السؤال الأول

1- ما الخطأ في هذا التعريف: ‏Arrays arrays = new Arrays();‏

الخطأ في هذا التعريف هو استخدام كلمة "Arrays" كاسم للفئة وكذلك استخدام الفئة "Arrays" لإنشاء كائن جديد باستخدام الكلمة

"new".

السؤال الثاني

2-Write and test this method:void reverse(int[] a)// reverses the elements of a[]

لا يمكن كتابة الطريقة المحددة بالشكل المقترح، وذلك لوجود بعض الأخطاء في الأسلوب وأيضاً لعدم توضيح الوظيفة المطلوبة من الطريقة.

إذا كنت تقصد كتابة طريقة لعكس عناصر مصفوفة من نوع int، فيمكنك كتابة هذه الطريقة على النحو التالي:

public static int[] reverseArray(int[] arr) {

int[] reversedArray = new int[arr.length];

int index = 0;

for (int i = arr.length - 1; i >= 0; i--) {

reversedArray[index] = arr[i];

index++;

}

return reversedArray;

}

لاختبار هذه الطريقة، يمكنك استخدام الشيفرة التالية:

public static void main(String[] args) {

int[] array = {1, 2, 3, 4, 5};

int[] reversedArray = reverseArray(array);

System.out.println("Original Array: " + Arrays.toString(array));

System.out.println ("Reversed Array: " + Arrays. toString (reversedArray) );

}

ستقوم الطريقة المذكورة بعكس عناصر المصفوفة المعطاة وإرجاع مصفوفة جديدة تحتوي على العناصر المعكوسة. سيتم طباعة المصفوفة الأصلية والمصفوفة المقلوبة في الإخراج للتحقق من العملية

السؤال الثالث

3- If linked lists are so much better than arrays, why are arrays used at all?

استخدام المصفوفات والقوائم المرتبطة يعتمد على سياق البرمجة والاحتياجات المحددة

الأداء: في العديد من الحالات، القوائم المرتبطة تتفوق على المصفوفات من حيث الأداء، خاصة عندما يتعلق الأمر بالإدخال والحذف في منتصف القائمة. في حالة المصفوفات، يجب نقل جميع البيانات الأخرى بعد العملية. ومع ذلك، المصفوفات لديها فوائد في الوصول العشوائي السريع للعناصر حيث يمكن الوصول إلى بيانات المصفوفة مباشرة عن طريق استخدام الفهرس.

التواتر الذاتي: إذا كان حجم البيانات ثابتًا ومعروفًا مسبقًا، فإن استخدام المصفوفات يمكن أن يكون مناسبًا. يسمح حجم المصفوفة المعروف مسبقًا بتعيين الذاكرة بشكل أكثر كفاءة وقدرة على التخطيط المسبق لها.

بساطة الاستخدام: المصفوفات تعد أكثر بساطة للاستخدام والتعامل معها. يمكن تعريفها وتمريرها بسهولة كمعاملات واستخدامها في العديد من البرامج والحلول. من السهل الوصول إلى عناصر المصفوفة وتعديلها مباشرة.

القيود على الذاكرة: في بعض الحالات، يكون هناك قيود على استخدام الذاكرة، وفي هذه الحالات يمكن أن تكون المصفوفات أكثر فعالية من القوائم المرتبطة. قد تستهلك القوائم المرتبطة مساحة إضافية لتخزين عناوين العناصر ومؤشرات الارتباط.

في النهاية، في حالة تفضيل القوائم المرتبطة على المصفوفات، يمكن استخدامها في الأوقات التي تحتاج فيها إلى أداء عالٍ للإدخال والحذف أكثر من النفاذ العشوائي، أو عندما تحتاج إلى إضافة وحذف عناصر في الحين الذي لا تعرف فيه حجم البيانات مسبقًا

السؤال الرابع

4- Mark the following statements as true or false.

a. In a linked list, the order of the elements is determined by the order in whichthe nodes were created to store the elements.

b. In a linked list, memory allocated for the nodes is sequential.c. A single linked list can be traversed in either direction

d. 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.

e. The head pointer of a linked list cannot be used to traverse the list.

Consider the linked list shown in Figure. Assume that the nodes are in theusual Element-Next form. Use this list to answer Exercises 5 through 8. Ifnecessary, declare additional variables. (Assume that list, p, s, A, and B are references of type Node.)

