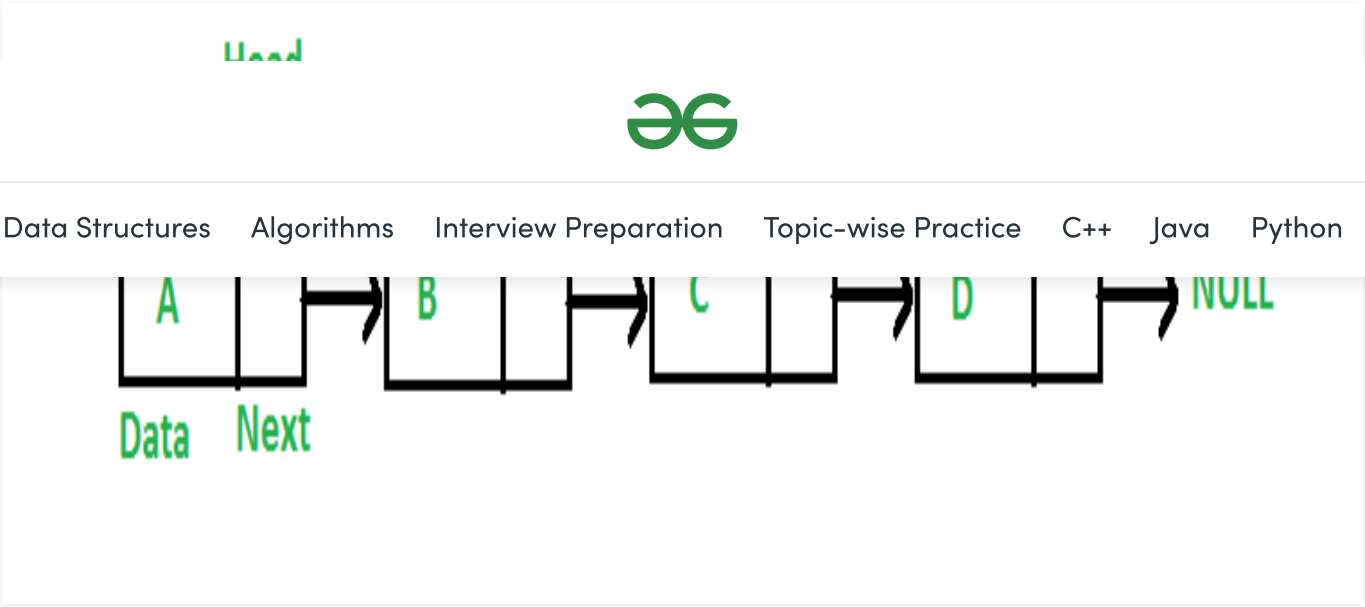
# Implementing a Linked List in Java using Class

Difficulty Level : Easy • Last Updated : 19 Feb, 2021

**Pre-requisite:** Linked List Data Structure

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




In Java, LinkedList can be represented as a class and a Node as a separate class. The LinkedList class contains a reference of Node class type.

