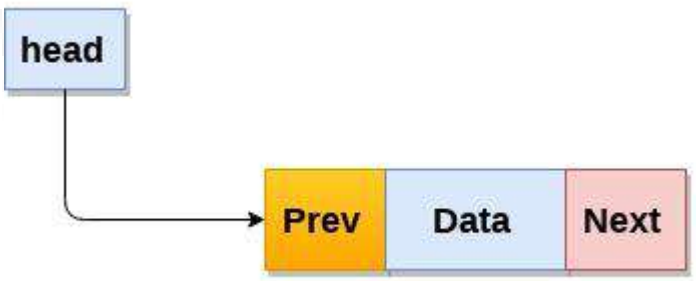
Lesson 9-10

Objectives

- Double Link List
  - Operations on Double Link List
    - Insertion
      - Start
      - End
      - After
      - Before
    - Deletion
      - Start
      - End
      - Specific node
    - Search Given Item
    - Locate Item at Given Position
- Link List Having Integer as Data Item

Introduction

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer). A sample node in a doubly linked list is shown in the figure.



We now know that a node in a doubly linked list must contain three variables:

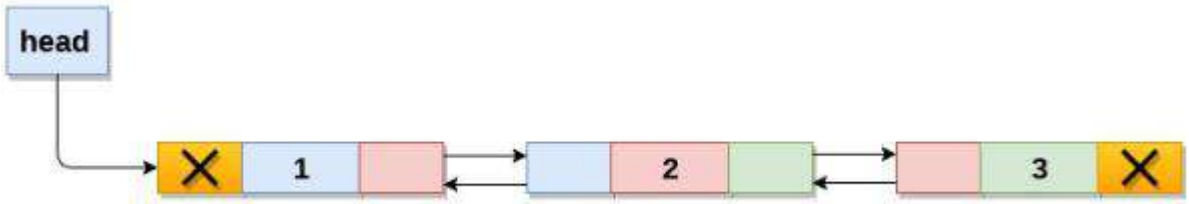
- A variable containing the actual data.

- A variable storing the pointer to the next node.

- A variable storing the pointer to the previous node.

With this information in hand, we can now create the class.

A doubly linked list containing three nodes having numbers from 1 to 3 in their data part, is shown in the following image.



Each element (we will call it a node) of a list is comprising of three items - the data and two reference to the next and previous node. The last node has a reference to null. The entry point into a linked list is called the head of the list. It should be noted that head is not a separate node, but the reference to the first node. If the list is empty then the head is a null reference.
A double linked list is a dynamic data structure. The number of nodes in a list is not fixed and can grow and shrink on demand. Any application which has to deal with an unknown number of objects will need to use a linked list.

One disadvantage of a linked list against an array is that it does not allow direct access to the individual elements. If you want to access a particular item then you have to start at the head and follow the references until you get to that item.

So we are ready for creating linked list.

```
public class Double_Link_List {

        private int data;

       private Double_Link_List next;
```

Doubly Linked list structure

Like a singly linked list, a doubly-linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Unlike a singly linked list, each node of the doubly singly list contains two fields that are references to the previous and to the next node in the sequence of nodes. The beginning and ending nodes' previous and next links, respectively, point to some kind of terminator, typically a sentinel node or null, to facilitate traversal of the list.
Doubly linked lists are like singly linked lists, except each node has two pointers -- one to the next node, and one to the previous node. This makes life nice in many ways:

- You can traverse lists forward and backward.

- You can insert anywhere in a list easily. This includes inserting before a node, after a node, at the front of the list, and at the end of the list.
- You can delete nodes very easily.

1) Insert from Start

At first initialize node type.

```
AnyType head = null;                    //empty linked list
```

Then we take the data input from the user and store in the `AnyType info` variable. Create a temporary node `AnyType temp` and allocate space for it.

```
Double_Link_List p;
p=new Double_Link_List();
```

Then place `info` to `temp.data`. So the first field of the `node temp` is filled. Now `temp.next and temp.prev` must become a part of the remaining linked list (although now linked list is empty but imagine that we have a 2 node linked list and head is pointed at the front) So `temp.next` must copy the address of the `head` (Because we want to insert at start) and `temp.prev must have head address` we also want that `head` will always point at front. So `head` must copy the address of the `node temp`.