أ. صحيح

ب. خاطئ

ج. خاطئ

د. صحيح

ه. خاطئ

بالنسبة للسؤال الثاني، للأسف لم يتم إرفاق أي صورة أو شكل. يرجى إرسال الصورة أو توضيح السؤال بشكل أكثر دقة حتى أتمكن من مساعدتك بشكل صحيح

.

السؤال السادس

-1What is the value of each of the following relational expressions?

a. list. getElement() >= 18

b. list.getNext() == A

c. A.getNext().getElement() == 16

d. B.getNext() == (NULL)

e. list. getElement() == 18

أ. قيمة هذه العبارة تعتمد على قيمة الوظيفة getElement() في الصنف قائمة. بدون مزيد من المعلومات، من غير الممكن تحديد القيمة.

ب. هذه العبارة لن تقوم بالتجميع لأن أ ليست متغير أو كائن صالح.

ج. قيمة هذه العبارة تعتمد على تنفيذ الوظائف getNext() و getElement() في الصنف A. بدون مزيد من المعلومات، من غير الممكن تحديد القيمة.

د. قيمة هذه العبارة تعتمد على تنفيذ الوظيفة getNext() في الصنف B. بدون مزيد من المعلومات، من غير الممكن تحديد القيمة.

ه. قيمة هذه العبارة تعتمد على قيمة الوظيفة getElement() في الصنف قائمة. بدون مزيد من المعلومات، من غير الممكن تحديد القيمة.

السؤال السابع

7- Write java Fragment code to do the following:

a- Make A point to the node containing element 23.b- Make list point to the node containing 16.c- Make B point to the last node in the list.d- Make list point to an empty list.

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

f- Create and insert the node with element 10 after the node pointed by A.g- Delete the node with element 23. Also, deallocate the memory occupied by

this node.

A. لجعل A يشير إلى العقدة التي تحتوي على العنصر 23:

java

A = findNode(23); // تجد العقدة التي تحتوي على العنصر 23 وتضعها في A

ب. لجعل القائمة تشير إلى العقدة التي تحتوي على 16:

java

list.head = findNode(16); // تجد العقدة التي تحتوي على العنصر 16 وتعينها كعقدة البداية في القائمة

ج. لجعل B يشير إلى العقدة الأخيرة في القائمة:

java

B = getLastNode(); // تجد العقدة الأخيرة في القائمة وتضعها في B

د. لجعل القائمة تشير إلى قائمة فارغة:

java

list.head = null; // تعين القيمة null لعقدة البداية في القائمة

هـ. لتعيين قيمة العقدة التي تحتوي على 25 إلى 35:

java

Node node = findNode(25); // تجد العقدة التي تحتوي على العنصر 25 وتضعها في node

node.setElement(35); // تعين قيمة 35 لعنصر العقدة

و. لإنشاء وإدراج العقدة مع العنصر 10 بعد العقدة التي يشير إليها A:

java

Node newNode = new Node(10); // إنشاء عقدة جديدة مع العنصر 10

insertAfter(A, newNode); // إدراج العقدة الجديدة بعد العقدة A في القائمة

ح. لحذف العقدة مع العنصر 23 وإلغاء تخصيص الذاكرة التي يشغلها هذه العقدة:

java

deleteNode(23); // حذف العقدة التي تحتوي على العنصر 23 من القائمة وإلغاء تخصيص الذاكرة

السؤال الثامن

What is the output of the following java code?

p = list:

while (p!= NULL){

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

P = p.getNext();}

الناتج من هذا الكود سيكون طباعة عناصر القائمة بالترتيب. وفقًا للكود المعطى:

1. يتم فحص إذا كانت ع (القائمة) مختلفة عن قيمة الـ NULL.

2. إذا كانت قيمة ع غير NULL، يتم طباعة عنصر العقدة الحالية (باستخدام p.getElement())، ثم يتم التحرك إلى العقدة التالية باستخدام p.getNext().

3. يتم تكرار الخطوتين السابقتين حتى يصل p إلى NULL (ينتهي القائمة).

بالتالي، سيتم طباعة عناصر القائمة بالترتيب الذي تم تخزينه في القائمة.

ملاحظة: القائمة ممكن أن تكون قائمة متصلة أو قائمة غير متصلة، وفقًا للرمز الذي لم يتم توضيحه في السؤال. في حالة القائمة الغير متصلة، يمكن أن تحدث أخطاء عندما يحاول الكود الوصول إلى عقدة تحتوي على قيمة NULL.

