1. What’s wrong with this definition: **Arrays arrays = new Arrays();**

Arrays الخطا في استخدام كلمة

1. Write and test this method: void reverse(int[] a)

// reverses the elements of a[]

|  |
| --- |
| public void reverse(int[] a) { int left = 0;  int right = a.length - 1;    while (left < right) {  // Swapping elements at left and right indices int temp = a[left]; a[left] = a[right]; a[right] = temp;    // Move the left index forward and the right index backward left++; right--;  }  } |

1. If linked lists are so much better than arrays, why are arrays used at all?

**1)** سهولة الاستخدام: المصفوفات أكثر بساطة في الاستخدام مقارنة بالقوائم المرتبطة .فهي تعمل بشكل مباشر ومبسط بدون الحاجة لإنشاء وإدارة وصيانة العقد والروابط

بين العناص ر

2( الوصول العشوائ ي

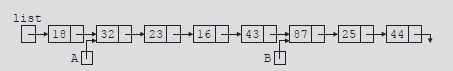
3( الكفاءة في التكرا ر

4) توفير الذاكرة الزائدة

**4- Mark the following statements as true or false.**

1. In a linked list, the order of the elements is determined by the order in which the nodes were created to store the elements. ture
2. In a linked list, memory allocated for the nodes is sequential. true
3. A single linked list can be traversed in either direction. fales
4. In a linked list, nodes are always inserted either at the beginning or the end because a linked link is not a random access data structure. fales
5. The head pointer of a linked list cannot be used to traverse the list. true

**Consider the linked list shown in Figure. Assume that the nodes are in the usual ElementNext form. Use this list to answer Exercises 5 through 8. If necessary, declare additional variables. (Assume that list, p, s, A, and B are references of type Node.)**



Linked list for Exercises 2–7 5- What is the output of each of the following java statements?

* 1. System.out.println( list.getElement()); 18
  2. System.out.println( A. getElement()); 32
  3. System.out.println( B.getNext().getElement()); 25
  4. System.out.println( list.getNext().getNext().getElement());44

1. What is the value of each of the following relational expressions?
   1. list. getElement() >= 18
   2. list.getNext() == A
   3. A.getNext().getElement() == 16
   4. B.getNext() == (NULL)
   5. list. getElement() == 18

1. Write java Fragment code to do the following:

a- Make A point to the node containing element 23.

1. Node currentNode = list;
2. while (currentNode != null) {
3. if (currentNode.getElement() == 23) {
4. currentNode.setMarked(true);
5. break;
6. }
7. currentNode = currentNode.getNext();
8. }

b-Make list point to the node containing 16.

Node currentNode = list; while (currentNode != null) { if (currentNode.getElement() == 16) { list = currentNode; break; }

currentNode = currentNode.getNext();

}

c-Make B point to the last node in the list.

|  |
| --- |
| Node currentNode = list; Node lastNode = null; while (currentNode != null) { lastNode = currentNode;  currentNode = currentNode.getNext(); } |

B = lastNode;

d-Make list point to an empty list.

list = null;

e-Set the value of the node containing 25 to 35.

Node currentNode = list; while (currentNode != null) { if (currentNode.getElement() == 25) { currentNode.setElement(35); break; }

currentNode = currentNode.getNext();

}

f-Create and insert the node with element 10 after the node pointed by A.

Node newNode = new Node(10); newNode.setNext(A.getNext());

A.setNext(newNode);

g-Delete the node with element 23. Also, deallocate the memory occupied by this node.

|  |
| --- |
| Node previousNode = null; Node currentNode = list; while (currentNode != null) { if (currentNode.getElement() == 23) { if (previousNode == null) { list = currentNode.getNext();  } else {  previousNode.setNext(currentNode.getNext());  }  currentNode = null; break; }  previousNode = currentNode; currentNode = currentNode.getNext(); } |

1. What is the output of the following java code?

p = list;

while (p != NULL){

System.out.println( p.getElement()); p = p.getNext(); }

1. Show what is produced by the following java code. Assume the node is in the usual **getElement()-getNext()** form with the info of type int. (**list** and **p** are pointers of type **node<E>()**.)

