LJ University
University with a Difference

# DATA STRUCTURE USING JAVA

## LECTURE NOTES FOR UNIT - 6

# UNIT – 6
# Singly Linked List

PREPARED BY:
MR. VISMAY SHAH

## Contents

## 1. What is Dynamic Memory Allocation?

- ✓ Resources are always a premium.
- ✓ We have strived to achieve better utilization of resources at all times; that is the premise of our progress.
- ✓ Related to this pursuit, is the concept of memory allocation.
- ✓ Memory has to be allocated to the variables that we create, so that actual variables can be brought to existence.
- ✓ Now there is a constraint as how we think it happens, and how it actually happens.

## 2. How Computer Creates A Variable?

- When we think of creating something, we think of creating something from the very scratch, while this isn't what actually happens when a computer creates a variable 'X'; to the computer, is more like an allocation, the computer just assigns a memory cell from a lot of pre-existing memory cells to X.
- It's like someone named 'RAJESH' being allocated to a hotel room from a lot of free or empty pre-existing rooms. This example probably made it very clear as how the computer does the allocation of memory.

## 3. What is Static Memory Allocation?

- When we declare variables, we actually are preparing all the variables that will be used, so that the compiler knows that the variable being used is actually an important part of the program that the user wants and not just a rogue symbol floating around.
- So, when we declare variables, what the compiler actually does is allocate those variables to their rooms (refer to the hotel analogy earlier). Now, if you see, this is being done before the program executes, you can't allocate variables by this method while the program is executing.

```
// All the variables in below program are statically allocated.
void fun()
{
  int a;
}
int main()
{
  int b;
  int c[10]
}
```

Why do we need to introduce another allocation method if this just gets the job done? Why would we need to allocate memory while the program is executing?

- Because, even though it isn't blatantly visible, not being able to allocate memory during run time precludes flexibility and compromises with space efficiency. Specially, those cases where the input isn't known beforehand, we suffer in terms of inefficient storage use and lack or excess of slots to enter data (given an array or similar data structures to store entries).
- So, here we define Dynamic Memory Allocation: The mechanism by which storage/memory/cells can be allocated to variables during the run time is called Dynamic Memory Allocation. So, as we have been going through it all, we can tell that it allocates the memory during the run time which enables us to use as much storage as we want, without worrying about any wastage

**Dynamic memory allocation is the process of assigning the memory space during the execution time or the run time.**

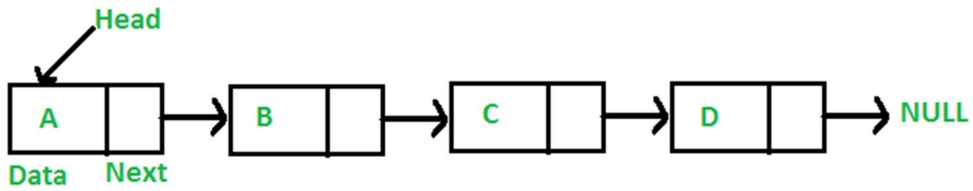
## 4. Reasons and Advantages of Allocating Memory Dynamically

**Reasons and Advantage of allocating memory dynamically:**

- ✓When we do not know how much amount of memory would be needed for the program beforehand.
- ✓When we want data structures without any upper limit of memory space.
- ✓When you want to use your memory space more efficiently. Example: If you have allocated memory space for a 1D array as array[20] and you end up using only 10 memory spaces then the remaining 10 memory spaces would be wasted and this wasted memory cannot even be utilized by other program variables.
- ✓Dynamically created lists insertions and deletions can be done very easily just by the manipulation of addresses whereas in case of statically allocated memory insertions and deletions lead to more movements and wastage of memory.
- ✓When you want you to use the concept of Objects and linked list in programming, dynamic memory allocation is a must.
- ✓There are two types of available memories- stack and heap.
- ✓Static memory allocation can only be done on stack
- ✓whereas dynamic memory allocation can be done on both stack and heap.
- ✓An example of dynamic allocation to be done on the stack is recursion where the functions are put into call stack in order of their occurrence and popped off one by one on reaching the base case.
- ✓Example of dynamic memory allocation on the heap is:
- ✓While allocating memory on heap we need to delete the memory manually as memory is not freed(deallocated) by the compiler itself even if the scope of allocated memory finishes(as in case of stack).