## Java

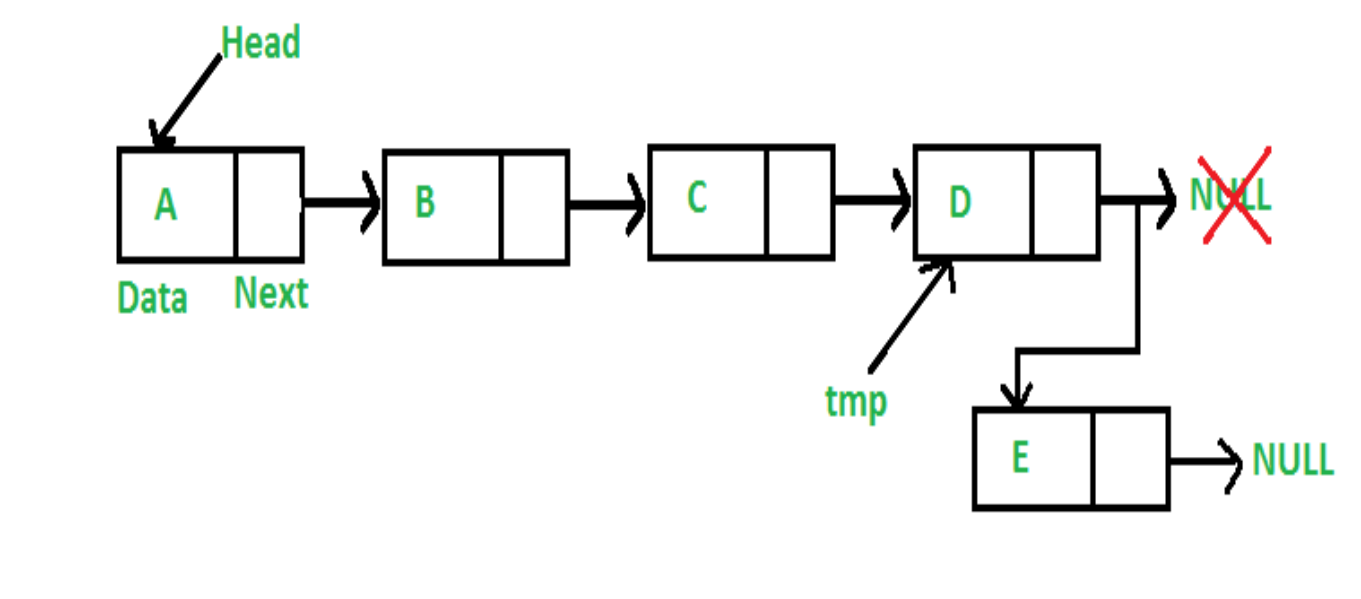
```
lass LinkedList {
```

```
        // as null
        Node(int d) { data = d; }
    }
}
```

### Creation and Insertion

In this article, insertion in the list is done at the end, that is the new node is added after the last node of the given Linked List. For example, if the given Linked List is 5->10->15->20->25 and 30 is to be inserted, then the Linked List becomes 5->10->15->20->25->30.

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




## Java

```
// so that main() can access it
static class Node {

    int data;
    Node next;

    // Constructor
    Node(int d)
    {
        data = d;
        next = null;
    }
}

// Method to insert a new node
public static LinkedList insert(LinkedList list, int data)
{
    // Create a new node with given data
    Node new_node = new Node(data);
    new_node.next = null;

    // If the Linked List is empty,
    // then make the new node as head
    if (list.head == null) {
        list.head = new_node;
    }
    else {
        // Else traverse till the last node
        // and insert the new_node there
        Node last = list.head;
        while (last.next != null) {
            last = last.next;
        }

        // Insert the new_node at last node
        last.next = new_node;
    }

    // Return the list by head
    return list;
}
```

```
            // Print the data at current node
            System.out.print(currNode.data + " ");

            // Go to next node
            currNode = currNode.next;
        }
    }

    // Driver code
    public static void main(String[] args)
    {
        /* Start with the empty list. */
        LinkedList list = new LinkedList();

        //
        // ******INSERTION******
        //

        // Insert the values
        list = insert(list, 1);
        list = insert(list, 2);
        list = insert(list, 3);
        list = insert(list, 4);
        list = insert(list, 5);
        list = insert(list, 6);
        list = insert(list, 7);
        list = insert(list, 8);

        // Print the LinkedList
        printList(list);
    }
}
```

