JAVA2NOVICE.COM

JAVA EXAMPLE PROGRAMS

DOUBLY LINKED LIST IMPLEMENTATION

A doubly-linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields, called links, 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. If there is only one sentinel node, then the list is circularly linked via the sentinel node. It can be conceptualized as two singly linked lists formed from the same data items, but in opposite sequential orders.

Here is the pictorial view of doubly linked list:

The two node links allow traversal of the list in either direction. While adding or removing a node in a doubly-linked list requires changing more links than the same operations on a singly linked list, the operations are simpler and potentially more efficient, because there is no need to keep track of the previous node during traversal or no need to traverse the list to find the previous node, so that its link can be modified.

Here is the pictorial view of inserting an element in the middle of a doubly linked list:

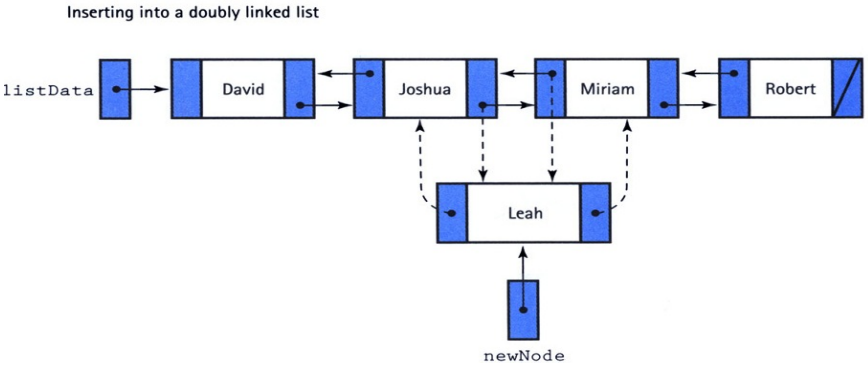


Image Reference: younginc.site11.com

Here is the pictorial view of deleting an element in the middle of a doubly linked list:

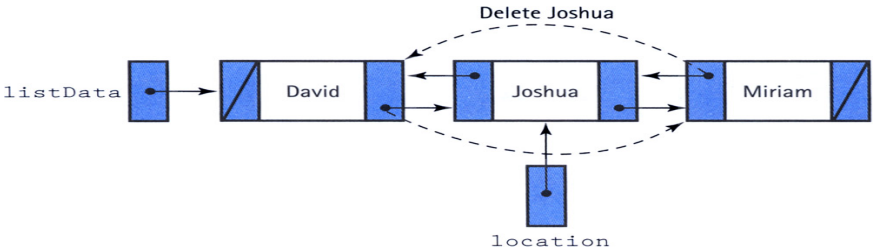
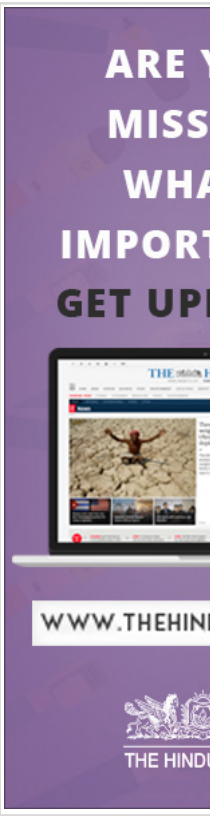


