Topics

* Implement Node Class
* Generics
* Implement SinglyLinkedList Class
* 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 T 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);

if (isEmpty()) {

tail = newNode;

}

head = 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;

}

T removedElement = head.getElement();

head = head.getNext();

size--;

if (isEmpty()) {

tail = null;

}

return removedElement;

}

}

Homework

* 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<E> otherList = (SinglyLinkedList<E>) obj;

if (size != otherList.size) {

return false;

}

Node<E> currentNode = head;

Node<E> otherCurrentNode = otherList.head;

while (currentNode != null) {

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

return false;

}

currentNode = currentNode.getNext();

otherCurrentNode = otherCurrentNode.getNext();

}

return true;

}

* 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<E> findSecondToLast() {

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

return null;

}

Node<E> current = head;

Node<E> previous = null;

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

previous = current;

current = current.getNext();

}

return previous;

}

* 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;

} public int size() {

int count = 0;

Node<E> currentNode = head;

while (currentNode != null) {

count++;

currentNode = currentNode.getNext();

}

return count;

}

* 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;

}

Node<E> rotatedNode = head;

head = head.getNext();

tail.setNext(rotatedNode);

rotatedNode.setNext(null);

tail = rotatedNode;

}

* 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 SinglyLinkedList<E> concatenateLists(SinglyLinkedList<E> L, SinglyLinkedList<E> M) {

if (L.isEmpty()) {

return M;

}

if (M.isEmpty()) {

return L;

}

SinglyLinkedList<E> concatenatedList = new SinglyLinkedList<>();

concatenatedList.head = L.head; // Set the head of concatenatedList to the head of L

concatenatedList.tail = M.tail; // Set the tail of concatenatedList to the tail of M

L.tail.setNext(M.head); // Connect the tail of L to the head of M

L.tail = M.tail; // Update the tail of L to be the same as the tail of M

return concatenatedList;

}

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

public void reverse() {

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

return;

}

Node<E> current = head;

Node<E> previous = null;

Node<E> next = null;

while (current != null) {

next = current.getNext();

current.setNext(previous);

previous = current;

current = next;

}

head = previous;

}