Linked List

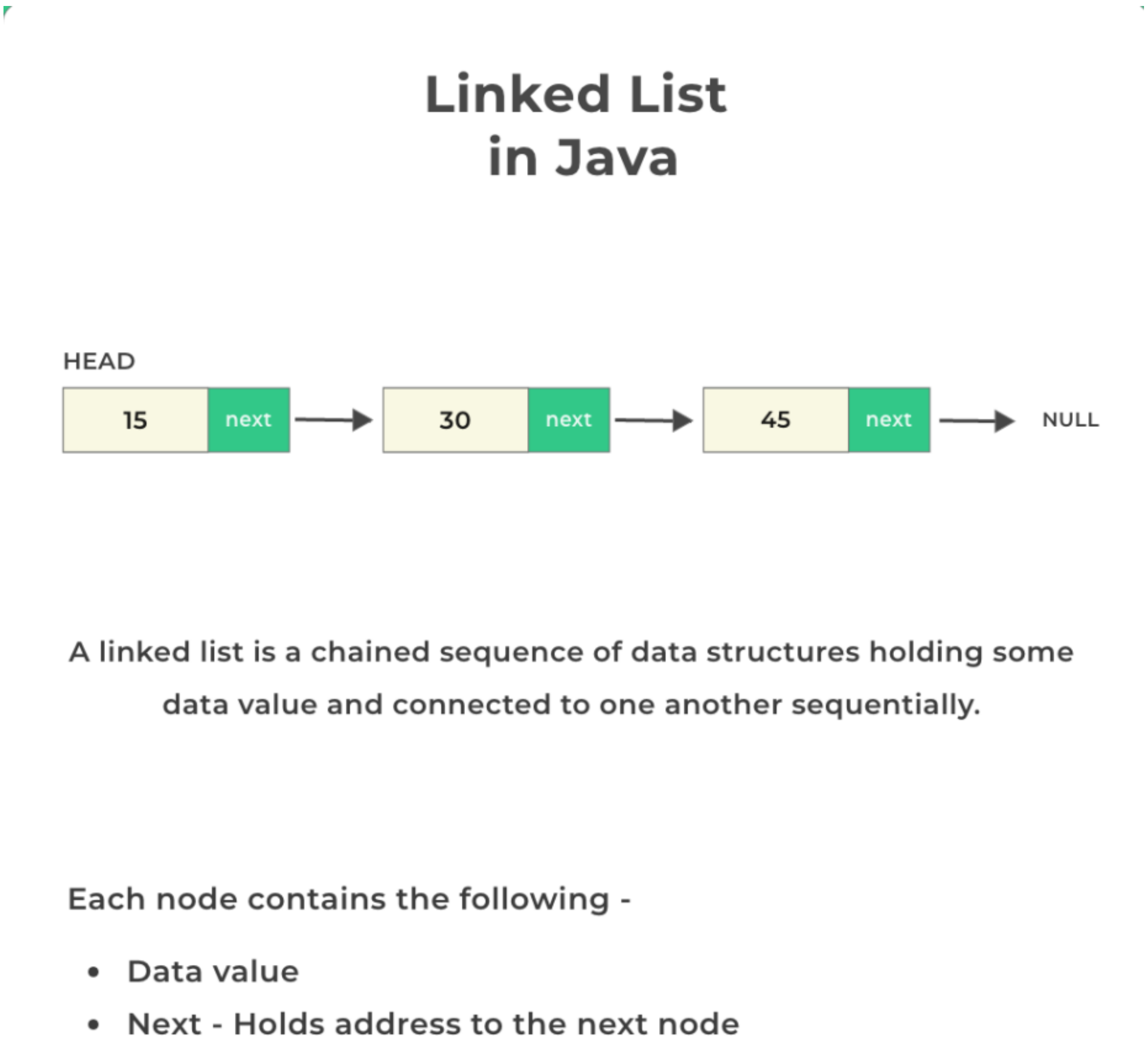
A linked list in Java is a linear data structure that we can use for storing a large amount of data with ease.

The data stored in Linked List is not in a contiguous manner, but each data is stored at a different location, which can be accessed according to one's need

Linked List is a preferred data structure over arrays, as inserting and deleting data in a linked list is easier than in an array.

A linked list is a linear data structure that is made up of several nodes, which is further divided into two parts-:

1. **Node –** This part stores the data.
2. **Link** – This part stores the address of the memory location, where the next data of the list is stored.



Linked List in Java

A linked list is a chained sequence of data structures holding some data value and connected to one another sequentially.

Each node contains the following -

- Data value
- Next - Holds address to the next node

Types of Linked Lists

There are many variations but below are the most important ones –

- Singly Linked List
- Doubly Linked List
- Circular Linked List

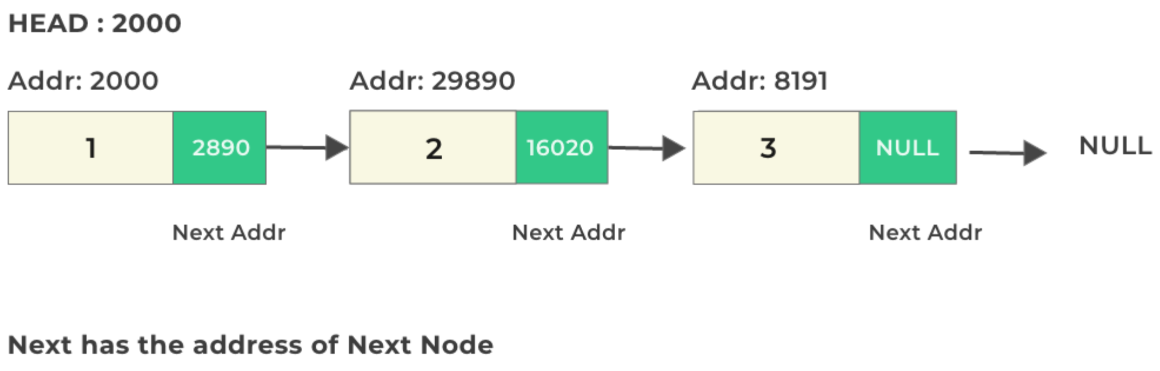
Singly Linked List

It is the most basic linked list, which is made up of several nodes, which can further be divided into two parts

- data
- next

Each node is connected to one another as the next value for each node holds the address to the next node in the sequence.

