Linked List Data Structure

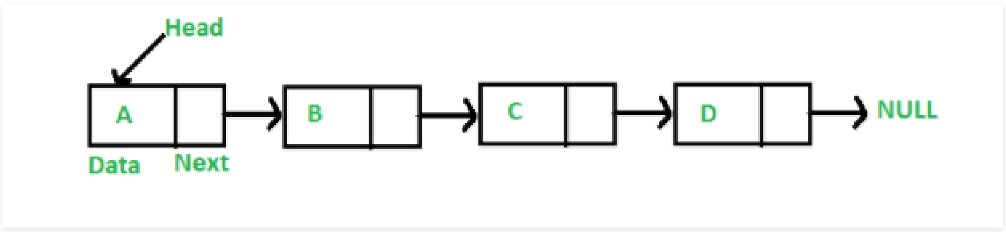
Static and Dynamic data structures

A **static data structure** is an organization of collection of data that is fixed in size. This results in the maximum size needing to be known in advance ("مسبقا), as memory cannot be reallocated at a later. Arrays are example of static data structure.

A dynamic data structure , where in with the latter the size of the structure can dynamically grow or shrink in size as needed. Linked lists are example of dynamic data structure.

Linked Lists

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at contiguous location; the elements are linked using pointers. It consists of group of nodes in a sequence which is divided in two parts. Each node consists of its own data and the address of the next node and forms a chain. Linked Lists are used to create trees and graphs.



What are different between linked lists and arrays?

Arrays	Linked lists
Have a pre-determined fixed size	No fixed size; grow one element at a time
Array items are store contiguously	Element can be stored at any place
Easy access to any element in constant time	No easy access to i-th element , need to hop through all previous elements start from start
Size = n x sizeof(element)	Size = n x sizeof (element) + n x sizeof(reference)
Insertion and deletion at particular position is complex(shifting)	Insertion and deletion are simple and faster
Only element store	Each element must store element and pointer to next element (extra memory for storage pointer).

Q: Why Linked List?
Arrays can be used to store linear data of similar types, but arrays have following limitations.
1) The size of the arrays is fixed**.**
2) Inserting a new element in an array of elements is expensive.

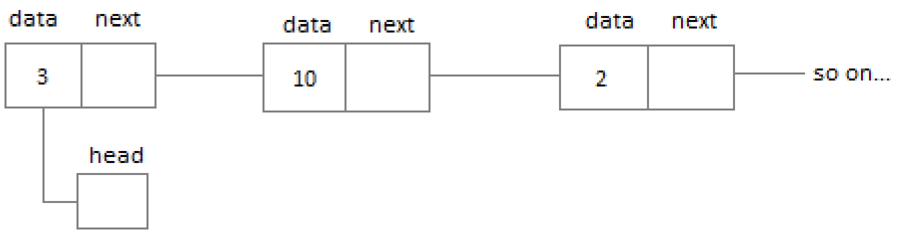
Advantages of Linked Lists
1) Dynamic size
2) Ease of insertion/deletion

Disadvantages of Linked Lists
1) Random access is not allowed.
2) Extra memory space for a pointer is required with each element of the list.

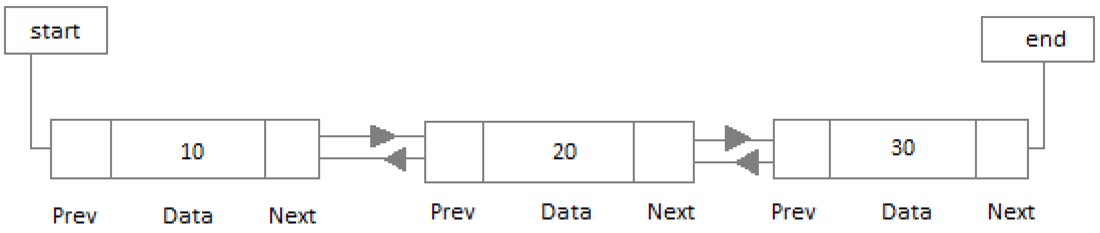
Applications of Linked Lists
Linked lists are used to implement stacks, queues, graphs, etc.