a- list = new node<E>(); list.setElement(10); p = new node<E>(); p. setElement(13);

p.setNext(null); list.setNext(p);

p = new node<E>(18, list.getNext()); list.setNext(p);

System.out.println(list.getElement()); System.out.println(p.getElement()); p = p.getNext();

System.out.println(p.getElement());

list = new node<E>(); list.setElement(20); p = new node<E>(); p. setElement(28);

p.setNext(NULL); list. setNext(p); p = new node<E>(); p.setElement(30);

p.setNext(list);

list = p;

p = new node<E>(); p.setElement(42);

p.setNext(list.getNext()); list.setNext(p);

p = List;

while (p != NULL)

{

System.out.println( p.getElement()); p = p.getNext(); }

1. **Consider the following java statements. (The class SingleLinkedList is as defined in the lectures).**

SingleLinkedList<int> list;

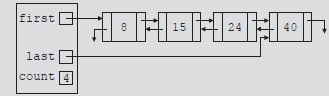
list.addFirst(15); list.addLast(28); list.addFirst(30); list.addFirst(2); list.addLast(45); list.addFirst(38); list.addLast(25); list.removeNode(30); list.addFirst(18); list.removeNode(28); list.removeNode(12);

list.print();

What is the output of this program segment?

38 2 18 15 45 25

1. For the following doubly linked list figure, show by java code how to insert value (info) 20 between values 15 & 24?



|  |
| --- |
| public class DoublyLinkedList { private Node head; private Node tail;    private class Node { int data; Node prev;  Node next;  public Node(int data) { this.data = data; this.prev = null; this.next = null;  }  } // إضافة قيمة في البداية public void addFirst(int data) { Node newNode = new Node(data);  if (head == null) { head = newNode; |

|  |
| --- |
| tail = newNode;  } else {  newNode.next = head; head.prev = newNode; head = newNode;  }  } // إضافة قيمة في النهاية public void addLast(int data) { Node newNode = new Node(data);  if (tail == null) { head = newNode; tail = newNode;  } else {  tail.next = newNode; newNode.prev = tail; tail = newNode;  }  } // إدراج قيمة بين قيمتين محددين public void insertBetween(int data, int value1, int value2) {  Node newNode = new Node(data);  Node currentNode = head;    while (currentNode != null) {  if (currentNode.data == value1 && currentNode.next != null && currentNode .next.data == value2) {  Node nextNode = currentNode.next; currentNode.next = newNode; newNode.prev = currentNode; newNode.next = nextNode; nextNode.prev = newNode; break;  }  currentNode = currentNode.next;  }  } // طباعة قيم القائمة public void print() { Node currentNode = head; while (currentNode != null) {  System.out.print(currentNode.data + " ");  currentNode = currentNode.next;  }  System.out.println();  }  public static void main(String[] args) {  DoublyLinkedList list = new DoublyLinkedList(); list.addFirst(15); list.addLast(24);  list.insertBetween(20, 15, 24); |

list.print(); // 24 20 15 :ستطبع

} }

1. Write and test this method for **SingleLinkedList class** : **Public int sum(Node<int> list)**

// returns: the sum of the integers in the specified list;

For example, if list is {25, 45, 65, 85}, then sum(list) will return 220.

|  |
| --- |
| public int sum() { int sum = 0;  Node currentNode = head;  while (currentNode != null) { sum += currentNode.data; currentNode = currentNode.next;  }  return sum;  }  public static void main(String[] args) {  SingleLinkedList list = new SingleLinkedList(); list.addLast(25); list.addLast(45); list.addLast(65); list.addLast(85);    int sum = list.sum(); System.out.println("مجموع القائمة: " + sum); // 220 :ستطبع: مجموع القائمة } |

1. Write and test this method for **DoublyLinkedList class**: **Public E removeLast(Node<E> list)**

// precondition: the specified list has at least two nodes;

// postcondition: the last node in the list has been deleted;

For example, if list is {22, 44, 66, 88}, then removeLast(list) will change it to {22, 44, 66}.