## 5. Linked List

- Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.




### a. Why Linked List?

Arrays can be used to store linear data of similar types, but arrays have the following limitations.

1. The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.
2. Inserting a new element in an array of elements is expensive because the room has to be created for the new elements and to create room existing elements have to be shifted but in Linked list if we have the head node then we can traverse to any node through it and insert new node at the required position.

For example, in a system, if we maintain a sorted list of IDs in an array id[].

id[] = [1000, 1010, 1050, 2000, 2040].

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

3) Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved due to this so much work is being done which affects the efficiency of the code.

## b. Advantages of linked list over arrays

- Dynamic size
- Ease of insertion/deletion

## c. Drawbacks of linked list

- Random access is not allowed. We have to access elements sequentially starting from the first node (head node). So, we cannot do binary search with linked lists efficiently with its default implementation. Read about it here.
- Extra memory space for a pointer is required with each element of the list.
- Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

## d. Representation of Linked List

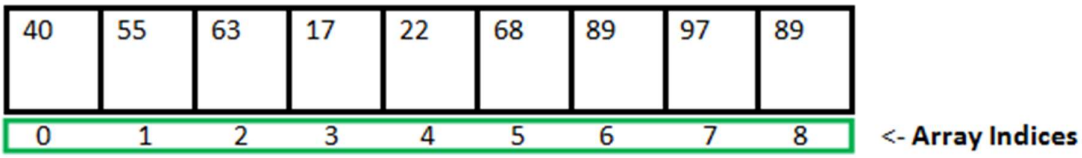
A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head points to NULL.

Each node in a list consists of at least two parts:

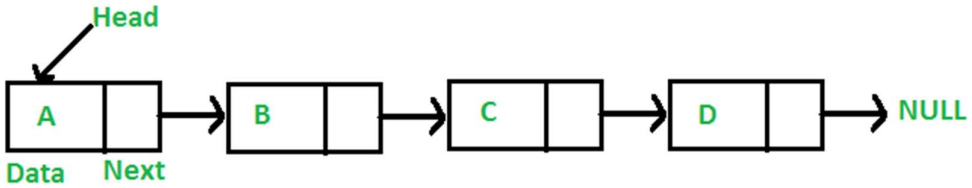
- Data (we can store integer, strings or any type of data).
- Reference to the next node (connects one node to another)

## 6. Linked List vs Array

Arrays store elements in contiguous memory locations, resulting in easily calculable addresses for the elements stored and this allows faster access to an element at a specific index. Linked lists are less rigid in their storage structure and elements are usually not stored in contiguous locations, hence they need to be stored with additional tags giving a reference to the next element. This difference in the data storage scheme decides which data structure would be more suitable for a given situation.




**Array Length = 9**
**First Index = 0**
**Last Index = 8**



## a. Major Differences between Array & Linked List

**Size:** Since data can only be stored in contiguous blocks of memory in an array, its size cannot be altered at runtime due to the risk of overwriting other data. However, in a linked list, each node points to the next one such that data can exist at scattered (non-contiguous) addresses; this allows for a dynamic size that can change at runtime.

**Memory allocation:** For arrays at compile time and at runtime for linked lists. but, a dynamically allocated array also allocates memory at runtime.

**Memory efficiency:** For the same number of elements, linked lists use more memory as a reference to the next node is also stored along with the data. However, size flexibility in linked lists may make them use less memory overall; this is useful when there is uncertainty about size or there are large variations in the size of data elements; memory equivalent to the upper limit on the size has to be allocated (even if not all of it is being used) while using arrays, whereas linked lists can increase their sizes step-by-step proportionately to the amount of data.