Types of Linked Lists

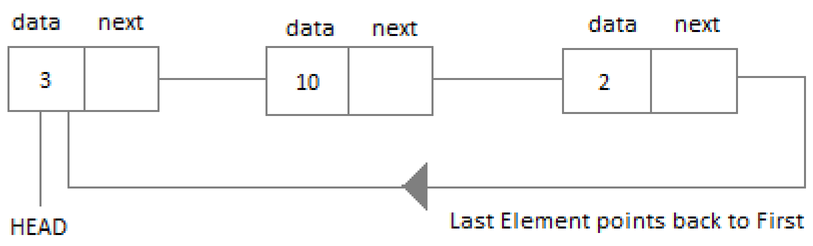
- **Singly (Linear) Linked List :** Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in sequence of nodes.



- **Doubly Linked List :** In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence.



- **Circular Linked List :** In the circular linked list the last node of the list contains the address of the first node and forms a circular chain.



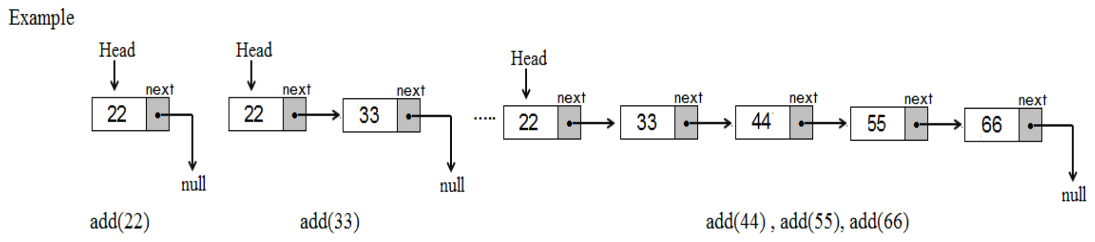
Basic Operations

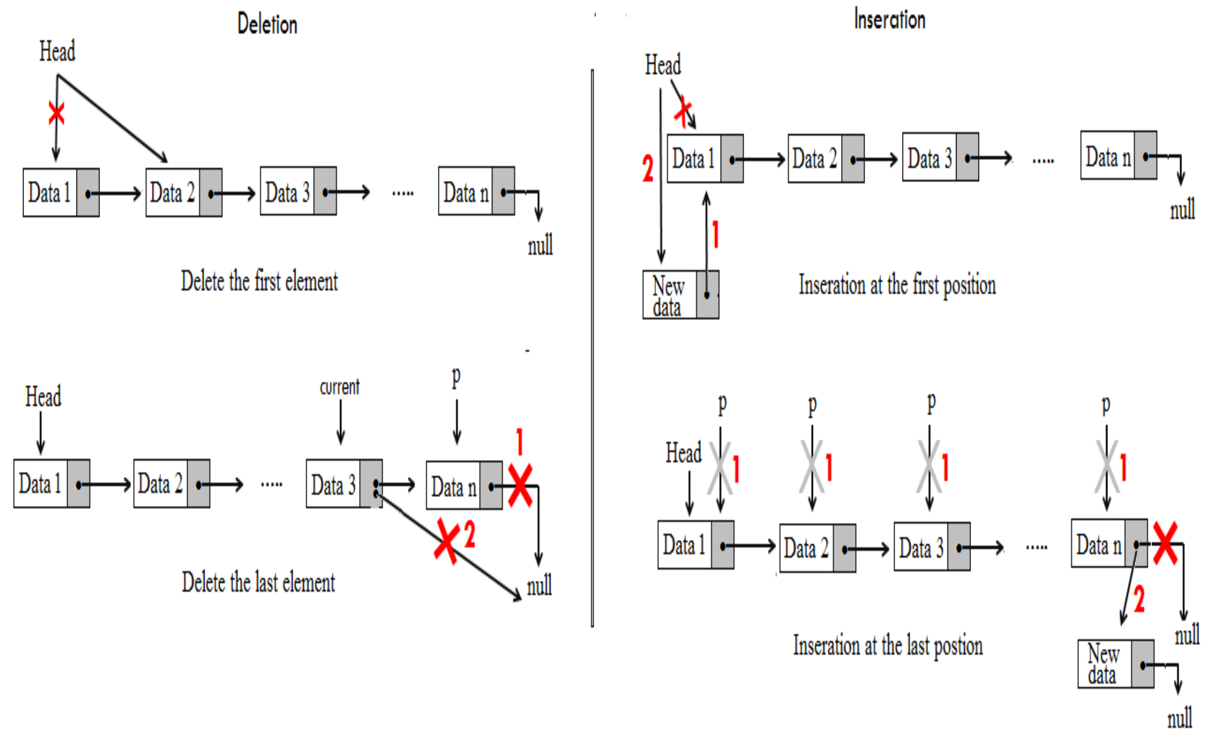
More useful operations on linked list are:

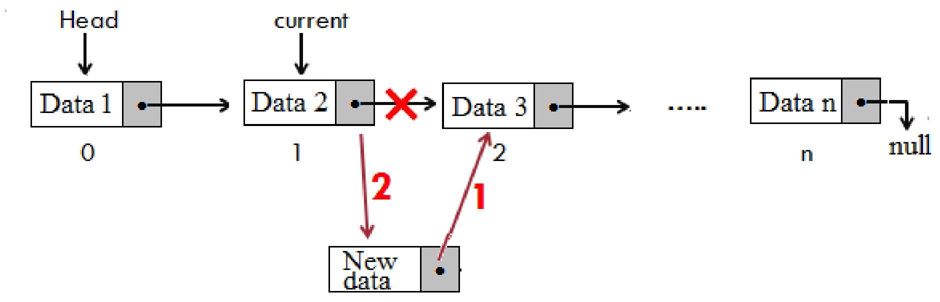
- Inserting a particular Node from the List
- Checking whether Linked List is empty or not.
- Searching any element in the Linked List
- Deleting a particular Node from the List

Singly Linked List

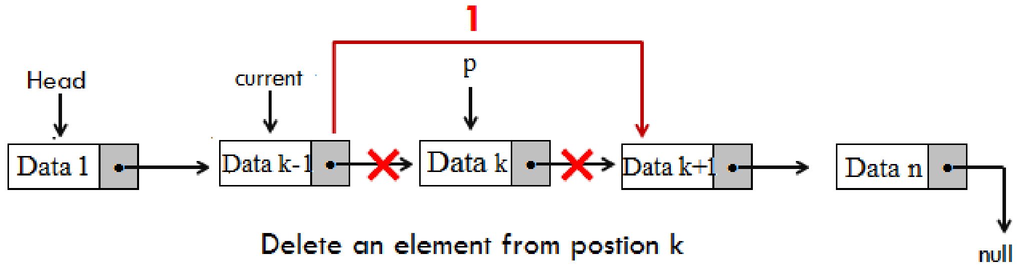
In the following figure shown some operation on singly linked list :







Insert an elmenet at postion 2 (the third element)



Delete an element from postion k

To construct singly linked list we need define two class:
1. node class
2. singly linked list class(SLL)

```
class Node {
   int data;
   node next;

    node(int data) {
       this.data = data;}
 }
```

```
class SLL {
   node head;

   public SLL() {
      this.head = null;}
   // methods
  }
}
```

In the following steps for create linked list with data(22, 33, 44, 55,66)

```
node head = new node(22);
head.next = new node(33);
head.next.next = new node(44);
head.next.next.next = new node(55);
head.next.next.next.next = new node(66);
```

or

```
head = new node(22);
node p= head;
p.next = new node(33);
p = p.next;
p.next = new node(44);
p = p.next;
p.next = new node(55);
p = p.next;
p.next = new node(66);
```

Some code segments in java to processing singly linked list

```
// method to print elements of singly linked list
public void printSLL() {
for (node p = head; p != null; p = p.next) {
  System.out.print(p.data + " "); }
}
```

```
// method for adding new node
void addLast ( int value){
      node newNode = new node(value);
      node p = head;
      if (head == null)
        head = newNode;

      else {
         while (p.next != null)
            p = p.next;
         p.next = newNode;
      }
   }
```

OR

```
void addLast(int value){
  node p;
  node newNode = new node(value);
  if (head==null)
    head=newNode;
 else {
   for (p = head; p.next!= null; p=p.next)
     { }
     p.next=newNode;
  }
}
```

```
// method to delete first element
public void deleteFirst() {
    head = head.next;

}
```

OR

```
public void deleteFirst() {
 node curr = head;
 head = head.next;
 curr.next = null;
 curr = null;}
```

```
//method to delete element after element in the list
public void deleteAfter(int value) {
    node p = head;
  while (p.data != value)
      p = p.next;
   node curr=p.next;
   p.next= curr.next;
}

// method to add an element after an item of the list
public void addAfter(int value, int newValue) {
  node p = head;
  while (p.data != value)
      p = p.next;
  node newNode = new node(newValue);
  newNode.next = p.next;
  p.next = newNode;}
```

code segments in java to processing singly linked list:

```
public class Node {
    int data;
    Node next;

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