### Output

```
LinkedList: 1 2 3 4 5 6 7 8
```

### Traversal

```
// a Singly Linked List
public class LinkedList {

    Node head; // head of list

    // Linked list Node.
    // Node is a static nested class
    // so main() can access it
    static class Node {

        int data;
        Node next;

        // Constructor
        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    // Method to insert a new node
    public static LinkedList insert(LinkedList list,
                                    int data)
    {
        // Create a new node with given data
        Node new_node = new Node(data);
        new_node.next = null;

        // If the Linked List is empty,
        // then make the new node as head
        if (list.head == null) {
            list.head = new_node;
        }
        else {
            // Else traverse till the last node
            // and insert the new_node there
            Node last = list.head;
            while (last.next != null) {
                last = last.next;
            }
```

```
// Method to print the LinkedList.
public static void printList(LinkedList list)
{
    Node currNode = list.head;

    System.out.print("LinkedList: ");

    // Traverse through the LinkedList
    while (currNode != null) {
        // Print the data at current node
        System.out.print(currNode.data + " ");

        // Go to next node
        currNode = currNode.next;
    }
}

// **************MAIN METHOD**************

// method to create a Singly linked list with n nodes
public static void main(String[] args)
{
    /* Start with the empty list. */
    LinkedList list = new LinkedList();

    //
    // ******INSERTION******
    //

    // Insert the values
    list = insert(list, 1);
    list = insert(list, 2);
    list = insert(list, 3);
    list = insert(list, 4);
    list = insert(list, 5);
    list = insert(list, 6);
    list = insert(list, 7);
    list = insert(list, 8);

    // Print the LinkedList
    printList(list);
}
```

The deletion process can be understood as follows:

**To be done:**

*Given a 'key', delete the first occurrence of this key in the linked list.*

**How to do it:**

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

1. Search the key for its first occurrence in the list
2. Now, Any of the 3 conditions can be there:
   - **Case 1: The key is found at the head**
     1. In this case, Change the head of the node to the next node of the current head.
     2. Free the memory of the replaced head node.
   - **Case 2: The key is found in the middle or last, except at the head**
     1. In this case, Find the previous node of the node to be deleted.
     2. Change the next the previous node to the next node of the current node.
     3. Free the memory of the replaced node.
   - **Case 3: The key is not found in the list**
     1. In this case, No operation needs to be done.


## Java

```java
import java.io.*;

// Java program to implement
// a Singly Linked List
public class LinkedList {

    Node head; // head of list

    // Linked list Node.
    // Node is a static nested class
    // so main() can access it
    static class Node {

        int data;
        Node next;

        // Constructor
        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    // Method to insert a new node
```

```
        if (list.head == null) {
            list.head = new_node;
        }
        else {
            // Else traverse till the last node
            // and insert the new_node there
            Node last = list.head;
            while (last.next != null) {
                last = last.next;
            }

            // Insert the new_node at last node
            last.next = new_node;
        }

        // Return the list by head
        return list;
    }

    // Method to print the LinkedList.
    public static void printList(LinkedList list)
    {
        Node currNode = list.head;

        System.out.print("LinkedList: ");

        // Traverse through the LinkedList
        while (currNode != null) {
            // Print the data at current node
            System.out.print(currNode.data + " ");

            // Go to next node
            currNode = currNode.next;
        }

        System.out.println();
    }

    // **************DELETION BY KEY**************

    // Method to delete a node in the LinkedList by KEY
    public static LinkedList deleteByKey(LinkedList list
```

```
if (currNode != null && currNode.data == key) {
    list.head = currNode.next; // Changed head

    // Display the message
    System.out.println(key + " found and deleted");

    // Return the updated List
    return list;
}

//
// CASE 2:
// If the key is somewhere other than at head
//

// Search for the key to be deleted,
// keep track of the previous node
// as it is needed to change currNode.next
while (currNode != null && currNode.data != key) {
    // If currNode does not hold key
    // continue to next node
    prev = currNode;
    currNode = currNode.next;
}

// If the key was present, it should be at currNode
// Therefore the currNode shall not be null
if (currNode != null) {
    // Since the key is at currNode
    // Unlink currNode from linked list
    prev.next = currNode.next;

    // Display the message
    System.out.println(key + " found and deleted");
}

//
// CASE 3: The key is not present
//

// If key was not present in linked list
// currNode should be null
```

```
// ^^^^^^^^^^^^^^MAIN METHOD^^^^^^^^^^^^^^

// method to create a Singly linked list with n nodes
public static void main(String[] args)
{
    /* Start with the empty list. */
    LinkedList list = new LinkedList();

    //
    // ******INSERTION******
    //

    // Insert the values
    list = insert(list, 1);
    list = insert(list, 2);
    list = insert(list, 3);
    list = insert(list, 4);
    list = insert(list, 5);
    list = insert(list, 6);
    list = insert(list, 7);
    list = insert(list, 8);

    // Print the LinkedList
    printList(list);

    //
    // ******DELETION BY KEY******
    //

    // Delete node with value 1
    // In this case the key is ***at head***
    deleteByKey(list, 1);

    // Print the LinkedList
    printList(list);

    // Delete node with value 4
    // In this case the key is present ***in the
    // middle***
    deleteByKey(list, 4);

    // Print the LinkedList
```

```
}
```