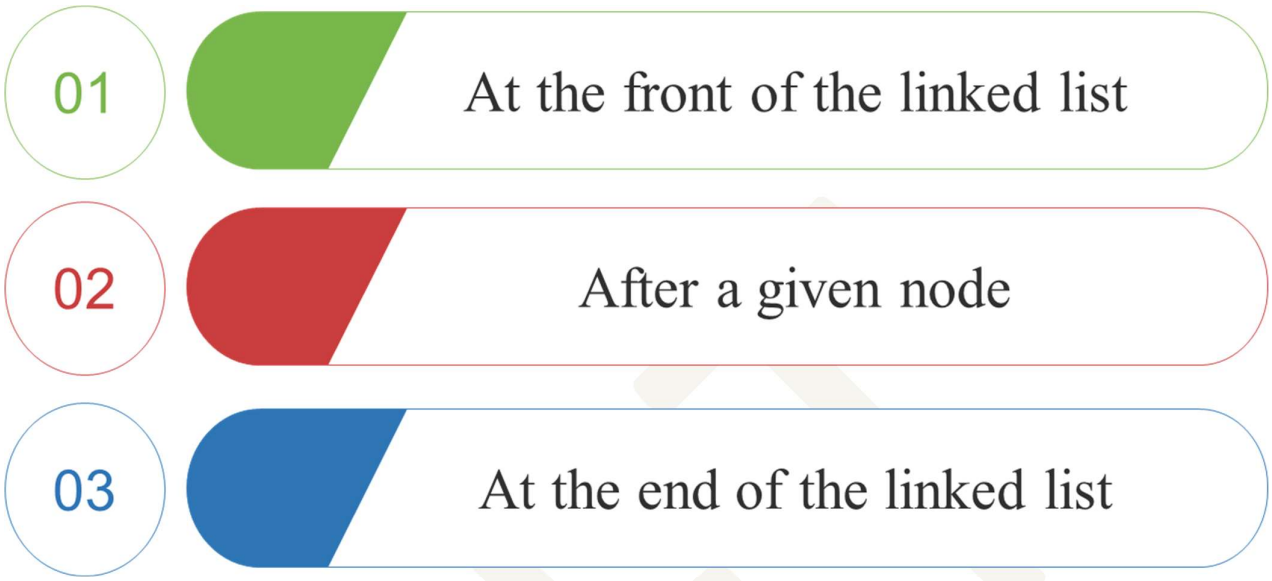
**Execution time:** Any element in an array can be directly accessed with its index; however in the case of a linked list, all the previous elements must be traversed to reach any element. Also, better cache locality in arrays (due to contiguous memory allocation) can significantly improve performance. As a result, some operations (such as modifying a certain element) are faster in arrays, while some others (such as inserting/deleting an element in the data) are faster in linked lists.

**Insertion:** In array insertion operation takes more time but in linked last these operations are fast. For example, if we want to insert an element in array at end position in array and array is full then we copy the array into another array and then we can add a element whereas if linked list is full then we find the last node and make it next to the new node.

| ARRAY | LINKED LISTS |
|---|---|
| 1. Arrays are stored in contiguous location. | 1. Linked lists are not stored in contiguous location. |
| 2. Fixed in size. | 2. Dynamic in size. |
| 3. Memory is allocated at compile time. | 3. Memory is allocated at run time. |
| 4. Uses less memory than linked lists. | 4. Uses more memory because it stores both data and the address of next node. |
| 5. Elements can be accessed easily. | 5. Element accessing requires the traversal of whole linked list. |
| 6. Insertion and deletion operation takes time. | 6. Insertion and deletion operation is faster. |

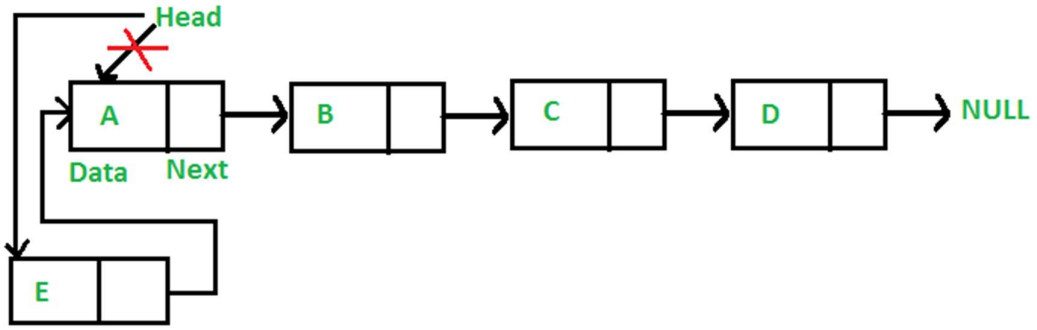
## 7. Inserting A Node in Linked List

A node can be added in three ways:



