## Topics

1. Implement Node Class
2. Generics
3. Implement SinglyLinkedList Class
4. Implement Basic Methods of SinglyLinkedList

* isEmpty()
* size()
* first()
* last()
* addFirst()
* addLast()
* removeFirst()

## Homework

1. develop an implementation of the equals method in the context of the SinglyLinkedList class.
2. Give an algorithm for finding the second-to-last node in a singly linked list in which the last node is indicated by a null next reference.
3. Give an implementation of the size( ) method for the SingularlyLinkedList class, assuming that we did not maintain size as an instance variable.
4. Implement a rotate( ) method in the SinglyLinkedList class, which has semantics equal to addLast(removeFirst( )), yet without creating any new node.
5. Describe an algorithm for concatenating two singly linked lists L and M, into a single list L′ that contains all the nodes of L followed by all the nodes of M.
6. Describe in detail an algorithm for reversing a singly linked list L using only a constant amount of additional space.

public class SinglyLinkedList<E> {

private static class Node<E> {

private E element;

private Node<E> next;

public Node(E element, Node<E> next) {

this.element = element;

this.next = next;

}

public E getElement() {

return element;

}

public Node<E> getNext() {

return next;

}

public void setNext(Node<E> next) {

this.next = next;

}

}

private Node<E> head = null;

private Node<E> tail = null;

private int size = 0;

public SinglyLinkedList() {}

public boolean isEmpty() {

return size == 0;

}

public void addFirst(E e) {

head = new Node<>(e, head); // إنشاء عقدة جديدة تربطها بالرأس القديم

if (size == 0) tail = head; // إذا كانت القائمة فارغة، الرأس هو أيضًا الذيل

size++;

}

public void addLast(E e) {

Node<E> newest = new Node<>(e, null); // إنشاء عقدة جديدة وإعدادها كآخر عقدة

if (isEmpty()) head = newest; // إذا كانت القائمة فارغة، تكون الرأس هي العقدة الجديدة

else tail.setNext(newest); / إذا لم تكن فارغة، يتم ربط العقدة الجديدة بالذيل الحالي

tail = newest; حديث الذيل ليصبح العقدة الجديدة

size++;

}

public E removeFirst() {

if (isEmpty()) return null; // إذا كانت القائمة فارغة، لا يوجد شيء لإزالته

E deleted = head.element; // تخزين العنصر الذي سيتم إزالته

head = head.next; // تحديث الرأس ليشير إلى العقدة التالية

size--;

if (size == 0) tail = null; / إذا أصبحت القائمة فارغة، يتم تحديث الذيل ليصبح فارغً

return deleted; // إرجاع العنصر المحذوف

}

public String getAll() {

all = new StringBuilder();

Node<E> current = head;

while (current != null) {/ / التكرار عبر جميع العقد

all.append(current.element).append(" ");

current = current.getNext();

}

return all.toString();

}

// 1. Method to find the second-to-last node

public Node<E> secondToLastNode() {

if (head == null || head.next == null) return null; // إذا كانت القائمة تحتوي على أقل من عقدتين

Node<E> current = head;

while (current.next.next != null) {التكرار حتى العقدة قبل الأخيرة

current = current.next;

}

return current; إرجاع العقدة الثانية من النهاية

}

// 2. Method to calculate size without maintaining size variable

public int calculateSize() {

int count = 0;

Node<E> current = head;

while (current != null) {

count++;

current = current.next;

}

return count;

}

// 3. Method to rotate the list

public void rotate() {

if (head == null || head.next == null) return null; // إذا كانت القائمة فارغة أو تحتوي على عنصر واحد