|  |
| --- |
| public void removeLast() { if (tail == null) { // القائمة فارغة return;  }  if (head == tail) { // القائمة تحتوي على عقدة واحدة فقط head = null; tail = null; return;  }  tail = tail.prev; tail.next = null; |
| }  public static void main(String[] args) {  DoublyLinkedList list = new DoublyLinkedList(); list.addLast(22); list.addLast(44); list.addLast(66); list.addLast(88);    System.out.print("القائمة قبل الإزالة: "); list.print(); // 88 66 44 22 :ستطبع    list.removeLast();    System.out.print("القائمة بعد الإزالة: "); list.print(); // 66 44 22 :ستطبع } |

1. Write and test this method for **SingleLinkedList class**:

**Public void append(Node<E> list1, Node<E> list2)**

// precondition: list1 has at least one node;

// postcondition: list1 has list2 appended to it;

For example, if list1 is {22, 33, 44, 55} and list2 is {66, 77, 88, 99}, then append(list1, list2) will change list1 to {22, 33, 44, 55, 44, 55, 66, 77, 88}. Note that no new nodes are created by this method.

|  |
| --- |
| public void append(SingleLinkedList<E> list2) { if (list2.head == null) { // لا يوجد عقد لإلحاقه return;  }    Node currentNode = head;  // الوصول إلى نهاية القائمة الأولى while (currentNode.next != null) { currentNode = currentNode.next;  { إلحاق القائمة الثانية بالقائمة الأولى // currentNode.next = list2.head;  }  public static void main(String[] args) {  SingleLinkedList<Integer> list1 = new SingleLinkedList<>(); list1.addLast(22); list1.addLast(33); list1.addLast(44); list1.addLast(55);    SingleLinkedList<Integer> list2 = new SingleLinkedList<>(); list2.addLast(66); list2.addLast(77); list2.addLast(88); |

list2.addLast(99);

System.out.print("القائمة 1 قبل الإلحاق: "); list1.print(); // 55 44 33 22 :ستطبع

list1.append(list2);

System.out.print("القائمة 1 بعد الإلحاق: "); list1.print(); // 99 88 77 66 55 44 33 22 :ستطبع }

1. Write and test this method for **SingleLinkedList class**: **Public Node<E> concat(Node<E> list1, Node<E> list2)**

// returns: a new list that contains a copy of list1, followed by a copy of list2;

For example, if list1 is {22, 33, 44, 55} and list2 is {66, 77, 88, 99}, then concat(list1, list2) will return the new list {22, 33, 44, 55, 44, 55, 66, 77, 88}. Note that the three lists should be completely independent of each other. Changing one list should have no effect upon the others.

|  |
| --- |
| public SingleLinkedList<E> concat(SingleLinkedList<E> list2) { SingleLinkedList<E> newList = new SingleLinkedList<>();  Node currentNode = head;  // نسخ القائمة 1 إلى القائمة الجديدة while (currentNode != null) { newList.addLast(currentNode.data); currentNode = currentNode.next;  { نسخ القائمة 2 إلى القائمة الجديدة //  currentNode = list2.head; while (currentNode != null) { newList.addLast(currentNode.data); currentNode = currentNode.next;  }  return newList;  }  public static void main(String[] args) {  SingleLinkedList<Integer> list1 = new SingleLinkedList<>(); list1.addLast(22); list1.addLast(33); list1.addLast(44); list1.addLast(55);    SingleLinkedList<Integer> list2 = new SingleLinkedList<>(); list2.addLast(66); list2.addLast(77); list2.addLast(88); list2.addLast(99);    System.out.print("1 القائمة: "); list1.print(); // 55 44 33 22 :ستطبع |
| System.out.print("2 القائمة: "); list2.print(); // 99 88 77 66 :ستطبع    SingleLinkedList<Integer> newList = list1.concat(list2);  System.out.print("القائمة الجديدة: "); newList.print(); // 99 88 77 66 55 44 33 22 :ستطبع } |

1. Write and test this method for **DoublyLinkedList class**:

**Public void swap(Node<E> list, int i, int j)**

// swaps the ith element with the jth element;

For example, if list is {22, 33, 44, 55, 66, 77, 88, 99}, then swap(list, 2, 5) will change list to {22, 33, 77, 55, 66, 44, 88, 99}.