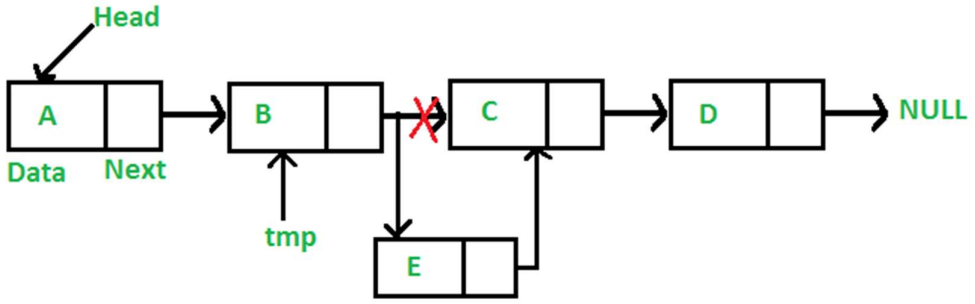
## a. Add A Node At the Front End (4 Steps Process)

The new node is always added before the head of the given Linked List. And newly added node becomes the new head of the Linked List. For example, if the given Linked List is 10->15->20->25 and we add an item 5 at the front, then the Linked List becomes 5->10->15->20->25. Let us call the function that adds at the front of the list is push(). The push() must receive a pointer to the head pointer, because push must change the head pointer to point to the new node. (See this)



## b. Add A Node After A Given Node (5 Steps Process)

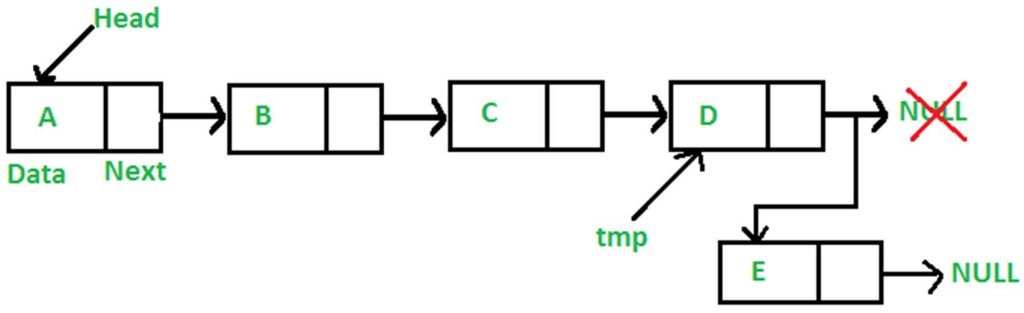
We are given a pointer to a node, and the new node is inserted after the given node.




## c. Add A Node At The End of Linked List (6 Step Process)

The new node is always added after the last node of the given Linked List. For example if the given Linked List is 5->10->15->20->25 and we add an item 30 at the end, then the Linked List becomes 5->10->15->20->25->30.

Since a Linked List is typically represented by the head of it, we have to traverse the list till the end and then change the next to last node to a new node.



## 8. Deleting a Node in A Linked List

We have discussed Linked List Introduction and Linked List Insertion in previous posts on a singly linked list.

Let us formulate the problem statement to understand the deletion process. Given a 'key', delete the first occurrence of this key in the linked list.