Node<E> oldHead = head; تخزين الرأس القديم

head = head.next; تحديث الرأس ليشير إلى العقدة

التالية

oldHead.next = null; فصل الرأس القديم

tail.next = oldHead; ربط الرأس القديم بالذيل

tail = oldHead; تحديث الذيل

}

// 4. Method to concatenate two lists

public static <E> SinglyLinkedList<E> concatenate(SinglyLinkedList<E> L, SinglyLinkedList<E> M) {

if (L.isEmpty()) return M; / إذا كانت القائمة L فارغة، إرجاع القائمة M

if (M.isEmpty()) return L; // إذا كانت القائمة M فارغة، إرجاع القائمة L

L.tail.next = M.head; // ربط ذيل القائمة L برأس القائمة M

L.tail = M.tail; / تحديث ذيل القائمة L ليصبح ذيل M

L.size += M.size; / إرجاع القائمة المدمجة

return L;

}

// 5. Method to reverse the list

public void reverse() {

Node<E> prev = null;

Node<E> current = head;

Node<E> next;

tail = head; // تحديث الذيل ليصبح الرأس القديم

while (current != null) {

next = current.next; تخزين العقدة التالية

current.next = prev; عكس الإشارة

prev = current; تحديث السابق

current = next; الانتقال إلى العقدة التالية

}

head = prev; تحديث الرأس ليصبح آخر عقدة

}

@Override

public boolean equals(Object obj) {

if (this == obj) return true; إذا كانتا نفس الكائن

if (obj == null || getClass() != obj.getClass()) return false; مقارنة النوع

SinglyLinkedList<?> other = (SinglyLinkedList<?>) obj;

if (this.size != other.size) return false; إذا كان الحجم مختلفً

Node<E> current1 = this.head;

Node<?> current2 = other.head;

while (current1 != null && current2 != null) {

if (!current1.getElement().equals(current2.getElement())) return false; مقارنة العناصر

current1 = current1.getNext();

current2 = current2.getNext();

}

return current1 == null && current2 == null;

}

}

// Test class

import java.util.Scanner;

public class Test {

public static void main(String[] args) {

SinglyLinkedList<String> list = new SinglyLinkedList<>();

Scanner in = new Scanner(System.in);

while (true) {

System.out.println("1. Add First\n2. Add Last\n3. Remove First\n4. Display Size\n5. Check if Empty");

System.out.println("6. Rotate List\n7. Reverse List\n8. Find Second-to-Last Node\n9. Concatenate Two Lists\n10. Compare Two Lists\n-1. Exit");

System.out.print("Enter your choice: ");

int choice = in.nextInt();

switch (choice) {

case 1:

System.out.print("Enter element: ");

list.addFirst(in.next());

break;

case 2:

System.out.print("Enter element: ");

list.addLast(in.next());

break;

case 3:

System.out.println("Removed: " + list.removeFirst());

break;

case 4:

System.out.println("Size: " + list.calculateSize());

break;

case 5:

System.out.println("Is Empty? " + list.isEmpty());

break;

case 6:

list.rotate();

System.out.println("List rotated.");

break;

case 7:

list.reverse();

System.out.println("List reversed.");

break;

case 8:

SinglyLinkedList.Node<String> secondToLast = list.secondToLastNode();

System.out.println("Second-to-last node: " + (secondToLast != null ? secondToLast.getElement() : "None"));

break;

case 9:

SinglyLinkedList<String> otherList = new SinglyLinkedList<>();

System.out.println("Enter elements for the second list (space-separated, end with -1):");

while (true) {

String input = in.next();

if (input.equals("-1")) break;

otherList.addLast(input);

}

SinglyLinkedList<String> concatenated = SinglyLinkedList.concatenate(list, otherList);

System.out.println("Concatenated list: " + concatenated.getAll());

break;

case 10:

SinglyLinkedList<String> anotherList = new SinglyLinkedList<>();

System.out.println("Enter elements for the second list (space-separated, end with -1):");

while (true) {

String input = in.next();

if (input.equals("-1")) break;

anotherList.addLast(input);

}

System.out.println("Are lists equal? " + list.equals(anotherList));

break;

case -1:

System.out.println("Goodbye!");

return;

default:

System.out.println("Invalid choice. Try again.");

}

System.out.println("Current List: " + list.getAll());

}

}

}