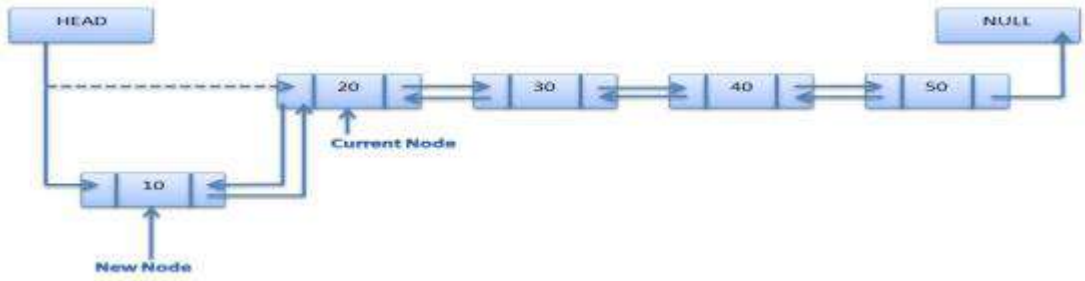


Figure: Insert at Start

```
Temp.data = info;                       // store data(first field)
Temp.next=null;    // store the address of the pointer head(second field)
Temp.prev=head;
head = temp;                            // transfer the address of 'temp' to 'head'
```

2) Traverse(Display Function)

Now we want to see the information stored inside the linked list. We create node `temp1`. Transfer the address of `head` to `temp1`. So `temp1` is also pointed at the front of the linked list. Linked list has 3 nodes.

We can get the data from first node using `temp1.data`. To get data from second node, we shift `temp1` to the second node. Now we can get the data from second node.

```
Temp=Head;
while ( temp1!=null )
{
System.out.println( temp1.data);// show the data in the linked list
 temp1 = temp1.next;      // transfer the address of 'temp.next' to 'temp'
}
```

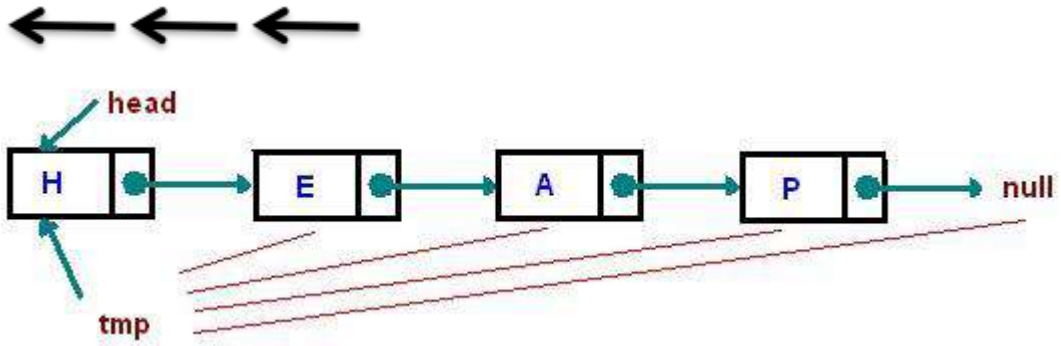


Figure: Traverse

This process will run until the linked list's next is *null*.

3) Insert from End

Insert data from End is very similar to the insert from start in the linked list. Here the extra job is to find the last node of the linked list.

```
Double_Link_lsit p;                                  // create a temporary node
p=new Double_Link_List ();                           // allocate space for node
if (Head==null)
  Head=p;                  // transfer the address of 'head' to
else
       {
        temp1 = head;
        while(temp1.next!=null)  // go to the last node
       temp1 = temp1.next;//tranfer the address of 'temp1.next' to 'temp1'

           }
Temp1.next=p;
p.prev=Temp1;
```

Now, Create a temporary node `node temp` and allocate space for it. Then place `info` to temp.data, so the first field of the node `node temp` is filled. `node temp` will be the last node of the linked list. For this reason, `temp.next` will be *null*. To create a connection between linked list and the new node, the last node of the existing linked list `node temp1`s second field `temp1.next` is pointed to `node temp and p node.prev must points to Temp1`.