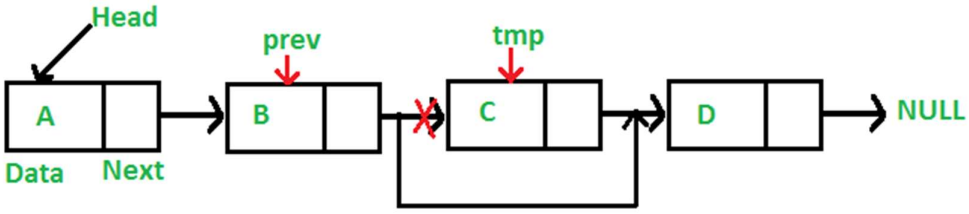
**Iterative Method:**

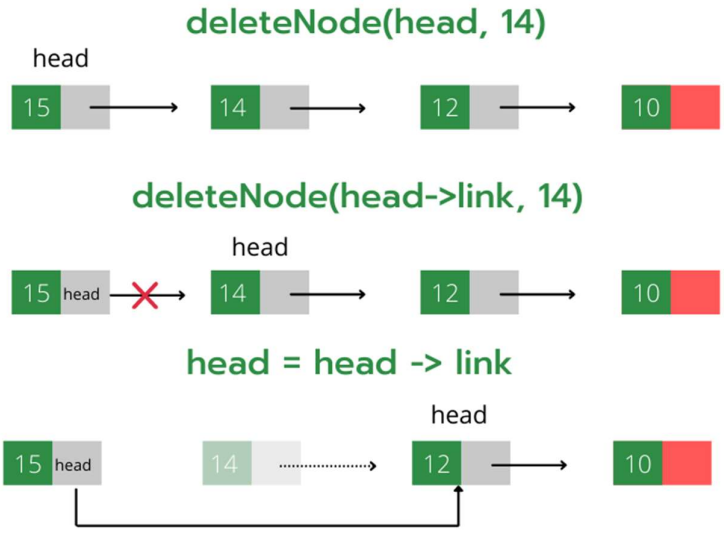
To delete a node from the linked list, we need to do the following steps.

1) Find the previous node of the node to be deleted.

2) Change the next of the previous node.

3) Free memory for the node to be deleted.



## a. Delete A Node At a Given Position in a Linked List

Given a singly linked list and a position, delete a linked list node at the given position.

Example:

Input: position = 1, Linked List = 8->2->3->1->7

Output: Linked List = 8->3->1->7

Input: position = 0, Linked List = 8->2->3->1->7

Output: Linked List = 2->3->1->7

If the node to be deleted is the root, simply delete it. To delete a middle node, we must have a pointer to the node previous to the node to be deleted. So if positions are not zero, we run a loop position-1 times and get a pointer to the previous node

