**Case study 2:**

Implement a data structure called doubly linked list with the following node structure. Duplicate data is allowed.

| **class Node<T> {**  **T data;** **Node<T> previous;** **Node<T> next;**  **}** |
| --- |

Implement the following operations:

1. Add node first
2. Add node last
3. Add node at an index
4. Remove first node
5. Remove last node
6. Remove node at an index
7. Find a node in O (1)

1. Add node first

public **static** <T> void addNodeFirst(T data) {

Node<T> newNode = new Node(data);

if (*head* == null) {

*head* = *tail* = newNode;

*head*.previous = null;

*tail*.next = null;

} else {

*head*.previous = newNode;

newNode.next = *head*;

newNode.previous = null;

*head* = newNode;

}

}

2.Add node last

public **static** <T> void addNodeLast(T data) {

Node<T> newNode = new Node(data);

if (*head* == null) {

*head* = *tail* = newNode;

*head*.previous = null;

*tail*.next = null;

} else {

*tail*.next = newNode;

newNode.previous = *tail*;

*tail* = newNode;

*tail*.next = null;

}

}

3.Add node at an index

public **static** <T> void addNodeAtIndex(T data, int position) {

Node<T> node = new Node(data);

node.data = data;

node.next = null;

if (*head* == null) {

if (position != 0) {

return;

} else {

*head* = node;

}

}

if (*head* != null && position == 0) {

node.next = *head*;

*head* = node;

return;

}

Node<T> current = *head*;

Node<T> previous = null;

int i = 0;

while (i < position) {

previous = current;

current = current.next;

if (current == null) {

break;

}

i++;

}

node.next = current;

previous.next = node;

}

4.Remove first node

public **static** <T> void removeFirstNode() {

if (*head* == null) {

return;

} else {

if (*head* != *tail*) {

*head* = *head*.next;

*head*.previous = null;

}

else {

*head* = *tail* = null;

}

}

}

5. Remove last node

public **static** <T> void removeLastNode() {

if (*head* != null) {

if (*head*.next == null) {

*head* = null;

} else {

Node<T> temp = new Node();

temp = *head*;

while (temp.next.next != null)

temp = temp.next;

Node lastNode = temp.next;

temp.next = null;

lastNode = null;

}

}

}

6. Remove node at an index

public **static** <T> void removeNodeAtIndex(int n) {

if (*head* == null) {

return;

} else {

Node current = *head*;

for (int i = 1; i < n; i++) {

current = current.next;

}

if (current == *head*) {

*head* = current.next;

} else if (current == *tail*) {

*tail* = *tail*.previous;

} else {

current.previous.next = current.next;

current.next.previous = current.previous;

}

current = null;

}

}

7.Find a node in O (1)

public **static** <T> int findNode(Node head\_ref, T x) {

Node<T> temp = head\_ref;

int pos = 0;

while (temp.data != x && temp.next != null) {

pos++;

temp = temp.next;

}

if (temp.data != x)

return -1;

return (pos + 1);

}

PROGRAM.:

class Node<T> {

T data;

Node<T> previous;

Node<T> next;

public Node(T data) {

super();

this.data = data;

}

public Node() {

}

}

public class TestMain {

**static** Node *head*;

**static** Node *tail* = null;

public **static** void main(String[] args) {

// Adding 1 to the list

*addNodeFirst*(1);

*addNodeFirst*(2);

*addNodeFirst*(3);

*display*();

*addNodeLast*(4);

*addNodeLast*(5);

*display*();

*addNodeAtIndex*(7, 1);

*addNodeAtIndex*(9, 3);

*addNodeAtIndex*(12, 5);

*display*();

*removeFirstNode*();

*removeLastNode*();

*display*();

*removeNodeAtIndex*(1);

*display*();

System.***out***.println(*findNode*(*head*, 12));

}

public **static** <T> void addNodeFirst(T data) {

Node<T> newNode = new Node(data);

if (*head* == null) {

*head* = *tail* = newNode;

*head*.previous = null;

*tail*.next = null;

} else {

*head*.previous = newNode;

newNode.next = *head*;

newNode.previous = null;

*head* = newNode;

}

}

public **static** <T> void addNodeLast(T data) {

Node<T> newNode = new Node(data);

if (*head* == null) {

*head* = *tail* = newNode;

*head*.previous = null;

*tail*.next = null;

} else {

*tail*.next = newNode;

newNode.previous = *tail*;

*tail* = newNode;

*tail*.next = null;

}

}

public **static** <T> void addNodeAtIndex(T data, int position) {

Node<T> node = new Node(data);

node.data = data;

node.next = null;

if (*head* == null) {

if (position != 0) {

return;

} else {

*head* = node;

}

}

if (*head* != null && position == 0) {

node.next = *head*;

*head* = node;

return;

}

Node<T> current = *head*;

Node<T> previous = null;

int i = 0;

while (i < position) {

previous = current;

current = current.next;

if (current == null) {

break;

}

i++;

}

node.next = current;

previous.next = node;

}

public **static** <T> void removeFirstNode() {

if (*head* == null) {

return;

} else {

if (*head* != *tail*) {

*head* = *head*.next;

*head*.previous = null;

}

else {

*head* = *tail* = null;

}

}

}

public **static** <T> void removeLastNode() {

if (*head* != null) {

if (*head*.next == null) {

*head* = null;

} else {

Node<T> temp = new Node();

temp = *head*;

while (temp.next.next != null)

temp = temp.next;

Node lastNode = temp.next;

temp.next = null;

lastNode = null;

}

}

}

public **static** <T> void removeNodeAtIndex(int n) {

if (*head* == null) {

return;

} else {

Node current = *head*;

for (int i = 1; i < n; i++) {

current = current.next;

}

if (current == *head*) {

*head* = current.next;

} else if (current == *tail*) {

*tail* = *tail*.previous;

} else {

current.previous.next = current.next;

current.next.previous = current.previous;

}

current = null;

}

}

public **static** <T> int findNode(Node head\_ref, T x) {

Node<T> temp = head\_ref;

int pos = 0;

while (temp.data != x && temp.next != null) {

pos++;

temp = temp.next;

}

if (temp.data != x)

return -1;

return (pos + 1);

}

public **static** <T> void display() {

Node<T> current = *head*;

if (*head* == null) {

System.***out***.println("List is empty");

return;

}

while (current != null) {

System.***out***.print(current.data + " ");

current = current.next;

}

System.***out***.println();

}

}