Write a Java program for the following: 1) Create a doubly linked list of elements. 2) Delete a given element from the above list. 3) Display the contents of the list after deletion.

**PROGRAM**

/\*

\* Java Program to Implement Doubly Linked List

\*/

import java.util.Scanner;

/\* Class Node \*/

class Node

{

protected int data;

protected Node next, prev;

/\* Constructor \*/

public Node()

{

next = null;

prev = null;

data = 0;

}

/\* Constructor \*/

public Node(int d, Node n, Node p)

{

data = d;

next = n;

prev = p;

}

/\* Function to set link to next node \*/

public void setLinkNext(Node n)

{

next = n;

}

/\* Function to set link to previous node \*/

public void setLinkPrev(Node p)

{

prev = p;

}

/\* Funtion to get link to next node \*/

public Node getLinkNext()

{

return next;

}

/\* Function to get link to previous node \*/

public Node getLinkPrev()

{

return prev;

}

/\* Function to set data to node \*/

public void setData(int d)

{

data = d;

}

/\* Function to get data from node \*/

public int getData()

{

return data;

}

}

/\* Class linkedList \*/

class linkedList

{

protected Node start;

protected Node end ;

public int size;

/\* Constructor \*/

public linkedList()

{

start = null;

end = null;

size = 0;

}

/\* Function to check if list is empty \*/

public booleanisEmpty()

{

return start == null;

}

/\* Function to get size of list \*/

public int getSize()

{

return size;

}

/\* Function to insert element at begining \*/

public void insertAtStart(int val)

{

Node nptr = new Node(val, null, null);

if(start == null)

{

start = nptr;

end = start;

}

else

{

start.setLinkPrev(nptr);

nptr.setLinkNext(start);

start = nptr;

}

size++;

}

/\* Function to insert element at end \*/

public void insertAtEnd(int val)

{

Node nptr = new Node(val, null, null);

if(start == null)

{

start = nptr;

end = start;

}

else

{

nptr.setLinkPrev(end);

end.setLinkNext(nptr);

end = nptr;

}

size++;

}

/\* Function to insert element at position \*/

public void insertAtPos(int val , int pos)

{

Node nptr = new Node(val, null, null);

if (pos == 1)

{

insertAtStart(val);

return;

}

Node ptr = start;

for (int i = 2; i<= size; i++)

{

if (i == pos)

{

Node tmp = ptr.getLinkNext();

ptr.setLinkNext(nptr);

nptr.setLinkPrev(ptr);

nptr.setLinkNext(tmp);

tmp.setLinkPrev(nptr);

}

ptr = ptr.getLinkNext();

}

size++ ;

}

/\* Function to delete node at position \*/

public void deleteAtPos(int pos)

{

if (pos == 1)

{

if (size == 1)

{

start = null;

end = null;

size = 0;

return;

}

start = start.getLinkNext();

start.setLinkPrev(null);

size--;

return ;

}

if (pos == size)

{

end = end.getLinkPrev();

end.setLinkNext(null);

size-- ;

}

Node ptr = start.getLinkNext();

for (int i = 2; i<= size; i++)

{

if (i == pos)

{

Node p = ptr.getLinkPrev();

Node n = ptr.getLinkNext();

p.setLinkNext(n);

n.setLinkPrev(p);

size-- ;

return;

}

ptr = ptr.getLinkNext();

}

}

/\* Function to display status of list \*/

public void display()

{

System.out.print("\nDoubly Linked List = ");

if (size == 0)

{

System.out.print("empty\n");

return;

}

if (start.getLinkNext() == null)

{

System.out.println(start.getData() );

return;

}

Node ptr = start;

System.out.print(start.getData()+ " <-> ");

ptr = start.getLinkNext();

while (ptr.getLinkNext() != null)

{

System.out.print(ptr.getData()+ " <-> ");

ptr = ptr.getLinkNext();

}

System.out.print(ptr.getData()+ "\n");

}

}

/\* Class DoublyLinkedList \*/

public class DoublyLinkedList

{

public static void main(String[] args)

{

Scanner scan = new Scanner(System.in);

/\* Creating object of linkedList \*/

linkedList list = new linkedList();

System.out.println("Doubly Linked List Test\n");

char ch;

/\* Perform list operations \*/

do

{

System.out.println("\nDoubly Linked List Operations\n");

System.out.println("1. insert at begining");

System.out.println("2. insert at end");

System.out.println("3. insert at position");

System.out.println("4. delete at position");

System.out.println("5. check empty");

System.out.println("6. display");

int choice = scan.nextInt();

switch (choice)

{

case 1 :

System.out.println("Enter integer element to insert");

list.insertAtStart( scan.nextInt() );

break;

case 2 :

System.out.println("Enter integer element to insert");

list.insertAtEnd( scan.nextInt() );

break;

case 3 :

System.out.println("Enter integer element to insert");

int num = scan.nextInt() ;

System.out.println("Enter position");

int pos = scan.nextInt() ;

if (pos < 1 || pos >list.getSize() )

System.out.println("Invalid position\n");

else

list.insertAtPos(num, pos);

break;

case 4 :

System.out.println("Enter position");

int p = scan.nextInt() ;

if (p < 1 || p >list.getSize() )

System.out.println("Invalid position\n");

else

list.deleteAtPos(p);

break;

case 5 :

System.out.println("Empty status = "+ list.isEmpty());

break;

case 6 :

System.out.println("Size = "+ list.getSize() +" \n");

break;

default :

System.out.println("Wrong Entry \n ");

break;

}

/\* Display List \*/

list.display();

System.out.println("\nDo you want to continue (Type y or n) \n");

ch = scan.next().charAt(0);

} while (ch == 'Y'|| ch == 'y');

}

}

**Write a Java program that implements Quick sort algorithm for sorting a list of names in ascending order.**

public class QuickSort

{

String n[];

int len;

public static void main(String[] args)

{

QuickSortob = new QuickSort();

String s[] = {"Ram", "Tom", "Carl", "Amal", "John", "Catherine", "Alan", "Rose", "Hari"}; System.out.println("Before sorting :\n");

for (String i : s)

{

System.out.print(i+ " ");

}

ob.sort(s);

System.out.println("\nAfter sorting :\n");

for (String i : s)

{

System.out.println(i);

}

}

void sort(String str[])

{

if (str == null || str.length == 0)

{

System.out.print("No string entered");

return;

}

this.n = str;

this.len = str.length;

quickSort(0, len - 1);

}

void quickSort(int lower, int upper)

{

int l = lower;

int u = upper;

String pivot = this.n[lower + (upper - lower) / 2];

while (l <= u)

{

while (this.n[l].compareToIgnoreCase(pivot) < 0)

{

l++;

}

while (this.n[u].compareToIgnoreCase(pivot) > 0)

{

u--;

}

if (l <= u)

{

exchangeNames(l, u);

l++;

u--;

}

}

if (lower < u)

{

quickSort(lower, u);

}

if (l < upper)

{

quickSort(l, upper);

}

}

void exchangeNames(int l, int u)

{

String temp = this.n[l];

this.n[l] = this.n[u];

this.n[u] = temp;

}

}