السؤال التاسع

Show what is produced by the following java code. Assume the node is in theusual 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);

= dnode<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());

b- 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():

a- الكود ينتج عنه الناتج التالي:

10

13

13

بالشرح:

1. يتم إنشاء قائمة جديدة وتعيين قيمة 10 لعنصرها الأول (list.setElement(10)).

2. يتم إنشاء عقدة جديدة وتعيين قيمة 13 لعنصرها (p.setElement(13)).

3. يتم تعيين العقدة الجديدة كـ العقدة التالية للقائمة (list.setNext(p)).

4. يتم إنشاء عقدة مزدوجة جديدة وتعيين قيمة 18 لعنصرها الأول والعقدة السابقة للقائمة كـ العقدة التالية (dnode<E>(18, list.getNext())).

5. يتم تعيين العقدة المزدوجة الجديدة كـ العقدة التالية للقائمة (list.setNext(p)).

6. يتم طباعة قيمة العنصر الأول للقائمة (System.out.println(list.getElement())).

7. يتم طباعة قيمة عنصر العقدة (p.getElement())، وهي 13.

8. يتم تعيين العقدة التالية للعقدة الحالية (p = p.getNext()).

9. يتم طباعة قيمة عنصر العقدة (p.getElement())، وهي 13.

b- الكود ينتج عنه الناتج التالي:

20

28

30

42

بالشرح:

1. يتم إنشاء قائمة جديدة وتعيين قيمة 20 لعنصرها الأول (list.setElement(20)).

2. يتم إنشاء عقدة جديدة وتعيين قيمة 28 لعنصرها (p.setElement(28)).

3. يتم تعيين العقدة الجديدة كـ العقدة التالية للقائمة (list.setNext(p)).

4. يتم إنشاء عقدة جديدة وتعيين قيمة 30 لعنصرها الأول والقائمة كـ العقدة التالية (p.setNext(list)).

5. يتم تعيين العقدة الجديدة كـ القائمة (list = p).

6. يتم إنشاء عقدة جديدة وتعيين قيمة 42 لعنصرها الأول والعقدة التالية للقائمة السابقة كـ العقدة التالية (p.setNext(list.getNext())).

7. يتم تعيين الع

السؤال العاشر

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?

The output of the program segment is:

18

2

38

15

45

25

السؤال الاحداعشر

To insert the value 20 between the values 15 and 24 in a doubly linked list, you can use the following Java code:

java

// Assuming you have a DoublyLinkedList class with appropriate methods and attributes

// Create a new node with the value 20

DoublyLinkedListNode newNode = new DoublyLinkedListNode(20);

// Traverse the doubly linked list to find the nodes with values 15 and 24

DoublyLinkedListNode currentNode = first;

while (currentNode != null) {

if (currentNode.info == 15) {

// Found the node with value 15, update the links

newNode.next = currentNode.next;

newNode.prev = currentNode;

currentNode.next.prev = newNode;

currentNode.next = newNode;

break;

}

currentNode = currentNode.next;

}

// Print the updated doubly linked list

currentNode = first;

while (currentNode != null) {

System.out.println(currentNode.info);

currentNode = currentNode.next;

}

This code assumes that you have a DoublyLinkedList class with appropriate methods and attributes, including a DoublyLinkedListNode class for the nodes in the doubly linked list. It traverses the list to find the node with value 15, then inserts the new node with value 20 between that node and its next node (which should have value 24). Finally, it prints the updated doubly linked list.

السؤال الثاني عشر

Sure! Here's the Java code for the sum method in the SingleLinkedList class:

java

public class SingleLinkedList {

// ... other methods and variables of the class ...

public int sum(Node<Integer> list) {

int sum = 0;

Node<Integer> current = list;

while (current != null) {

sum += current.getData();

current = current.getNext();

}

return sum;

}

}

class Node<T> {

private T data;

private Node<T> next;

public Node(T data) {

this.data = data;

this.next = null;

}

// getter and setter methods for data and next

// ...

}

To test the sum method, you can create an instance of the SingleLinkedList class and call the sum method on a linked list:

java

public static void main(String[] args) {

SingleLinkedList list = new SingleLinkedList();

Node<Integer> node1 = new Node<>(25);

Node<Integer> node2 = new Node<>(45);

Node<Integer> node3 = new Node<>(65); Node<Integer> node4 = new Node<>(85);

node1.setNext(node2);

node2.setNext(node3);

node3.setNext(node4);

int sum = list.sum(node1);

System.out.println("Sum: " + sum); // Output: Sum: 220

}

This code creates a linked list with four nodes containing integers 25, 45, 65, and 85. The sum method is

called on the first node of the list, and the sum of all the integers in the list is printed.

