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

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

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.

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

1. تنفيذ فئة العقدة (Node):

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 void setElement(E element) {

this.element = element;

}

public Node<E> getNext() {

return next;

}

public void setNext(Node<E> next) {

this.next = next;

}

}

تم تنفيذ فئة العقدة (Node) التي تحتوي على عنصر ومرجع إلى العقدة التالية.

1. التعامل مع العناصر الجنريك (Generics):

تم استخدام العناصر الجنريك في الأمثلة الموجودة أعلاه. يتيح لك العناصر الجنريك تحديد نوع البيانات التي ستتعامل معها الهياكل والطرق.

1. تنفيذ فئة قائمة مرتبطة (SinglyLinkedList):

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, null);

if (isEmpty()) {

head = newNode;

tail = newNode;

} else {

newNode.setNext(head);

head = newNode;

}

size++;

}

public void addLast(E element) {

Node<E> newNode = new Node<>(element, null);

if (isEmpty()) {

head = newNode;

tail = 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;

}

}

تم تنفيذ فئة القائمة المرتبطة (SinglyLinkedList) التي تحتوي على العديد من العمليات الأساسية للتعامل مع العناصر في القائمة.

1. تنفيذ الطرق الأساسية لفئة SinglyLinkedList:

public class SinglyLinkedList<E> {

// الطرق الأخرى

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) {

// إضافة العنصر في بداية القائمة

}

public void addLast(E element) {

// إضافة العنصر في نهاية القائمة

}

public E removeFirst() {

// إزالة العنصر الأول من القائمة وإرجاعه

}

}

الطرق المنفذة تشمل:

* isEmpty(): تحقق مما إذا كانت القائمةفارغة أم لا.
* size(): يُرجع حجم القائمة.
* first(): يُرجع العنصر الأول في القائمة.
* last(): يُرجع العنصر الأخير في القائمة.
* addFirst(): يُضيف عنصرًا في بداية القائمة.
* addLast(): يُضيف عنصرًا في نهاية القائمة.
* removeFirst(): يُزيل العنصر الأول من القائمة ويُرجعه.

1. تنفيذ طريقة equals() في سياق فئة SinglyLinkedList:

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

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

return false;

}

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

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

return false;

}

Node<E> currentNode = head;

Node<?> otherCurrentNode = other.head;

while (currentNode != null) {

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

return false;

}

currentNode = currentNode.getNext();

otherCurrentNode = otherCurrentNode.getNext();

}

return true;

}

تم تنفيذ طريقة equals() للقيام بمقارنة قائمتين مرتبطتين من حيث المحتوى. يتم مقارنة عناصر كل قائمة واحدة تلو الأخرى باستخدام طريقة equals() المعرفة لكل عنصر.

1. الخوارزمية لإيجاد العقدة الثانية إلى الأخيرة في قائمة مرتبطة واحدة:

public Node<E> findSecondToLastNode(SinglyLinkedList<E> list) {

if (list.size() < 2) {

return null;

}

Node<E> currentNode = list.head;

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

currentNode = currentNode.getNext();

}

return currentNode;

}

تقوم الخوارزمية بتحديد العقدة الثانية إلى الأخيرة في قائمة مرتبطة واحدة. يتم البدء من العقدة الأولى والتقدم حتى الوصول إلى العقدة قبل العقدة الأخيرة.

1. تنفيذ طريقة size() لفئة SinglyLinkedList بدون الاحتفاظ بمتغير الحجم كمتغير فردي:

public int size() {

int count = 0;

Node<E> currentNode = head;

while (currentNode != null) {

count++;

currentNode = currentNode.getNext();

}

return count;

}

تستخدم الخوارزمية العداد لتحديد حجم القائمة. تبدأ من العقدة الأولى وتزيد العداد بمقدار واحد لكل عقدة يتم الانتقال إليها حتى تصل إلى نهاية القائمة.

1. تنفيذ طريقة rotate() في فئة SinglyLinkedList:

public void rotate() {

if (isEmpty() || size() == 1) {

return;

}

Node<E> oldHead = head;

head = head.getNext();

tail.setNext(oldHead);

tail = oldHead;

tail.setNext(null);

}

تنفيذ طريقة rotate() التي تقوم بتدوير القائمة بحيث تُزال العقدة الأولى