```
public class SingleLL {
    Node head;
    SingleLL(){
        head=null;
    }

    void printList(){
        for (Node p=head; p!=null; p=p.next ) {
            System.out.print(p.data+" ");
        }
        System.out.println();
    }

    void addItem( int data){
        Node p= new Node(data);
        Node c=head;
        if (head==null)
          head=p;
        else{
            for (c=head; c.next!=null; c=c.next){}
            c.next=p;}
       }

    void addBefore(int data, int item){
        Node p=new Node(data);
        Node cur=head;
        Node prv=head;
```

```
     if (head.data==item){
         p.next=head;
         head=p;}
     else{
       while (cur.data!=item && cur.next!=null){
         prv=cur;
         cur=cur.next; }
      if (cur.next==null)
        addItem(data);
      else{
        p.next=cur;
        prv.next=p;}
     }
 }
    void addAfter(int data, int item){
       Node p=new Node(data);
       Node cur=head;
       while (cur.data!=item && cur.next!=null)
           cur=cur.next;
       if (cur.next==null) addItem(data);
       else {
         p.next=cur.next;
         cur.next=p;}
     }


    void deleteItem(int item){
        Node cur=head;
        Node prv=head;
        if (head.data==item)
            head=cur.next;
        else{
            while (cur.data!=item && cur.next!=null){