# 9. Write a Java Program to implement All the Operation of insert, delete and display for a Singly Linked List.

```
import java.util.*;
class SLL //operation class//
{
    class Node //created a nested class//
    {
        int data;
        Node next; //created a reference variable//
        Node (int data) //created a parameterized constructor//
            {
            this.data=data;
            next =null; //set the initial value of reference variable as null//
            }
    }
    Node First = null; //created another reference variable First with value null//
    void addFirst(int data) //created a method to insert the data at First in Linked
    List//
    {
        Node n = new Node(data); //created a node n//
        if (First == null)
        {
            First = n;
        }
        else
        {
        n.next=First; //the value for First shall be given to n's next//
        First=n; //now the new value of First shall be new node n's address//
        }
    }
    void display() //created a method to display the elements of Linked List//
    {
        Node temp = First; //created another reference variable temp which will have the
        same value as first//
        while(temp!=null) //until we get the temp = null this loop will run//
        {
        System.out.print(temp.data + "-");
        temp=temp.next;
        }
        System.out.println("NULL"); //every linked list has the last reference as null//
    }
    void addLast(int data) //created a method to insert the data at last in Linkedlist//
    {
        Node n=new Node(data); //created a node n//
        if (First == null) //checking if the value of first is null or not//
        {
            First = n; //then the value of new node n shall be given to First//
        }
        else
        {
            Node temp=First;
            while (temp.next!=null) //untill temp's next is not null//
            {
                temp=temp.next;
            }
            temp.next=n; //once temp's next is null then new node n's value assigned to
            temp's next//
        }
    }
    Node deleteFirst() //created a method to delete the element from the start of the
    linked list//
    {
        if(First == null) //checking for underflow condition//
        {
            System.out.println("Underflow");
            return null;
        }
        else
        {
```

```
64              Node del = First;
65              First=First.next;
66              return del;
67          }
68      }
69      void deleteLast() //created a method to delete the element of a linked list from
        last//
70      {
71          if(First == null) //checking if the linked list is empty or not//
72          {
73              System.out.println("No Elements in LinkedList");
74          }
75          else
76          {
77              Node del=First;
78              while(del.next.next != null)
79              {
80                  del = del.next;
81              }
82              del.next=null;
83          }
84      }
85      void insertBeforeValue(int data,int value) //created a method to add the value
        before a particular value in the linked list//
86      {
87
88          int flag = 0;
89          if(First==null)
90          {
91              System.out.println("Linked is empty");
92          }
93          else
94          {
95              Node dummy = First;
96              while(dummy != null)
97              {
98                  if(dummy.data == value)
99                  {
100                 flag = 1;
101                 }
102                 dummy = dummy.next;
103             }
104                 if(flag == 0)
105                 {
106                     System.out.println("The asked value is not inside the linked list");
107                 }
108                 else
109                 {
110                     Node n = new Node(data);
111                     if(First.next== null)
112                     {
113                         System.out.println("It will be inserted at first");
114                         addFirst(data);
115                     }
116                     else if(First.data == value)
117                     {
118                         addFirst(data);
119                     }
120                     else
121                     {
122                         Node temp=First;
123                         while(temp.next.data!=value)
124                         {
125                             temp=temp.next;
126                         }
127                         n.next=temp.next;
128                         temp.next=n;
```

```
                    }
                }
            }
        }
        void insertAfterValue(int data,int value) //created a method to add the value after
        a particular value in the linked list//
        {

            int flag = 0;
            if(First==null)
            {
                System.out.println("Linked is empty");
            }
            else
            {
                Node dummy = First;
                while(dummy != null)
                {
                    if(dummy.data == value)
                    {
                    flag = 1;
                    }
                    dummy = dummy.next;
                }
                    if(flag == 0)
                    {
                        System.out.println("The asked value is not inside the linked list");
                    }
                    else
                    {
                        Node n = new Node(data);
                        if(First.data == value && First.next== null)
                        {
                            First.next=n;
                        }
                        else if(First.data == value)
                        {
                            n.next=First.next;
                            First.next=n;
                        }
                        else
                        {
                            Node temp=First;
                            while(temp.data!=value)
                            {
                                temp=temp.next;
                            }
                            n.next=temp.next;
                            temp.next=n;
                        }
                    }
            }
        }
        void deleteValue(int value) //created a method to delete the particular value from
        the linked list//
        {
            int flag = 0;
            if(First==null)
            {
                System.out.println("Linked is empty");
            }
            else
            {
                Node dummy = First;
                while(dummy != null) //checking that the value we want to delete is already
                a part of linked list or not//
                {
```

```
                    if(dummy.data == value)
                    {
                        flag = 1;
                    }
                    dummy = dummy.next;
                }
                if(flag == 0)
                {
                    System.out.println("The asked value is not inside the linked list");
                }
                else
                {
                    if(First.data == value && First.next==null)
                    {
                        First = null;
                    }
                    else if(First.data == value)
                    {
                            First = First.next;
                    }
                    else
                    {
                        Node temp=First;
                        while(temp.next.data!=value)
                        {
                            temp = temp.next;
                        }
                        Node q= temp.next;
                        temp.next=q.next;
                        q = null;
                    }
                }
            }
        }
    }
    class Run //created the run class to call the method by making objects//
    {
        public static void main(String args[])
        {
            SLL s = new SLL();
            s.addFirst(10);
            s.display();
            s.addFirst(5);
            s.display();
            s.addFirst(11);
            s.display();
            s.addLast(15);
            s.display();
            s.deleteFirst();
            s.display();
            s.deleteLast();
            s.display();
            s.insertBeforeValue(100,5);
            s.display();
            s.insertAfterValue(200,10);
            s.display();
            s.deleteValue(5);
            s.display();
            s.deleteValue(100);
            s.display();
        }
    }
```