## Topics

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

* isEmpty()
* size()
* first()
* last()
* addFirst()
* addLast()
* removeFirst()
* public 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;
* }
* }
* public class SinglyLinkedList<E> {
* private Node<E> head;
* private Node<E> tail;
* private int size;
* public SinglyLinkedList() {
* head = null;
* tail = null;
* size = 0;
* }
* }
* public boolean isEmpty() {
* return size == 0;
* }
* public int size() {
* return size;
* }
* public E first() {
* if (isEmpty()) {
* return null;
* }
* return head.getElement();
* }
* public E last() {
* if (isEmpty()) {
* return null;
* }
* return tail.getElement();
* }
* public void addFirst(E element) {
* Node<E> newNode = new Node<>(element, head);
* head = newNode;
* if (isEmpty()) {
* tail = newNode;
* }
* size++;
* }
* public void addLast(E element) {
* Node<E> newNode = new Node<>(element, null);
* if (isEmpty()) {
* head = newNode;
* } else {
* tail.setNext(newNode);
* }
* tail = newNode;
* size++;
* }
* public E removeFirst() {
* if (isEmpty()) {
* return null;
* }
* E removedElement = head.getElement();
* head = head.getNext();
* size--;
* if (isEmpty()) {
* tail = null;
* }
* return removedElement;

}

## Homework

1. develop an implementation of the equals method in the context of the SinglyLinkedList class.

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj == null || getClass() != obj.getClass()) {

return false;

}

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

if (size() != otherList.size()) {

return false;

}

Node<E> currentNode = head;

Node<?> otherCurrentNode = otherList.head;

while (currentNode != null) {

if (!currentNode.getElement().equals(otherCurrentNode.getElement())) {

return false;

}

currentNode = currentNode.getNext();

otherCurrentNode = otherCurrentNode.getNext();

}

return true;

}

1. 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.

Node<E> findSecondToLast() {

if (head == null || head.getNext() == null) {

return null; // List has fewer than 2 nodes

}

Node<E> currentNode = head;

while (currentNode.getNext().getNext() != null) {

currentNode = currentNode.getNext();

}

return currentNode;

}

1. Give an implementation of the size( ) method for the SingularlyLinkedList class, assuming that we did not maintain size as an instance variable.

public int size() {

int count = 0;

Node<E> currentNode = head;

while (currentNode != null) {

count++;

currentNode = currentNode.getNext();

}

return count;

}

1. Implement a rotate( ) method in the SinglyLinkedList class, which has semantics equal to addLast(removeFirst( )), yet without creating any new node.

public void rotate() {

if (head == null || head.getNext() == null) {

return; // List has 0 or 1 node, no rotation needed

}

Node<E> oldHead = head;

head = head.getNext();

tail.setNext(oldHead);

oldHead.setNext(null);

tail = oldHead;

}

1. 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.

Node<E> concatenateLists(Node<E> headL, Node<E> headM) {

if (headL == null) {

return headM;

}

Node<E> currentNode = headL;

while (currentNode.getNext() != null) {

currentNode = currentNode.getNext();

}

currentNode.setNext(headM);

return headL;

}

1. Describe in detail an algorithm for reversing a singly linked list L using only a constant amount of additional space.

Node<E> reverseList(Node<E> head) {

Node<E> prev = null;

Node<E> current = head;

Node<E> next = null;

while (current != null) {

next = current.getNext();

current.setNext(prev);

prev = current;

current = next;

}

return prev;

}