Kth Node From End Of Linked List

Question

1. You are given a partially written LinkedList class.  
2. Here is a list of existing functions:  
2.1 addLast - adds a new element with given value to the end of Linked List  
2.2. display - Prints the elements of linked list from front to end in a single line. All elements are separated by space  
2.3. size - Returns the number of elements in the linked list.  
2.4. removeFirst - Removes the first element from Linked List.   
2.5. getFirst - Returns the data of first element.   
2.6. getLast - Returns the data of last element.   
2.7. getAt - Returns the data of element available at the index passed.   
2.8. addFirst - adds a new element with given value in front of linked list.  
2.9. addAt - adds a new element at a given index.  
2.10. removeLast - removes the last element of linked list.  
2.11. removeAt - remove an element at a given index  
3. You are required to complete the body of kthFromLast function. The function should be an iterative function and should return the kth node from end of linked list. Also, make sure to not use size data member directly or indirectly (by calculating size via making a traversal). k is a 0-based index. Assume that valid values of k will be passed.  
4. Input and Output is managed for you.

Input Format

Input is managed for you

Output Format

Output is managed for you

Constraints

1. Size property should not be used directly or indirectly  
2. Constant time, single traversal is expected  
3. Iterative solution, (not recursion) is expected

Sample Input

addLast 10  
getFirst  
addLast 20  
addLast 30  
getFirst  
getLast  
getAt 1  
addLast 40  
kthFromEnd 3  
getLast  
addLast 50  
removeFirst  
getFirst  
removeFirst  
removeFirst  
kthFromEnd 0  
removeFirst  
removeFirst  
getFirst  
quit

Sample Output

10  
10  
30  
20  
10  
40  
20  
50  
List is empty

import java.io.\*;

import java.util.\*;

public class Main {

public static class Node {

int data;

Node next;

}

public static class LinkedList {

Node head;

Node tail;

int size;

void addLast(int val) {

Node temp = new Node();

temp.data = val;

temp.next = null;

if (size == 0) {

head = tail = temp;

} else {

tail.next = temp;

tail = temp;

}

size++;

}

public int size() {

return size;

}

public void display() {

for (Node temp = head; temp != null; temp = temp.next) {

System.out.print(temp.data + " ");

}

System.out.println();

}

public void removeFirst() {

if (size == 0) {

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

} else if (size == 1) {

head = tail = null;

size = 0;

} else {

head = head.next;

size--;

}

}

public int getFirst() {

if (size == 0) {

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

return -1;

} else {

return head.data;

}

}

public int getLast() {

if (size == 0) {

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

return -1;

} else {

return tail.data;

}

}

public int getAt(int idx) {

if (size == 0) {

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

return -1;

} else if (idx < 0 || idx >= size) {

System.out.println("Invalid arguments");

return -1;

} else {

Node temp = head;

for (int i = 0; i < idx; i++) {

temp = temp.next;

}

return temp.data;

}

}

public void addFirst(int val) {

Node temp = new Node();

temp.data = val;

temp.next = head;

head = temp;

if (size == 0) {

tail = temp;

}

size++;

}

public void addAt(int idx, int val) {

if (idx < 0 || idx > size) {

System.out.println("Invalid arguments");

} else if (idx == 0) {

addFirst(val);

} else if (idx == size) {

addLast(val);

} else {

Node node = new Node();

node.data = val;

Node temp = head;

for (int i = 0; i < idx - 1; i++) {

temp = temp.next;

}

node.next = temp.next;

temp.next = node;

size++;

}

}

public void removeLast() {

if (size == 0) {

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

} else if (size == 1) {

head = tail = null;

size = 0;

} else {

Node temp = head;

for (int i = 0; i < size - 2; i++) {

temp = temp.next;

}

tail = temp;

tail.next = null;

size--;

}

}

public void removeAt(int idx) {

if (idx < 0 || idx >= size) {

System.out.println("Invalid arguments");

} else if (idx == 0) {

removeFirst();

} else if (idx == size - 1) {

removeLast();

} else {

Node temp = head;

for (int i = 0; i < idx - 1; i++) {

temp = temp.next;

}

temp.next = temp.next.next;

size--;

}

}

private Node getNodeAt(int idx) {

Node temp = head;

for (int i = 0; i < idx; i++) {

temp = temp.next;

}

return temp;

}

public void reverseDI() {

int li = 0;

int ri = size - 1;

while(li < ri){

Node left = getNodeAt(li);

Node right = getNodeAt(ri);

int temp = left.data;

left.data = right.data;

right.data = temp;

li++;

ri--;

}

}

public void reversePI(){

if(size <= 1){

return;

}

Node prev = null;

Node curr = head;

while(curr != null){

Node next = curr.next;

curr.next = prev;

prev = curr;

curr = next;

}

Node temp = head;

head = tail;

tail = temp;

}

public int kthFromLast(int k){

Node temp=head, ans= head;

while(k>0){

temp=temp.next;

k--;

}

while(temp.next!= null){

temp=temp.next;

ans=ans.next;

}

return ans.data;

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

LinkedList list = new LinkedList();

String str = br.readLine();

while (str.equals("quit") == false) {

if (str.startsWith("addLast")) {

int val = Integer.parseInt(str.split(" ")[1]);

list.addLast(val);

} else if (str.startsWith("size")) {

System.out.println(list.size());

} else if (str.startsWith("display")) {

list.display();

} else if (str.startsWith("removeFirst")) {

list.removeFirst();

} else if (str.startsWith("getFirst")) {

int val = list.getFirst();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("getLast")) {

int val = list.getLast();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("getAt")) {

int idx = Integer.parseInt(str.split(" ")[1]);

int val = list.getAt(idx);

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("addFirst")) {

int val = Integer.parseInt(str.split(" ")[1]);

list.addFirst(val);

} else if (str.startsWith("addAt")) {

int idx = Integer.parseInt(str.split(" ")[1]);

int val = Integer.parseInt(str.split(" ")[2]);

list.addAt(idx, val);

} else if (str.startsWith("removeLast")) {

list.removeLast();

} else if (str.startsWith("removeAt")) {

int idx = Integer.parseInt(str.split(" ")[1]);

list.removeAt(idx);

} else if(str.startsWith("reverseDI")){

list.reverseDI();

} else if(str.startsWith("reversePI")){

list.reversePI();

} else if(str.startsWith("kthFromEnd")){

int idx = Integer.parseInt(str.split(" ")[1]);

System.out.println(list.kthFromLast(idx));

}

str = br.readLine();

}

}

}