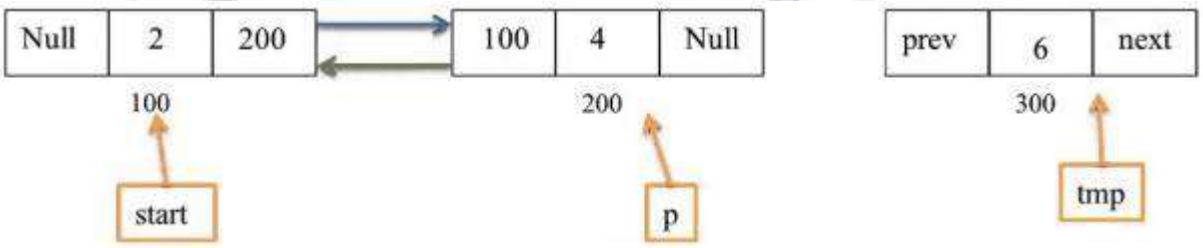


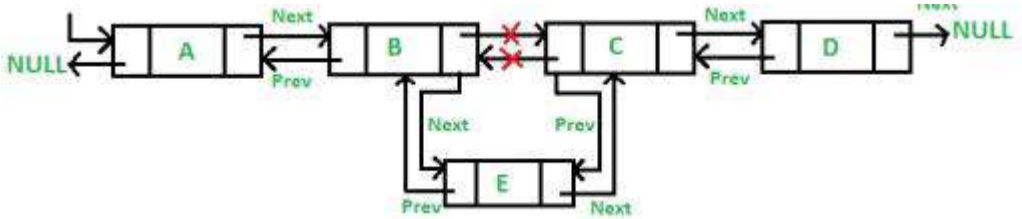
Figure: Insert at End

```
node temp;                                  // create a temporary node
temp = new node;    // allocate space for node
temp.data = info;                          // store data(first field)
temp.next = null;                          // second field will be null(last node)
temp1.next = temp;                         // 'temp' node will be the last node
```

```
temp.prev = temp1;                    // 'temp' node will be the last node
```

4) Insert after specified number of nodes

Insert data in the linked list after specified number of node is a little bit complicated. But the idea is simple. Suppose, we want to add a node after 2nd position. So, the new node must be in 3rd position. The first step is to go the specified number of node. Let, node temp1 is pointed to the 2nd node now.




```
System.out.println("ENTER THE NODE NUMBER:");
node_number=scan.NextInt();                    // take the node number from user

node temp1;                                    // create a temporary node
temp1=new node(); // allocate space for node
temp1 = head;

for( int i = 1 ; i < node_number ; i++ )
{
      temp1 = temp1.next;                      // go to the next node

if( temp1 == null )
      {
            System.out.println(node_number+" node is not exist");
break;
      }
}
```

Now, Create a temporary node `node temp` and allocate space for it. Then place `info` to `temp.data` , so the first field of the node `node temp` is filled.

```
node temp;                                     // create a temporary node
temp = new node();                             // allocate space for node
temp.data = info;                              // store data(first field)
```

To establish the connection between new node and the existing linked list, new node's `next` must pointed to the 2nd node's (temp1) `next`. The 2nd node's (temp1) next must pointed to the new node(temp).

```
temp.next = temp1.next;                  //transfer the address of temp1.next to temp1next.prev=temp;
temp1.next = temp;        //transfer the address of temp to temp1.next
temp.prev=temp1;
```

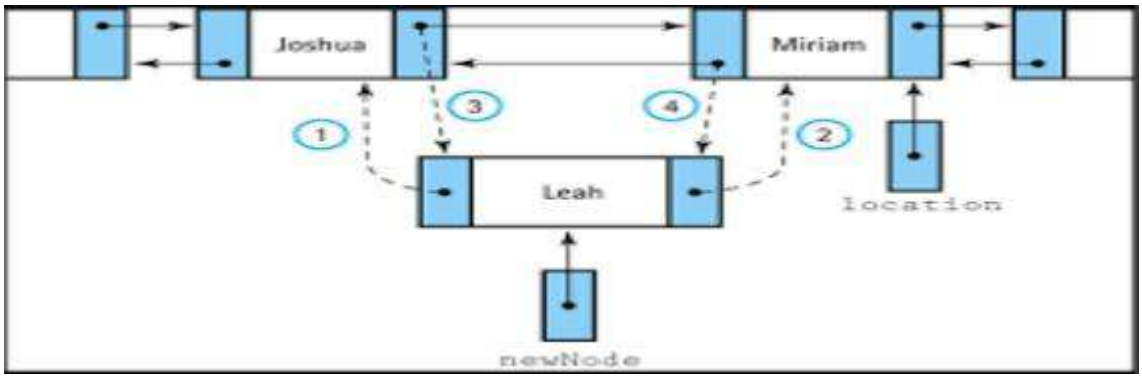


Figure: Insert after specified number of nodes

Deletion

Find a node containing "key" and delete it. In the picture below we delete a node containing "A"

