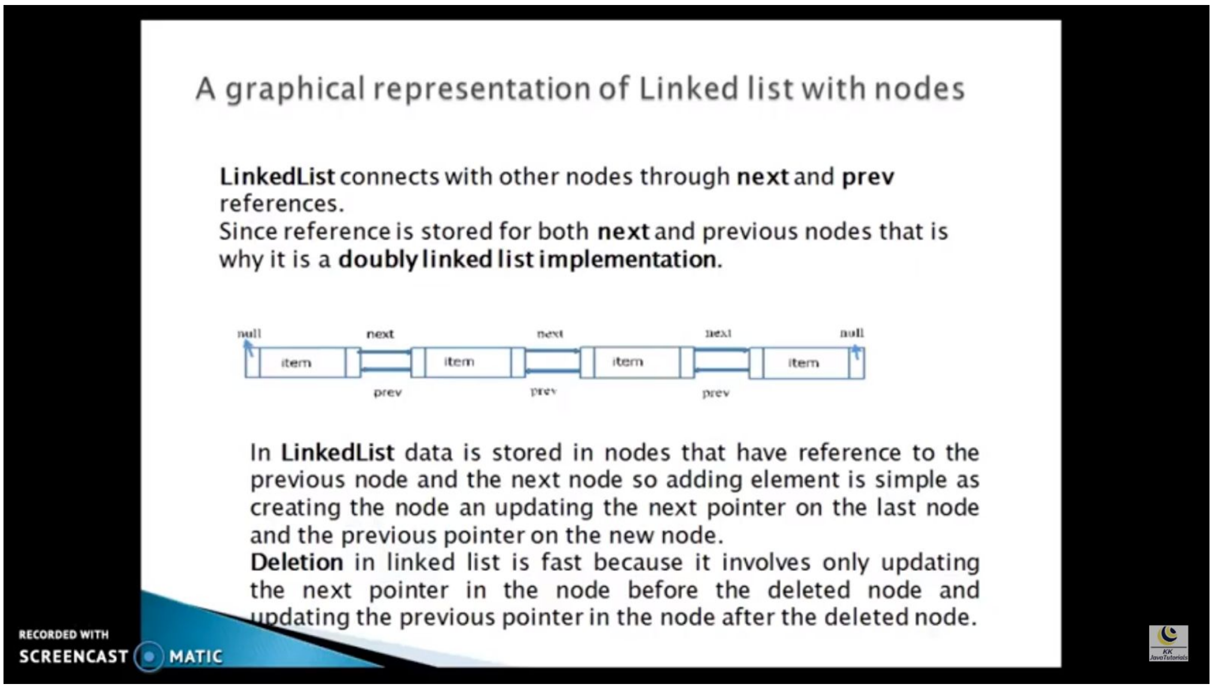
Linked List Implementation in Java



A graphical representation of Linked list with nodes

LinkedList connects with other nodes through **next** and **prev** references.
Since reference is stored for both **next** and previous nodes that is why it is a **doubly linked list implementation**.

In **LinkedList** data is stored in nodes that have reference to the previous node and the next node so adding element is simple as creating the node an updating the next pointer on the last node and the previous pointer on the new node.
Deletion in linked list is fast because it involves only updating the next pointer in the node before the deleted node and updating the previous pointer in the node after the deleted node.

Addition / insertion in Linked list is fast, because it just involves updating the pointers (next / previous) of the relative elements. It does not involve re-indexing.

So if there has to be a very frequent addition / deletion of elements in the Collection, Linked List is ideal.

How does LinkedList class store its element

There is a private class called Node, inside LinkedList class, which provides the structure for the nodes of a double Linked List (contains references to the next and previous nodes in the List).
It has an Item variable for holding the actual value, and 2 variables for storing the next and previous variables.
Internally LinkedList class in Java uses objects of type **Node** to store the added elements. Node is implemented as a static class within the LinkedList class. Since *LinkedList class is implemented as a doubly linked list* so each node stores reference to the next as well as previous nodes along with the added element.