**Output**

```
LinkedList: 1 2 3 4 5 6 7 8
1 found and deleted
LinkedList: 2 3 4 5 6 7 8
4 found and deleted
LinkedList: 2 3 5 6 7 8
10 not found
LinkedList: 2 3 5 6 7 8
```

**Deletion At Position**

This deletion process can be understood as follows:

**To be done:**

*Given a **'position'**, delete the node at this position from the linked list.*

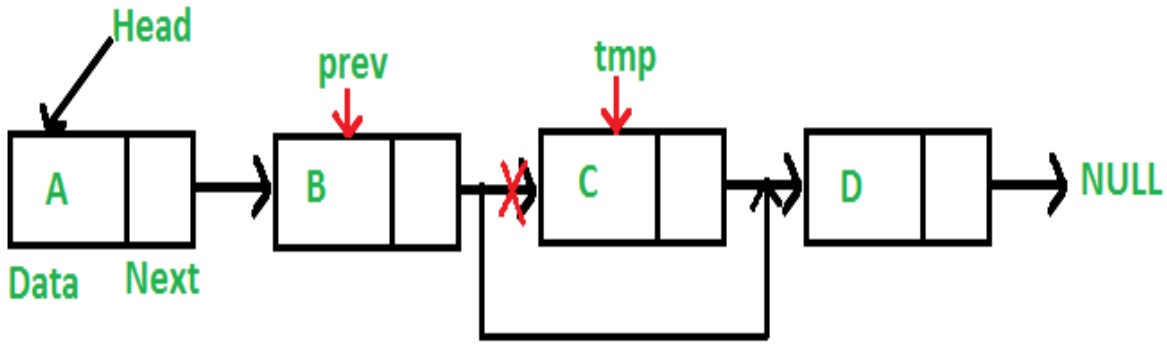
**How to do it:**

The steps to do it are as follows:

1. Traverse the list by counting the index of the nodes
2. For each index, match the index to be same as position
3. Now, Any of the 3 conditions can be there:
   - **Case 1: The position is 0, i.e. the head is to be deleted**
     1. In this case, Change the head of the node to the next node of current head.
     2. Free the memory of replaced head node.
   - **Case 2: The position is greater than 0 but less than the size of the list, i.e. in the middle or last, except at head**