The algorithm is similar to insert "before" algorithm. It is convenient to use two references prev and cur. When we move along the list we shift these two references, keeping prev one step before cur. We continue until cur reaches the node which we need to delete. There are three exceptional cases, we need to take care of:

1. list is empty
2. delete the head node
3. node is not in the list

5) Delete from Start

Delete a node from linked list is relatively easy. First, we create node `temp`. Transfer the address of `head` to `temp`. So `temp` is pointed at the front of the linked list. We want to delete the first node. So transfer the address of `temp.next` to `head` so that it now pointed to the second node. Now free the space allocated for first node. We can also use Delete keyword for deletion in Linked List.

```
node temp;                                          // create a temporary node
temp = new node();   // allocate space for node
temp = head;                                        // transfer the address of 'head' to 'temp'
head = temp.next;          // transfer the address of 'temp.next' to 'head'
```

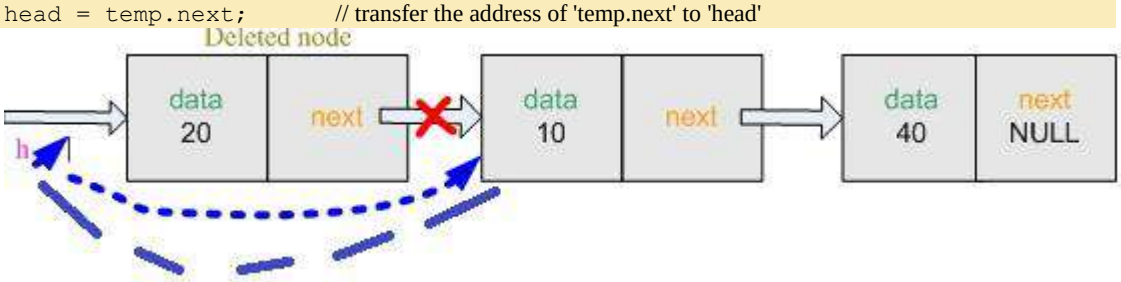


Figure: Delete from start

6) Delete from End

The last node`s `next` of the linked list always pointed to *null*. So when we will delete the last node, the previous node of last node is now pointed at *NULL*. So, we will track last node and previous node of the last node in the linked list. Create temporary `node temp1` and `old_temp`.

```
// create a temporary node
node temp1;
temp1 = new node();                          // allocate space for node
temp1 = head;                                //transfer the address of head to temp1
node old_temp;                               // create a temporary node
old_temp = new node();                       // allocate space for node

while(temp1.next!=null)                      // go to the last node
{
       Old.temp = temp1;     // transfer the address of 'temp1' to 'old_temp'
       temp1 = temp1.next;    //transfer the address of 'temp1.next' to 'temp1'
}
```

Now `node temp1` is now pointed at the last node and `old_temp` is pointed at the previous node of the last node. Now rest of the work is very simple. Previous node of the last node `old_temp` will be NULL so it become the last node of the linked list. Free the space allocated for last lode.

```
old_temp.next = null;                // previous node of the last node is null
```