السؤال الثالث عشر

Certainly! Here's the Java code for the removeLast method in the DoublyLinkedList class:

java

public class DoublyLinkedList<E> {

// ... other methods and variables of the class ...

public E removeLast(Node<E> list) {

Node<E> current = list;

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

current = current.getNext();

}

Node<E> previous = current.getPrevious();

previous.setNext(null);

return current.getData();

}

}

class Node<E> {

private E data;

private Node<E> previous;

private Node<E> next;

// constructor, getter and setter methods

// ...

}

To test the removeLast method, you can create an instance of the DoublyLinkedList class and call the removeLast method on a doubly linked list:

java

public static void main(String[] args) {

DoublyLinkedList<Integer> list = new DoublyLinkedList<>();

Node<Integer> node1 = new Node<>(22);

Node<Integer> node2 = new Node<>(44);

Node<Integer> node3 = new Node<>(66);

Node<Integer> node4 = new Node<>(88);

node1.setNext(node2);

node2.setPrevious(node1);

node2.setNext(node3);

node3.setPrevious(node2);

node3.setNext(node4);

node4.setPrevious(node3);

Integer removedData = list.removeLast(node1);

System.out.println("Removed: " + removedData); // Output: Removed: 88

Node<Integer> current = node1;

while (current != null) {

System.out.print(current.getData() + " "); // Output: 22 44 66

current = current.getNext(); }}

This code creates a doubly linked list with four nodes containing integers 22, 44, 66, and 88. The removeLast method is called on the first node of the list, and the data of the removed last node is printed. Finally, the remaining data in the modified list is printed.

السؤال الرابع عشر

Certainly! Here's the Java code for the append method in the SingleLinkedList class:

java

public class SingleLinkedList<E> {

// ... other methods and variables of the class ...

public void append(Node<E> list1, Node<E> list2) {

Node<E> current = list1;

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

current = current.getNext();

}

current.setNext(list2);

}

}

class Node<E> {

private E data;

private Node<E> next;

// constructor, getter and setter methods

// ...

}

To test the append method, you can create an instance of the SingleLinkedList class and call the append method on two single linked lists:

java

public static void main(String[] args) {

SingleLinkedList<Integer> linkedList = new SingleLinkedList<>();

Node<Integer> list1 = new Node<>(22);

Node<Integer> node2 = new Node<>(33);

Node<Integer> node3 = new Node<>(44);

Node<Integer> node4 = new Node<>(55);

list1.setNext(node2);

node2.setNext(node3);

node3.setNext(node4);

Node<Integer> list2 = new Node<>(66);

Node<Integer> node6 = new Node<>(77);

Node<Integer> node7 = new Node<>(88);

Node<Integer> node8 = new Node<>(99);

list2.setNext(node6);

node6.setNext(node7);

node7.setNext(node8);

linkedList.append(list1, list2);

Node<Integer> current = list1;

while (current != null) {

System.out.print(current.getData() + " "); // Output: 22 33 44 55 66 77 88 99

current = current.getNext();

}

}

This code creates two single linked lists, list1 and list2, with four nodes each. The append method is called on list1 and list2, and the modified list1 is printed. The output demonstrates the appended list.

السؤال الخامس عشر

Certainly! Here's the Java code for the concat method in the SingleLinkedList class:

java

public class SingleLinkedList<E> {

// ... other methods and variables of the class ...

public Node<E> concat(Node<E> list1, Node<E> list2) {

Node<E> current = list1;

Node<E> newList = null;

Node<E> newListCurrent = null;

while (current != null) {

Node<E> newNode = new Node<>(current.getData());

if (newList == null) {

newList = newNode;

newListCurrent = newNode;

} else {

newListCurrent.setNext(newNode);

newListCurrent = newNode;

}

current = current.getNext();

}

current = list2;

while (current != null) {

Node<E> newNode = new Node<>(current.getData());

if (newList == null) {

newList = newNode;

newListCurrent = newNode;

} else {

newListCurrent.setNext(newNode);

newListCurrent = newNode;

}

current = current.getNext();

}

return newList;

}

}

class Node<E> {

private E data;

private Node<E> next;

// constructor, getter and setter methods

// ...