the list

1. In this case, No operation needs to be done.




## Java

```
import java.io.*;

// Java program to implement
// a Singly Linked List
public class LinkedList {

    Node head; // head of list

    // Linked list Node.
    // Node is a static nested class
    // so main() can access it
    static class Node {
```

```
    }
}

// Method to insert a new node
public static LinkedList insert(LinkedList list,
                                int data)
{
    // Create a new node with given data
    Node new_node = new Node(data);
    new_node.next = null;

    // If the Linked List is empty,
    // then make the new node as head
    if (list.head == null) {
        list.head = new_node;
    }
    else {
        // Else traverse till the last node
        // and insert the new_node there
        Node last = list.head;
        while (last.next != null) {
            last = last.next;
        }

        // Insert the new_node at last node
        last.next = new_node;
    }

    // Return the list by head
    return list;
}

// Method to print the LinkedList.
public static void printList(LinkedList list)
{
    Node currNode = list.head;

    System.out.print("LinkedList: ");

    // Traverse through the LinkedList
    while (currNode != null) {
        // Print the data at current node
```

```
// Method to delete a node in the LinkedList by POSITION
public static LinkedList
deleteAtPosition(LinkedList list, int index)
{
    // Store head node
    Node currNode = list.head, prev = null;

    //
    // CASE 1:
    // If index is 0, then head node itself is to be
    // deleted

    if (index == 0 && currNode != null) {
        list.head = currNode.next; // Changed head

        // Display the message
        System.out.println(
            index + " position element deleted");

        // Return the updated List
        return list;
    }

    //
    // CASE 2:
    // If the index is greater than 0 but less than the
    // size of LinkedList
    //
    // The counter
    int counter = 0;

    // Count for the index to be deleted,
    // keep track of the previous node
    // as it is needed to change currNode.next
    while (currNode != null) {

        if (counter == index) {
            // Since the currNode is the required
            // position Unlink currNode from linked list
            prev.next = currNode.next;

            // Display the message
```

```
                counter++;
            }
        }

        // If the position element was found, it should be
        // at currNode Therefore the currNode shall not be
        // null
        //
        // CASE 3: The index is greater than the size of the
        // LinkedList
        //
        // In this case, the currNode should be null
        if (currNode == null) {
            // Display the message
            System.out.println(
                index + " position element not found");
        }

        // return the List
        return list;
    }

    // **************MAIN METHOD**************

    // method to create a Singly linked list with n nodes
    public static void main(String[] args)
    {
        /* Start with the empty list. */
        LinkedList list = new LinkedList();

        //
        // ******INSERTION******
        //

        // Insert the values
        list = insert(list, 1);
        list = insert(list, 2);
        list = insert(list, 3);
        list = insert(list, 4);
        list = insert(list, 5);
        list = insert(list, 6);
        list = insert(list, 7);
```

```
        // Delete node at position 0
        // In this case the key is ***at head***
        deleteAtPosition(list, 0);

        // Print the LinkedList
        printList(list);

        // Delete node at position 2
        // In this case the key is present ***in the
        // middle***
        deleteAtPosition(list, 2);

        // Print the LinkedList
        printList(list);

        // Delete node at position 10
        // In this case the key is ***not present***
        deleteAtPosition(list, 10);

        // Print the LinkedList
        printList(list);
    }
}
```

### Output

```
LinkedList: 1 2 3 4 5 6 7 8
0 position element deleted
LinkedList: 2 3 4 5 6 7 8
2 position element deleted
LinkedList: 2 3 5 6 7 8
10 position element not found
LinkedList: 2 3 5 6 7 8
```