Node class code in JDK 10 :

```
private static class Node<E> {
        E item;
        Node<E> next;
        Node<E> prev;

        Node(Node<E> prev, E element, Node<E> next) {
            this.item = element;
            this.next = next;
            this.prev = prev;
        }
}
```

Is it a doubly linked list

As already shown Java LinkedList class is implemented as a doubly linked list and each node stores reference to the next as well as previous nodes.

How add() method works in a LinkedList

Since it is a linked list so apart from regular **add()** method to add sequentially there are **addFirst()** and **addLast()** methods also in Java LinkedList class.
If you call addFirst method, internally the linkFirst() method is called.
There are also separate variables for holding the reference of the first and last nodes of the linked list.

```
/**
* Pointer to first node.
*/
transient Node<E> first;

/**
* Pointer to last node.
*/
transient Node<E> last;
```

If you call the regular add() method or addLast() method, internally linkLast() method is called. In this method a new node is created to store the added element and the variable last starts referring to this node (as this new node becomes the last node). There is also a check to see if it is the very first insertion in that case this node is the first node too.

linkLast() method implementation in the LinkedList class

```
/**
     * Links e as last element.
     */
  void linkLast(E e) {
      final Node<E> l = last;
      final Node<E> newNode = new Node<>(l, e, null);
      last = newNode;
      if (l == null)
          first = newNode;
      else
          l.next = newNode;
      size++;
      modCount++;
  }
```

If you call **addFirst()** method internally **linkFirst()** method is called. In this method a new node is created to store the added element and the variable first starts referring to this node (as this new node becomes the first node). There is also a check to see if it is the very first insertion in that case this node is the last node too. If it is not the first node then the previous “first” node now comes at the second position so its prev reference has to refer to the new node.

LinkFirst() method implementation in the LinkedList class

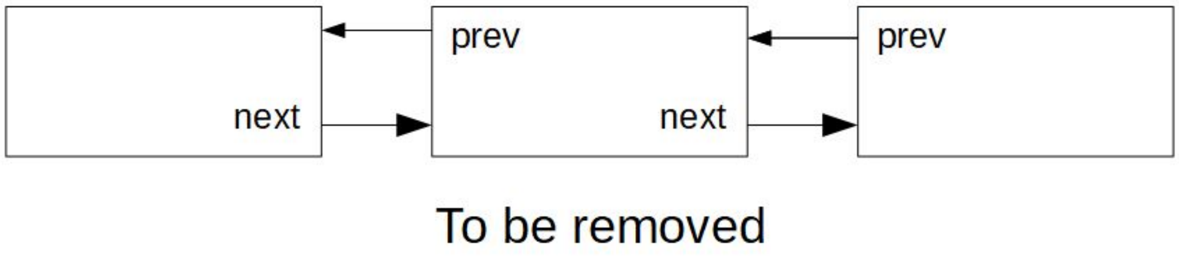
```
/**
     * Links e as first element.
     */
   private void linkFirst(E e) {
       final Node<E> f = first;
       final Node<E> newNode = new Node<>(null, e, f);
       first = newNode;
       if (f == null)
          last = newNode;
       else
          f.prev = newNode;
       size++;
       modCount++;
   }
```

There is also an add() method to add element at a specific index. If that method is called then the already existing element at the passed index has to be shifted right.

How remove() method works in Java LinkedList class

Just like add() method for removing an element from the LinkedList apart from regular **remove()** method (where index or the element is passed) there are also **removeFirst()** and **removeLast()** methods.

When remove() method is called then the reference of the nodes at the left and right of the removed nodes have to be changed so that next of the node at left starts referring to the node at the right and the prev of the node at right starts referring to the node at the left of the node to be removed.



How get() method works in Java LinkedList class

In the case of **get()** method again there are **getFirst()** and **getLast()** method too. In case of get() method it has to get the node for the passed index and return the node.item.

```
public E get(int index) {
    checkElementIndex(index);
    return node(index).item;
}
```

In case of getFirst() and getLast() reference is also stored in the first and last variables so just need to return the value first.item or last.item.