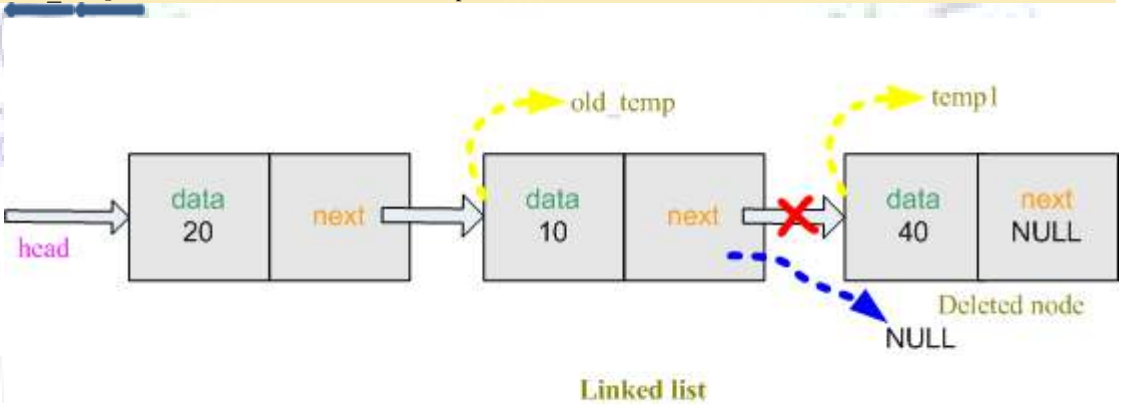


Figure: Delete at Start last

7) Delete specified number of node

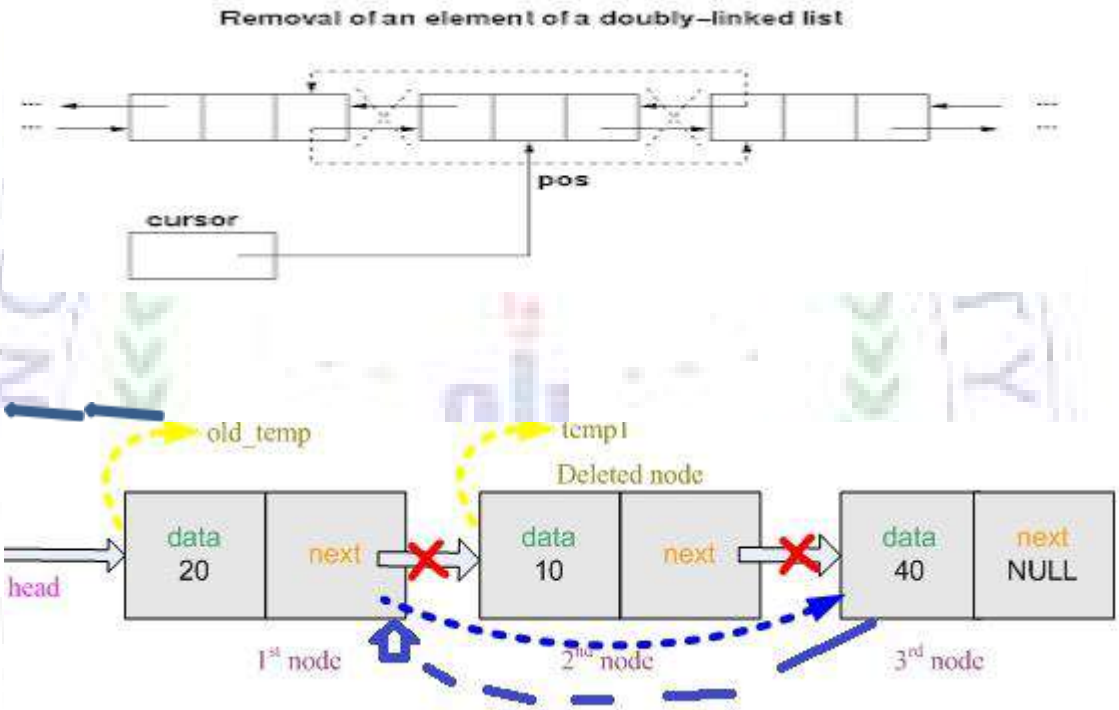
To delete a specified node in the linked list, we also require to search the specified node and previous node of the specified node. Create temporary `node temp1`, `old_temp` and allocate space for it. Take the input from user to know the number of the node.

```
node temp1;                                  // create a temporary node
temp1 = new node(); // allocate space for node
temp1 = head;                                // transfer the address of 'head' to 'temp1'

node old_temp;                               // create a temporary node
old_temp = new node();       // allocate space for node
old_temp = temp1;            // transfer the address of 'temp1' to 'old_temp'
Syste.out.println("ENTER THE NODE NUMBER:");
System.out.print(node_number);                                  // take location
```

```
for( int i = 1 ; i < node_number ; i++ )
{
        old_temp = temp1;                          // store previous node
        temp1 = temp1.next;                        // store current node

}
```

Now `node temp1` is now pointed at the specified node and `old_temp` is pointed at the previous node of the specified node. The previous node of the specified node must connect to the rest of the linked list so we transfer the address of tmp.prev.next= tmp.next; tmp.next.prev=tmp.prev. Now free the space allocated for the specified node.

```
tmp.prev.next= tmp.next;
 tmp.next.prev=tmp.prev.
```



Conclusion

From the above discussions, I hope that everybody understands what linked list is and how we can create it.