|  |
| --- |
| public void swap(Node<E> list, int i, int j) { if (i == j) {  // No need to swap if i and j are the same return;  }    Node<E> node1 = getNodeAtIndex(list, i);  Node<E> node2 = getNodeAtIndex(list, j);  if (node1 == null || node2 == null) { // Invalid indices, cannot perform swap return;  }    E temp = node1.data; node1.data = node2.data; node2.data = temp;  }  private Node<E> getNodeAtIndex(Node<E> list, int index) {  Node<E> currentNode = list; int currentIndex = 0;  while (currentNode != null && currentIndex < index) { currentNode = currentNode.next; currentIndex++;  }  return currentNode;  }  public static void main(String[] args) {  DoublyLinkedList<Integer> list = new DoublyLinkedList<>(); list.addLast(22); list.addLast(33); list.addLast(44); list.addLast(55); list.addLast(66); list.addLast(77); list.addLast(88); list.addLast(99); |

System.out.print("List before swap: ");

list.print(); // Will print: 22 33 44 55 66 77 88 99

list.swap(list.head, 2, 5);

System.out.print("List after swap: ");

list.print(); // Will print: 22 33 77 55 66 44 88 99

}

1. Describe in detail(without java code) an algorithm for reversing a singly linked list *L* using only a constant amount of additional space.

(previous) لتتبع العقدة الحالية ، والثاني (current) قم بتعيين ثلاثة مؤشرات: الأو ل

.لتخزين العقدة التالية (next) لتتبع العقدة السابقة ، والثالث

.nullإلى previousللعقدة الأولى في القائمة والمؤشر currentابدأ بتعيين المؤش ر

:قم بتكرار الخطوات التالية حتى تصل إلى نهاية القائمة

|  |  |
| --- | --- |
|  | .كمرجع للعقدة التالية nextاحفظ المؤشر  قم بتغيير مؤشر العقدة التالية ليشير إلى العقدة السابقة بدلاً من العقدة  .التالي ة  .قم بتغيير مؤشر العقدة السابقة ليشير إلى العقدة الحالي ة  قم بتغيير مؤشر العقدة الحالية ليشير إلى العقدة التالية المحفوظة في .nextالمؤشر |
| بمجرد الوصول إلى نهاية القائمة، قم بتغيير رأس القائمة ليشير إلى العقدة السابقة .previousفي المؤش ر | |

1. Implement the equals( ) method for the DoublyLinkedList class.

|  |
| --- |
| 1. @Override 2. public boolean equals(Object obj) { 3. if (this == obj) { 4. // If the objects are the same instance, they are equal 5. return true; 6. }   33-   1. if (obj == null || getClass() != obj.getClass()) { 2. // If the objects are of different classes or obj is null, they are not e qual 3. return false; 4. }   38-  39- DoublyLinkedList<E> otherList = (DoublyLinkedList<E>) obj; 40-   1. if (size() != otherList.size()) { 2. // If the lists have different sizes, they are not equal |
| 1. return false; 2. }   45-   1. Node<E> currentNode = head; 2. Node<E> otherNode = otherList.head;   48-   1. while (currentNode != null) { 2. if (!currentNode.data.equals(otherNode.data)) { 3. // If the data in the current nodes is not equal, the lists are not e qual 4. return false; 5. }   54-   1. currentNode = currentNode.next; 2. otherNode = otherNode.next; 3. }   58-  59- // If all elements are equal, the lists are equal 60- return true;  61- } |

27-Implement the rotate() methode in CircularLinkedList class.

|  |
| --- |
| public void rotate() {  if (head == null || head.next == null) {  // If the list is empty or contains only one element, no rotation is needed return;  }    Node<E> lastNode = head; while (lastNode.next != head) { lastNode = lastNode.next;  }    // Move the head to the next node head = head.next;    // Make the last node point to the new head lastNode.next = head;  } |

28-Implement the addFirst() method in CircularLinkedList class.

public void addFirst(E data) { Node<E> newNode = new Node<>(data);

if (head == null) {

// If the list is empty, set the new node as the head and make it point to it self head = newNode; newNode.next = newNode;

} else {

// If the list is not empty, insert the new node before the head and update t he pointers

Node<E> lastNode = head;

|  |
| --- |
| while (lastNode.next != head) { lastNode = lastNode.next;  }  lastNode.next = newNode; newNode.next = head; head = newNode;  }  } |