The first node is called as head and the last node is called as tail, where the last node's next value is null.

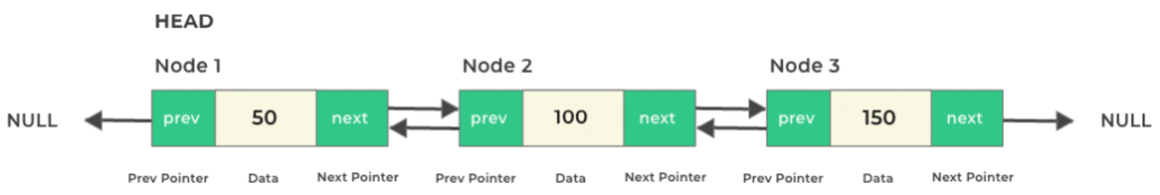


Doubly Linked List

This is a successor of a normal linked list. Nodes of a double linked list have three parts –

- Data – Data / Value held
- Next – Reference Address to the previous node
- Next – Reference Address to next node

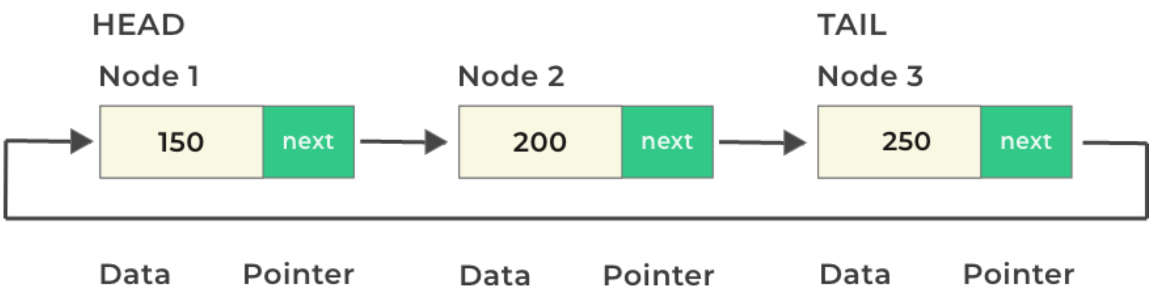
The main advantage is we can traverse in any direction forwards and backwards.



Circular Linked List

It is very similar to Singly Linked List with one change that rather than the last node pointing to null. It has the address to the first (head) node.

Which makes it circular in Nature, it is used in OS to implement round robin scheduling algorithms too.



Last Node points to Head Node

```
package linklist;

public class LinkedList1 {
  Node head;
  // not using parameterized constructor would by default
  // force head instance to become null
  // Node head = null; // can also do this, but not required
  // Node Class
  class Node {
    int data;
    Node next;
    Node(int x) // parameterized constructor
    {
      data = x;
      next = null;
    }
  }




  public Node insert(int data) {
    // Creating newNode memory & assigning data value
    Node newNode = new Node(data);
    // assigning this newNode's next as current head node
    newNode.next = head;

    // re-assigning head to this newNode
    head = newNode;

    return head;
  }

  public void display() {
    Node node = head;
    // as linked list will end when Node reaches Null
    while (node != null) {
      System.out.print(node.data + " ");
      node = node.next;
    }
    System.out.println();
  }
  public void delete() {
    if (head == null) {
      System.out.println("List is empty, not possible to delete");
      return;
    }

    System.out.println("Deleted: " + head.data);
    // move head to next node
    head = head.next;
  }
  public static void main(String args[]) {
    LinkedList1 ll = new LinkedList1();

    ll.insert(6);
    ll.insert(5);
    ll.insert(3);
    ll.insert(4);
    ll.insert(2);
    ll.insert(1);

    ll.display();

    ll.delete();
    ll.delete();

    ll.display();
  }
}
```

```
package linklist;

class Node {
  int data;
  Node next;

  Node(int x) // parameterized constructor
  {
    data = x;
    next = null;
  }
}

public class LinkedList2 {
  static Node insertStart(Node head, int data) {
    // Creating newNode memory & assigning data value
    Node newNode = new Node(data);

    // assigning this newNode's next as current head node
    newNode.next = head;
    // re-assigning head to this newNode
    head = newNode;

    return head;
  }
  public static Node delete(Node head) {
    if (head == null) {
      System.out.println("List is empty, not possible to delete");
      return head;
    }

    System.out.println("Deleted: " + head.data);
    // move head to next node
    head = head.next;

    return head;
  }

  static void display(Node node) {

    // as linked list will end when Node is Null
    while (node != null) {
      System.out.print(node.data + " ");
      node = node.next;
    }
    System.out.println("");
  }

  public static void main(String args[]) {
    Node head = null;
    head = insertStart(head, 6);
    head = insertStart(head, 5);
    head = insertStart(head, 4);
    head = insertStart(head, 3);
    head = insertStart(head, 2);
    head = insertStart(head, 1);

    display(head);

    head = delete(head);
    head = delete(head);

    display(head);

  }
}
```

Singly Linked List

Doubly Linked List

Circular Linked List