            temp = temp.next;
        }
        System.out.println("");
    }

    public void search(T key)
    {
        Node temp = head;
        for(int i = 1; i<=count; i++)
        {
            if(temp.data == key)
            {
                System.out.println(key +" Found in the List");
                return;
            }
            temp = temp.next;
        }
        System.out.println(key + " Not Found in the List");
    }
}
```

Code above is the code for Circular Linked List class. However, to demonstrate the working of circular Linked List class we will use the following code in the main class.

```
public class CircularLinkedListDemo
{
    public static void main(String[] args)
    {
        CircularLL<Integer> cll = new CircularLL<Integer>();
```

```
        cll.deleteAtStart();
        cll.deleteAtEnd();

        cll.insertAtStart(10);
        cll.insertAtStart(20);
        cll.insertAtStart(30);
        cll.insertAtStart(40);
        cll.traverse();

        // Testing weather the list id circular
        // It Should Print Twice
        cll.traverse2();

        cll.search(30);
        cll.search(50);
        System.out.println("Deleted At End: " + cll.deleteAtEnd());
        System.out.println("Deleted At End: " + cll.deleteAtEnd());
        cll.traverse();
        System.out.println("Deleted At End: " + cll.deleteAtEnd());
        System.out.println("Deleted At End: " + cll.deleteAtEnd());

        cll.deleteAtStart();
        cll.deleteAtEnd();

        cll.insertAtEnd(10);
        cll.insertAtEnd(20);
        cll.insertAtEnd(30);
        cll.insertAtEnd(40);
        cll.traverse();

        // Testing weather the list id circular
        // It Should Print Twice
        cll.traverse2();

        cll.search(30);
        cll.search(50);
        System.out.println("Deleted At Start: " + cll.deleteAtStart());
        System.out.println("Deleted At Start: " + cll.deleteAtStart());
        cll.traverse();
        System.out.println("Deleted At Start: " + cll.deleteAtStart());
        System.out.println("Deleted At Start: " + cll.deleteAtStart());
    }
}
```

Output:

```
Linked List is Empty
Linked List is Empty
Items in List: 40   30   20   10
Items in List: 40   30   20   10   40   30   20   10
30 Found in the List
```

```
50 Not Found in the List
Deleted At End: 10
Deleted At End: 20
Items in List: 40   30
Deleted At End: 30
Deleted At End: 40
Linked List is Empty
Linked List is Empty
Items in List: 10   20   30   40
Items in List: 10   20   30   40   10   20   30   40
30 Found in the List
50 Not Found in the List
Deleted At Start: 10
Deleted At Start: 20
Items in List: 30   40
Deleted At Start: 30
Deleted At Start: 40
```