### All Operations

Now is the complete program that applies each operation together:

```
Node head; // head of list

// Linked list Node.
// Node is a static nested class
// so main() can access it
static class Node {

    int data;
    Node next;

    // Constructor
    Node(int d)
    {
        data = d;
        next = null;
    }
}

// **************INSERTION**************

// Method to insert a new node
public static LinkedList insert(LinkedList list,
                                int data)
{
    // Create a new node with given data
    Node new_node = new Node(data);
    new_node.next = null;

    // If the Linked List is empty,
    // then make the new node as head
    if (list.head == null) {
        list.head = new_node;
    }
    else {
        // Else traverse till the last node
        // and insert the new_node there
        Node last = list.head;
        while (last.next != null) {
            last = last.next;
        }
```

```
// Method to print the LinkedList.
public static void printList(LinkedList list)
{
    Node currNode = list.head;

    System.out.print("\nLinkedList: ");

    // Traverse through the LinkedList
    while (currNode != null) {
        // Print the data at current node
        System.out.print(currNode.data + " ");

        // Go to next node
        currNode = currNode.next;
    }
    System.out.println("\n");
}

// **************DELETION BY KEY**************

// Method to delete a node in the LinkedList by KEY
public static LinkedList deleteByKey(LinkedList list,
                                     int key)
{
    // Store head node
    Node currNode = list.head, prev = null;

    //
    // CASE 1:
    // If head node itself holds the key to be deleted

    if (currNode != null && currNode.data == key) {
        list.head = currNode.next; // Changed head

        // Display the message
        System.out.println(key + " found and deleted");

        // Return the updated List
        return list;
    }

    //
```

```
            // continue to next node
            prev = currNode;
            currNode = currNode.next;
        }

        // If the key was present, it should be at currNode
        // Therefore the currNode shall not be null
        if (currNode != null) {
            // Since the key is at currNode
            // Unlink currNode from linked list
            prev.next = currNode.next;

            // Display the message
            System.out.println(key + " found and deleted");
        }

        //
        // CASE 3: The key is not present
        //

        // If key was not present in linked list
        // currNode should be null
        if (currNode == null) {
            // Display the message
            System.out.println(key + " not found");
        }

        // return the List
        return list;
    }

    // **************DELETION AT A POSITION**************

    // Method to delete a node in the LinkedList by POSITION
    public static LinkedList
    deleteAtPosition(LinkedList list, int index)
    {
        // Store head node
        Node currNode = list.head, prev = null;

        //
        // CASE 1:
```

```
        // Return the updated List
        return list;
    }

    //
    // CASE 2:
    // If the index is greater than 0 but less than the
    // size of LinkedList
    //
    // The counter
    int counter = 0;

    // Count for the index to be deleted,
    // keep track of the previous node
    // as it is needed to change currNode.next
    while (currNode != null) {

        if (counter == index) {
            // Since the currNode is the required
            // position Unlink currNode from linked list
            prev.next = currNode.next;

            // Display the message
            System.out.println(
                index + " position element deleted");
            break;
        }
        else {
            // If current position is not the index
            // continue to next node
            prev = currNode;
            currNode = currNode.next;
            counter++;
        }
    }

    // If the position element was found, it should be
    // at currNode Therefore the currNode shall not be
    // null
    //
    // CASE 3: The index is greater than the size of the
```

```
        // return the List
        return list;
    }

    // **************MAIN METHOD**************

    // method to create a Singly linked list with n nodes
    public static void main(String[] args)
    {
        /* Start with the empty list. */
        LinkedList list = new LinkedList();

        //
        // ******INSERTION******
        //

        // Insert the values
        list = insert(list, 1);
        list = insert(list, 2);
        list = insert(list, 3);
        list = insert(list, 4);
        list = insert(list, 5);
        list = insert(list, 6);
        list = insert(list, 7);
        list = insert(list, 8);

        // Print the LinkedList
        printList(list);

        //
        // ******DELETION BY KEY******
        //

        // Delete node with value 1
        // In this case the key is ***at head***
        deleteByKey(list, 1);

        // Print the LinkedList
        printList(list);

        // Delete node with value 4
        // In this case the key is present ***in the
```

```
        // Print the LinkedList
        printList(list);

        //
        // ******DELETION AT POSITION******
        //

        // Delete node at position 0
        // In this case the key is ***at head***
        deleteAtPosition(list, 0);

        // Print the LinkedList
        printList(list);

        // Delete node at position 2
        // In this case the key is present ***in the
        // middle***
        deleteAtPosition(list, 2);

        // Print the LinkedList
        printList(list);

        // Delete node at position 10
        // In this case the key is ***not present***
        deleteAtPosition(list, 10);

        // Print the LinkedList
        printList(list);
    }
}
```

**Output**

```
LinkedList: 1 2 3 4 5 6 7 8

1 found and deleted
```

```
LinkedList: 2 3 5 6 7 8

0 position element deleted

LinkedList: 3 5 6 7 8

2 position element deleted

LinkedList: 3 5 7 8

10 position element not found

LinkedList: 3 5 7 8
```

## Article Contributed By :

RishabhPrabhu
@RishabhPrabhu

## Vote for difficulty

Current difficulty : Easy

GeeksforGeeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Lifestyle

SQL

## Web Development

Web Tutorials
Django Tutorial

HTML

CSS

JavaScript

Bootstrap

## Contribute

Write an Article
Improve an Article

Pick Topics to Write

Write Interview Experience

Internships

Video Internship