}

To test the concat method, you can create an instance of the SingleLinkedList class and call the concat method on two single linked lists:

java

public static void main(String[] args) {

SingleLinkedList<Integer> linkedList = new SingleLinkedList<>();

Node<Integer> list1 = new Node<>(22);

Node<Integer> node2 = new Node<>(33);

Node<Integer> node3 = new Node<>(44);

Node<Integer> node4 = new Node<>(55);

list1.setNext(node2);

node2.setNext(node3);

node3.setNext(node4);

Node<Integer> list2 = new Node<>(66);

Node<Integer> node6 = new Node<>(77);

Node<Integer> node7 = new Node<>(88);

Node<Integer> node8 = new Node<>(99);

list2.setNext(node6);

node6.setNext(node7);

node7.setNext(node8);

Node<Integer> concatenatedList = linkedList.concat(list1, list2);

Node<Integer> current = concatenatedList;

while (current != null) {

System.out.print(current.getData() + " "); // Output: 22 33 44 55 66 77 88 99

current = current.getNext();

}

}

This code creates two single linked lists, list1 and list2, with four nodes each. The concat method is called on list1 and list2, and the returned concatenated list is printed. The output demonstrates the concatenated list, which is a completely independent list from list1 and list2.

السؤال السادس عشر

تفضل الطريقة المطلوبة:

java

public void swap(Node<E> list, int i, int j) {

if (list == null) {

throw new IllegalArgumentException("List cannot be null");

}

Node<E> nodeI = getNodeAtIndex(list, i);

Node<E> nodeJ = getNodeAtIndex(list, j);

if (nodeI == null || nodeJ == null) {

throw new IndexOutOfBoundsException("Invalid index");

}

// Swap the values of the nodes

E temp = nodeI.data;

nodeI.data = nodeJ.data;

nodeJ.data = temp;

}

private Node<E> getNodeAtIndex(Node<E> list, int index) {

if (index < 0) {

return null;

}

Node<E> current = list;

int count = 0;

while (current != null && count < index) {

current = current.next;

count++;

}

return current;

}

للاختبار، يمكنك استخدام الشفرة التالية:

java

DoublyLinkedList<Integer> list = new DoublyLinkedList<>();

list.add(22);

list.add(33);

list.add(44);

list.add(55);

list.add(66);

list.add(77);

list.add(88);

list.add(99);

System.out.println("Original list: " + list);

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

System.out.println("Swapped list: " + list);

ستظهر النتيجة التالية:

Original list: (22, 33, 44, 55, 66, 77, 88, 99)

Swapped list: (22, 33, 77, 55, 66, 44, 88, 99)

يرجى ملاحظة أن الطريقة المقترحة تستخدم فئة Node و DoublyLinkedList المفترضة، ولذا يجب التأكد من وجود تعريف لهذه الفئتين قبل استخدام الشفرة.

السؤال السابع عشر

لعكس قائمة مرتبطة منفردة L باستخدام مقدار ثابت فقط من المساحة الإضافية، يمكن اتباع الخوارزمية التالية:

1. إنشاء ثلاثة مؤشرات: الأول يشير إلى العقدة الحالية، والثاني يشير إلى العقدة السابقة للعقدة الحالية، والثالث يشير إلى الالعقدة القادمة في القائمة.

2. قم بتهيئة المؤشرات بالطريقة التالية:

- المؤشر الحالي = رأس القائمة (أول عنصر في القائمة)

- المؤشر السابق = null (لا يشير إلى أي عقدة حتى الآن)

- المؤشر القادم = null (لا يشير إلى أي عقدة حتى الآن)

3. قم بتحديث المؤشرات بالتالي:

- المؤشر القادم = المؤشر الحالي.next (العقدة التالية)

- المؤشر الحالي.next = المؤشر السابق (قم بعكس اتجاه العقدة)

- المؤشر السابق = المؤشر الحالي (حدث المؤشر السابق ليكون العقدة الحالية)

- المؤشر الحالي = المؤشر القادم (حدث المؤشر الحالي ليكون العقدة التالية)

4. كرر الخطوة 3 حتى يصبح المؤشر الحالي يشير إلى null (نهاية القائمة).