Image Reference: younginc.site11.com

Below shows the java implementation of doubly linked list:

```
1  package com.java2novice.ds.linkedlist;
2
3  import java.util.NoSuchElementException;
4
5  public class DoublyLinkedListImpl<E> {
6
7      private Node head;
8      private Node tail;
9      private int size;
10
11     public DoublyLinkedListImpl() {
12         size = 0;
13     }
14     /**
15      * this class keeps track of each element information
16      * @author java2novice
17      *
18      */
19     private class Node {
20         E element;
```

```
21          Node next;
22          Node prev;
23
24          public Node(E element, Node next, Node prev) {
25              this.element = element;
26              this.next = next;
27              this.prev = prev;
28          }
29      }
30      /**
31       * returns the size of the linked list
32       * @return
33       */
34      public int size() { return size; }
35
36      /**
37       * return whether the list is empty or not
38       * @return
39       */
40      public boolean isEmpty() { return size == 0; }
41
42      /**
43       * adds element at the starting of the linked list
44       * @param element
45       */
46      public void addFirst(E element) {
47          Node tmp = new Node(element, head, null);
48          if(head != null ) {head.prev = tmp;}
49          head = tmp;
50          if(tail == null) { tail = tmp;}
51          size++;
52          System.out.println("adding: "+element);
53      }
54
55      /**
56       * adds element at the end of the linked list
57       * @param element
58       */
59      public void addLast(E element) {
60
61          Node tmp = new Node(element, null, tail);
62          if(tail != null) {tail.next = tmp;}
63          tail = tmp;
64          if(head == null) { head = tmp;}
65          size++;
66          System.out.println("adding: "+element);
67      }
68
69      /**
70       * this method walks forward through the linked list
71       */
72      public void iterateForward(){
73
74          System.out.println("iterating forward..");
75          Node tmp = head;
76          while(tmp != null){
77              System.out.println(tmp.element);
78              tmp = tmp.next;
79          }
80      }
81
82      /**
83       * this method walks backward through the linked list
84       */
85      public void iterateBackward(){
86
87          System.out.println("iterating backword..");
88          Node tmp = tail;
89          while(tmp != null){
90              System.out.println(tmp.element);
91              tmp = tmp.prev;
92          }
93      }
94
95      /**
96       * this method removes element from the start of the linked list
97       * @return
98       */
99      public E removeFirst() {
100         if (size == 0) throw new NoSuchElementException();
101         Node tmp = head;
102         head = head.next;
103         head.prev = null;
104         size--;
105         System.out.println("deleted: "+tmp.element);
106         return tmp.element;
107     }
108
109     /**
110      * this method removes element from the end of the linked list
111      * @return
112      */
113     public E removeLast() {
114         if (size == 0) throw new NoSuchElementException();
115         Node tmp = tail;
116         tail = tail.prev;
117         tail.next = null;
```

```
118             size--;
119             System.out.println("deleted: "+tmp.element);
120             return tmp.element;
121         }
122
123         public static void main(String a[]){
124
125             DoublyLinkedListImpl<Integer> dll = new DoublyLinkedListImpl<Integer>();
126             dll.addFirst(10);
127             dll.addFirst(34);
128             dll.addLast(56);
129             dll.addLast(364);
130             dll.iterateForward();
131             dll.removeFirst();
132             dll.removeLast();
133             dll.iterateBackward();
134         }
135     }
```

Output:

```
adding: 10
adding: 34
adding: 56
adding: 364
iterating forward..
34
10
56
364
deleted: 34
deleted: 364
iterating backword..
56
10
```

<< Previous Program

LIST OF LINKED LIST DATA STRUCTURE EXAMPLES

Singly linked list implementation in java

Doubly linked list in Java

About Author

I'm Nataraja Gootooru, programmer by profession and passionate about technologies. All examples given here are as simple as possible to help beginners. The source code is compiled and tested in my dev environment.

If you come across any mistakes or bugs, please email me to java2novice@gmail.com or you can comment on the page.

Most Visited Pages

Java Interview Questions

How to Create Java Custom Exception

Java Interview Programs

Java StringTokenizer With Multiple De-limiters Sample Code

Java Constructor Chaining Examples

Spring Framework Examples

Write a program to find maximum repeated words from a file.

Java Data Structures

Other Interesting Sites

Techie Park

Java Design Pattern Tutorial

Software Testing

Wikipedia

Tips2Healthy

query2nataraj.blogspot.in