|  |  |
| --- | --- |
| Homework\_5 |  |
| 1. 1\_ Consider the implementation of CircularlyLinkedList.addFirst, in Code Fragment 3.16. The else body at lines 39 and 40 of that method relies on a locally declared variable, newest. Redesign that clause to avoid use of any local variable. | public void addFirst(E e) {  if (isEmpty()) {  tail = new Node<>(e, null);  tail.next = tail; // دائري: تشير لنفسها  } else {  tail.next = new Node<>(e, tail.next); // إضافة مباشرة  }  } |
| 1. 2\_Give an implementation of the size( ) method for the CircularlyLinkedList class, assuming that we did not maintain size as an instance variable. | public void addFirst(E e) {  if (isEmpty()) {  tail = new Node<>(e, null);  tail.next = tail; // دائري: تشير لنفسها  } else {  tail.next = new Node<>(e, tail.next); // إضافة مباشرة  }  } |
| 1. 3\_Implement the equals( ) method for the CircularlyLinkedList class, assuming that two lists are equal if they have the same sequence of elements, with corresponding elements currently at the front of the list. | public boolean equals(CircularlyLinkedList<E> other) {  if (this.size() != other.size()) return false; // الأحجام مختلفة  Node<E> currentA = this.tail.next; // الرأس الأولى  Node<E> currentB = other.tail.next; // الرأس الثانية  do {  if (!currentA.element.equals(currentB.element)) return false;  currentA = currentA.next;  currentB = currentB.next;  } while (currentA != this.tail.next); // العودة للبداية  return true;  } |
| 1. 4\_Suppose you are given two circularly linked lists, L and M. Describe an algorithm for telling if L and M store the same sequence of elements (but perhaps with different starting points). | public boolean areCircularListsEqual(CircularlyLinkedList<E> list1, CircularlyLinkedList<E> list2) {  if (list1.size() != list2.size()) {  return false; // إذا لم تكن الأحجام متساوية  }  Node<E> start1 = list1.tail.next; // الرأس الأولى  Node<E> start2 = list2.tail.next; // الرأس الثانية  for (int i = 0; i < list1.size(); i++) {  Node<E> current1 = start1;  Node<E> current2 = start2;  boolean isEqual = true;  // تحقق من العناصر في القائمتين  for (int j = 0; j < list1.size(); j++) {  if (!current1.element.equals(current2.element)) {  isEqual = false;  break;  }  current1 = current1.next;  current2 = current2.next;  }  if (isEqual) {  return true; // إذا تطابق التسلسل  }  start2 = start2.next; // جرب نقطة بداية جديدة  }  return false; // إذا لم نجد أي تسلسل مطابق  } |
| 1. 5\_Given a circularly linked list L containing an even number of nodes, describe how to split L into two circularly linked lists of half the size. | public CircularlyLinkedList<E>[] split() {  if (size() % 2 != 0) throw new IllegalStateException("List size must be even");  Node<E> slow = tail.next; // الرأس  Node<E> fast = tail.next;  while (fast.next != tail.next && fast.next.next != tail.next) {  slow = slow.next;  fast = fast.next.next;  }  CircularlyLinkedList<E> firstHalf = new CircularlyLinkedList<>();  CircularlyLinkedList<E> secondHalf = new CircularlyLinkedList<>();  firstHalf.tail = slow;  secondHalf.tail = tail;  tail = slow;  return new CircularlyLinkedList[]{firstHalf, secondHalf};  } |
| 1. 6\_Implement the clone( ) method for the CircularlyLinkedList class. 6: تنفيذ clone() 2. الحل: 3. قم بإنشاء قائمة جديدة ونسخ كل عنصر من القائمة الأصلية. | public CircularlyLinkedList<E> clone() {  CircularlyLinkedList<E> clonedList = new CircularlyLinkedList<>();  if (!isEmpty()) {  Node<E> current = tail.next; // الرأس  do {  clonedList.addLast(current.element); // نسخ كل عنصر  current = current.next;  } while (current != tail.next); // الدورة كاملة  }  return clonedList;  } |