                prv=cur;
                cur=cur.next;
            }
          if (cur.next==null && cur.data==item) prv.next=null;
          else if (cur.next==null)
             System.out.println(item+ "  not found");
          else
             prv.next=cur.next;
            }
    }
}
```

Doubly linked lists

In this type of linked list each node contains a pair of references. One reference points to the node that precedes the node (prior) and the other points to the node that follows the node (next).

Advantages:

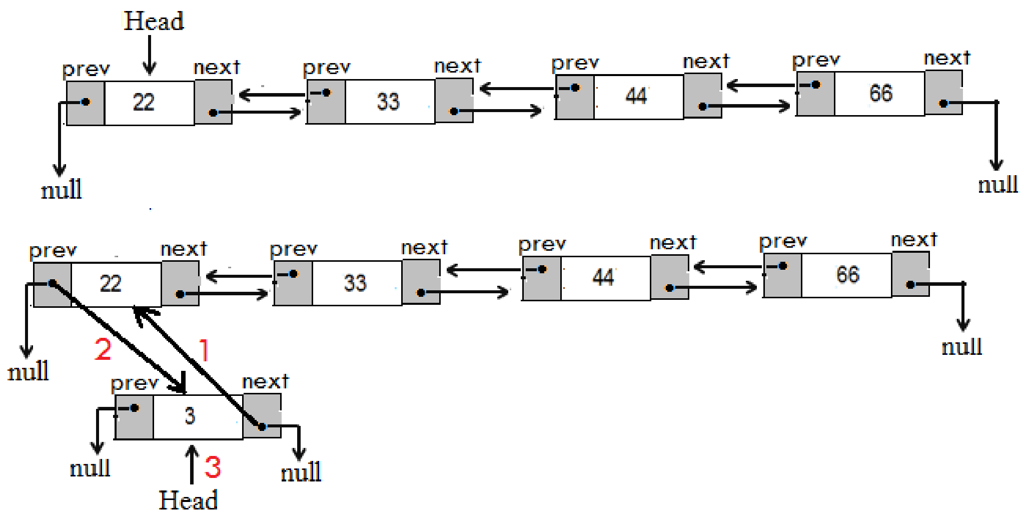
- Can be traversed in either direction.
- Some operations, such as deletion and inserting before a node, become easier.

Disadvantages:

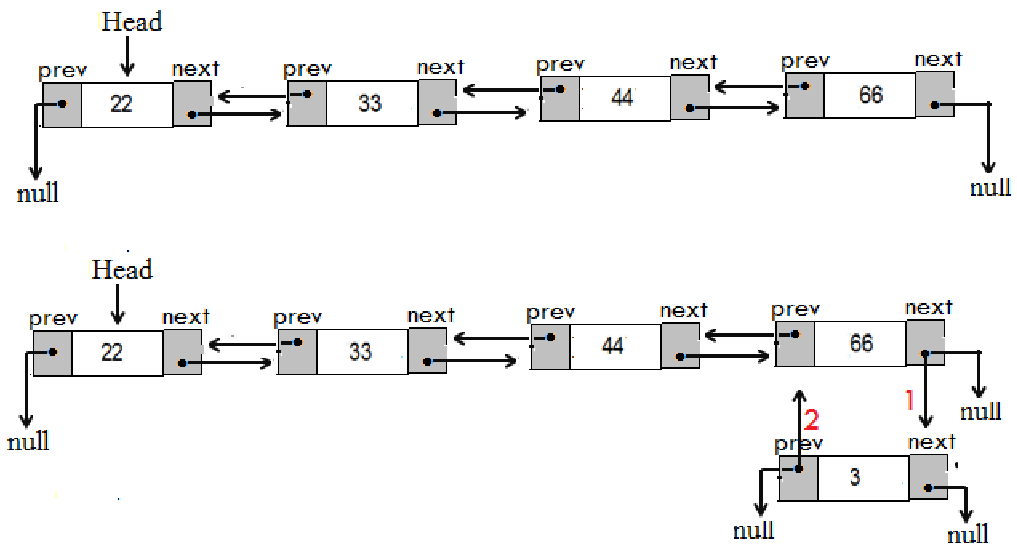
- Requires more space.
- List manipulations are slower (because more links must be changed)
- Greater chance of having errors (because more links must be manipulated)

In the following figure shown some operation on double linked list :

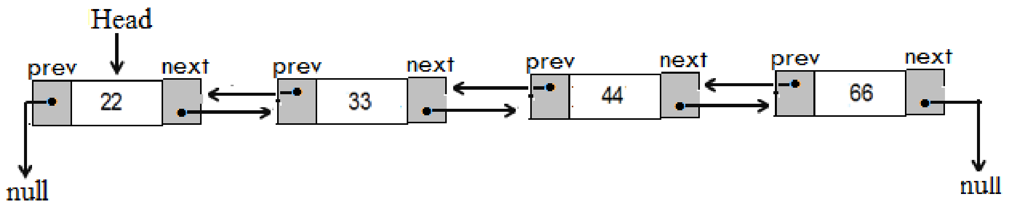
insert in the first position

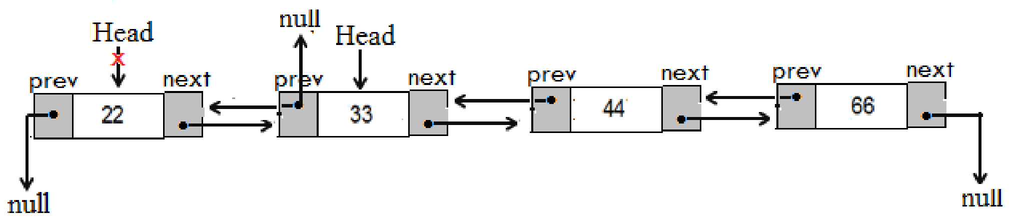


insert in last position

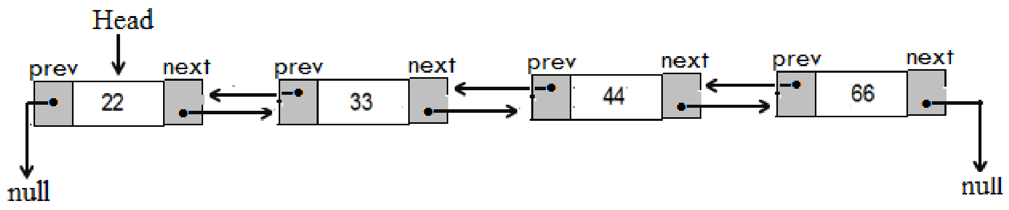


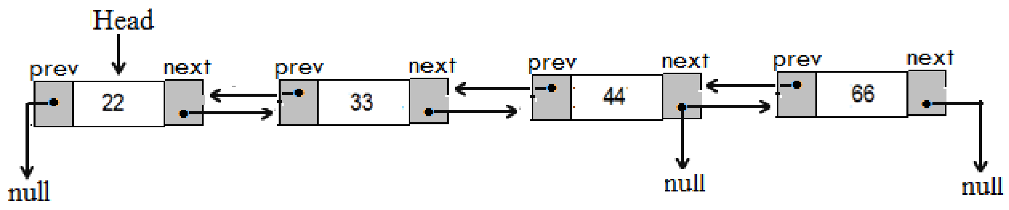
delete first element





delete last element





To construct double linked list we need define two class:

