## 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 e, Node<E> n) {
* element = e;
* next = n;
* }
* public E getElement() { return element; }
* public Node<E> getNext() { return next; }
* public void setNext(Node<E> n) { next = n; }
* }
* // 3. Implement SinglyLinkedList Class
* public class SinglyLinkedList<E> {
* private Node<E> head = null;
* private Node<E> tail = null;
* private int size = 0;
* public SinglyLinkedList() { }
* // 4. Implement Basic Methods of SinglyLinkedList
* 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 e) {
* head = new Node<>(e, head);
* if (size == 0)
* tail = head;
* size++;
* }
* public void addLast(E e) {
* Node<E> newest = new Node<>(e, null);
* if (isEmpty())
* head = newest;
* else
* tail.setNext(newest);
* tail = newest;
* size++;
* }
* public E removeFirst() {
* if (isEmpty()) return null;
* E answer = head.getElement();
* head = head.getNext();
* size--;
* if (size == 0)
* tail = null;
* return answer;
* }
* }

## Homework

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

public class SinglyLinkedList {

Node head; // head of list

/\* Linked list Node\*/

static class Node {

int data;

Node next;

// Constructor to create a new node

Node(int d) {

data = d;

next = null;

}

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

SinglyLinkedList that = (SinglyLinkedList) o;

Node current1 = this.head;

Node current2 = that.head;

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

if (current1.data != current2.data) return false;

current1 = current1.next;

current2 = current2.next;

}

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

}

}

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.

public Node secondToLast() {

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

// List is empty or has only one node, so there is no second-to-last node

return null;

}

Node current = head;

while (current.next.next != null) {

current = current.next;

}

// Now current.next is the last node, so current is the second-to-last node

return current;

}

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

public class SinglyLinkedList {

Node head; // head of list

/\* Linked list Node\*/

static class Node {

int data;

Node next;

// Constructor to create a new node

Node(int d) {

data = d;

next = null;

}

}

public int size() {

Node current = head;

int count = 0;

while (current != null) {

count++;

current = current.next;

}

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 class SinglyLinkedList {

Node head; // head of list

/\* Linked list Node\*/

static class Node {

int data;

Node next;

// Constructor to create a new node

Node(int d) {

data = d;

next = null;

}

}

public void rotate() {

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

// List is empty or has only one node, so rotation is not required

return;

}

Node current = head;

while (current.next.next != null) {

current = current.next;

}

// Now current.next is the last node and head is the first node

current.next.next = head; // Make last node point to first node

head = current.next; // Make second last node as head

current.next = null; // Make second last node as last node

}

}

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.

public class SinglyLinkedList {

Node head; // head of list

/\* Linked list Node\*/

static class Node {

int data;

Node next;

// Constructor to create a new node

Node(int d) {

data = d;

next = null;

}

}

public void concatenate(SinglyLinkedList M) {

if (head == null) {

// If L is empty, L' should be M

head = M.head;

} else {

// Traverse L until the last node

Node current = head;

while (current.next != null) {

current = current.next;

}

// Now current is the last node of L

current.next = M.head; // Concatenate M after L

}

}

}

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

public class SinglyLinkedList {

Node head; // head of list

/\* Linked list Node\*/

static class Node {

int data;

Node next;

// Constructor to create a new node

Node(int d) {

data = d;

next = null;

}

}

public void reverse() {

Node prev = null;

Node current = head;

Node next = null;

while (current != null) {

next = current.next; // Store next node

current.next = prev; // Reverse current node's pointer

prev = current; // Move pointers one position ahead

current = next;

}

head = prev;

}

}