```
class DListNode  {
    int data;
    DListNode  prior,  next;
 }// DListNode
```

```
class DLL {
  node head;
  public DLL() {
    this.head = null;}

    // methods
}
```

code segments in java to processing double linked list:

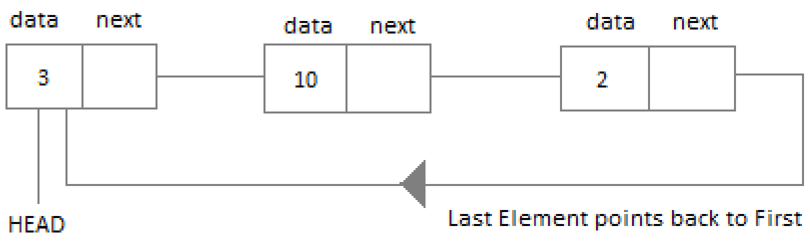
```
public class DNode {
    int data;
    DNode next, prev;

    DNode(int data){
      this.data=data;
      next = null;
      prev = null; }
   }


public class DoubleLL {
    DNode head=null;
    DoubleLL(){
        head=null;
    }

    void printDLL(){
        for (DNode p=head; p!=null; p=p.next)
          System.out.print(p.data + " ");
        System.out.println();
    }

    void RprintDLL(){
        DNode p;
        for (p=head; p.next!=null; p=p.next){}
        for (; p!=null; p=p.prev)
          System.out.print(p.data + " ");
        System.out.println(); }

    void addItem(int data){
        DNode p = new DNode(data);
        DNode c;
        if (head==null)
          head=p;
        else{
           for (c=head; c.next!=null; c=c.next){}
           c.next=p;
           p.prev=c;}
     }

    void addAfter(int data, int item){
      DNode p= new DNode(data);
      DNode c=head;
      while (c.data!=item && c.next!=null)
          c=c.next;
```

```
       if (c.next==null && c.data!=item)
          System.out.println(item + "  not found");
       else if (c.next==null)
           addItem(data);
       else{
          p.next=c.next;
          c.next.prev=p;
          c.next=p;
          p.prev=c;}
        }

     void addBefore(int data, int item){
        DNode p=new DNode(data);
        DNode c=head;
        if (head.data==item){
           p.next=head;
           head.prev=p;
           head=p;}
        else {
           while (c.data!=item && c.next!=null)
              c=c.next;
           if (c.data!=item && c.next==null)
              System.out.println(item + "  not found");

           else{
            p.next=c;
            DNode q=c.prev;
            q.next=p;
            p.prev=q;
            c.prev=p;}
           }
      }

  void deleteItem(int item){
        DNode p=head;
        if (head.data==item){
           head.next.prev=null;
           head=head.next;
        }
        else {
           while (p.data!=item && p.next!=null)
              p=p.next;
           if (p.data!=item && p.next==null)
               System.out.println(item + "  not found");
           else if (p.data==item && p.next==null){
             DNode q=p.prev;
             q.next=null;}
```

```
else {
  DNode q=p.prev;
  q.next=p.next;
  p.next.prev=q;
}
  }
```

Circular Linked List

Circular Linked List is little more complicated linked data structure. In the circular linked list the previous element stores the address of the next element and the last element stores the address of the starting element. The elements points to each other in a circular way which forms a circular chain.



in this type of linked list:

- Last node references the first node
- Every node has a successor
- No node in a circular linked list contains *null*

Both singly linked list and doubly linked list can be made into as circular linked list:

- Singly linked list as circular, the next pointer of the last node points to the first node.

- Doubly linked list as circular, the next pointer of the last node points to the first node and the previous pointer of the first node points to the last node making the circular in both directions

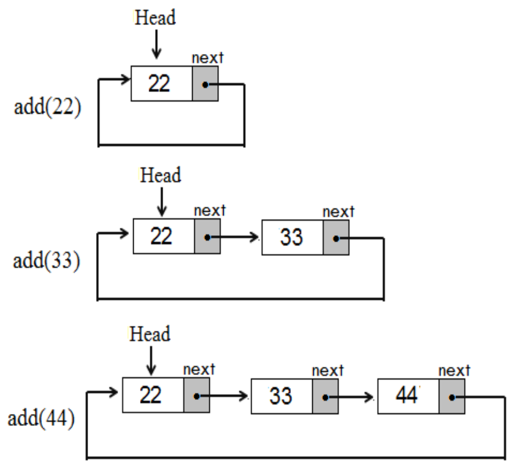
To construct circular linked list we need define two class:

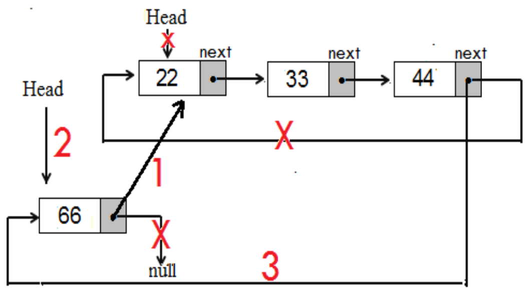
1. node class
2. circular linked list class(CLL)

```
class Node {
    int data;
    node next;

     node(int data) {
        this.data = data;
   }
 }
```

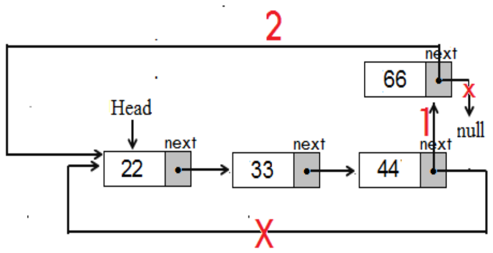
```
class CLL {
  node head;

  public CLL() {
    this.head = null;

}
```

Insertion



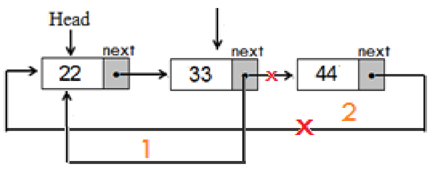


insert in the first position

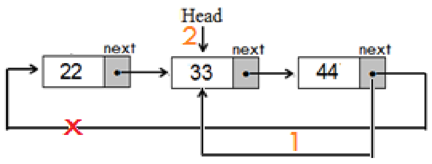


insert in the last position

Delete



delete last element



delete first element

code segments in java to processing circular linked list

```
public class CirclLL {
   Node head;
   CirclLL(){
      head=null;
   }

   void printList(){
      Node p;
      for (p=head; p.next!=head; p=p.next ) {
         System.out.print(p.data+" ");
      }
      System.out.println(p.data+"\n");
   }
```

```
void addItem( int data){
    Node p= new Node(data);
    Node c=head;
    if (head==null){
      head=p;
      p.next=head;}

    else{
        for (c=head; c.next!=head; c=c.next){}
        c.next=p;
        p.next=head;}
   }

void addBefore(int data, int item){
    Node p=new Node(data);
    Node cur=head;
    Node prv=head;
    if (head.data==item){
        for (cur=head; cur.next!=head; cur=cur.next ) {}
        p.next=head;
        head=p;
        cur.next=head;
        }

    else{
      while (cur.data!=item && cur.next!=head){
        prv=cur;
        cur=cur.next; }
      if ( cur.next==head && cur.data==item){
          p.next=cur;
          prv.next=p;
      }

      else if (cur.next==head)
       addItem(data);
     else{
       p.next=cur;
       prv.next=p;}
    }
}

   void addAfter(int data, int item){
      Node p=new Node(data);
      Node cur=head;
      while (cur.data!=item && cur.next!=null)
          cur=cur.next;
      if (cur.next==head) addItem(data);
      else {
        p.next=cur.next;
        cur.next=p;}
    }


   void deleteItem(int item){
       Node cur=head;
       Node prv=head;
       if (head.data==item){
          for (cur=head; cur.next!=head; cur=cur.next ) {}
          head=head.next;
          cur.next=head;}
       else{
          while (cur.data!=item && cur.next!=head){
             prv=cur;
             cur=cur.next;
          }
        if (cur.next==head && cur.data==item) prv.next=head;
        else if (cur.next==head)
           System.out.println(item+ "  not found");
        else
           prv.next=cur.next;
          }
   }
}
```