5. يتم الانتهاء من عكس القائمة. يمكن استخدام المؤشر السابق الذي يشير إلى رأس القائمة المعكوسة.\

يجب ملاحظة أنه لإجراء عملية عكس القائمة باستخدام مقدار ثابت من المساحة الإضافية، يتعين تنفيذ الخوارزمية في مكانها وتعديل روابط العقد الموجودة بدلاً من إنشاء قائمة جديدة

السؤال الثامن عشر

بالتأكيد! سأقوم بتطبيق طريقة equals() لفئة DoublyLinkedList وفقًا للمتطلبات التالية:

java

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

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

return false;

}

DoublyLinkedList other = (DoublyLinkedList) obj;

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

return false;

}

Node currentNode = head;

Node otherCurrentNode = other.head;

while (currentNode != null) {

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

return false;

}

currentNode = currentNode.next;

otherCurrentNode = otherCurrentNode.next;

}

return true;

}

في هذا الأسلوب، نقوم بتحقيق عدة شروط:

1. نقوم بفحص ما إذا كانت الكائنات المقارنة هي نفس الكائنات باستخدام العامل ==. إذا كانت كذلك، فهما متساويان ونعيد true.

2. نقوم بفحص ما إذا كان الكائن المقارن null أو من نوع مختلف عن فئة DoublyLinkedList. إذا كان أيًا من ذلك صحيحًا، فهما غير متساوين ونعيد false.

3. نقوم بتحقيق ما إذا كانت حجم القوائم المزدوجة المقارنة متساوية. إذا لم تكن متساوية، فهما غير متساويين ونعيد false.

4. نقوم بتحقيق ما إذا كانت العقد في القوائم المزدوجة المقارنة متساوية بالترتيب. إذا لم تكن متساوية، فهما غير متساويين ونعيد false.

5. إذا تم تجاوز جميع الشروط ولم نعثر على أي اختلافات، فهما متساويان ونعيد true.

يمكنك استخدام هذا الأسلوب للمقارنة بين كائنين من فئة DoublyLinkedList لمعرفة ما إذا كانت محتوياتهما متساوية أم لا.

السؤال التاسع عشر

بالتأكيد! سأقوم بتنفيذ طريقة التدوير (rotate()) في فئة القائمة المرتبطة الدائرية (CircularLinkedList) وفقًا للمتطلبات التالية:

java

public void rotate() {

if (head != null) {

Node lastNode = head;

while (lastNode.next != head) {

lastNode = lastNode.next;

}

lastNode.next = head.next;

head.next = head.next.next;

lastNode.next.next = head;

head = head.next;

}

}

في هذا الأسلوب، نقوم بتنفيذ الخطوات التالية لتنفيذ عملية التدوير:

1. نقوم بفحص ما إذا كان الرأس (head) ليس فارغًا. إذا كان فارغًا، فلا حاجة لتنفيذ عملية التدوير.

2. نقوم بالتحرك إلى أخر عقدة في القائمة المرتبطة المدورة ونخزنها في متغير lastNode.

3. نقوم بتعديل الروابط لتنفيذ عملية التدوير. نقوم بتعديل رابط العقدة الأخيرة المؤدي إلى الرأس ليكون الرابط الذي يأتي بعده بالقائمة المرتبطة.

4. نقوم بتعديل رابط العقدة الأولى (الرأس) لتكون العقدة الثانية في القائمة المرتبطة.

5. نقوم بتعديل رابط العقدة التي تأتي بعد العقدة الثانية في القائمة المرتبطة لتصبح الرأس.

بهذه الطريقة، يمكنك استخدام طريقة rotate() لتنفيذ عملية التدوير في القائمة المرتبطة الدائرية.

السؤال العشرين

بالتأكيد! سأقوم بتنفيذ طريقة addFirst() في فئة CircularLinkedList وفقًا للمتطلبات التالية:

java

public void addFirst(Object data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

newNode.next = head;

} else {

Node temp = head;

while (temp.next != head) {

temp = temp.next;

}

newNode.next = head;

temp.next = newNode;

head = newNode;

}

}

في هذا الأسلوب، نقوم بتنفيذ الخطوات التالية لإضافة عقدة جديدة في بداية القائمة المرتبطة المدورة:

1. إنشاء عقدة جديدة باستخدام القيمة المعطاة (data) وتخزينها في متغير newNode.

2. نقوم بفحص ما إذا كان الرأس (head) فارغًا. إذا كان الرأس فارغًا، يعني ذلك أن القائمة المرتبطة فارغة ويجب أن تضيف العقدة الجديدة كعقدة وحيدة وتجعل الروابط تشير إليها.

3. إذا لم يكن الرأس فارغًا، نقوم بتحديث الروابط لإضافة العقدة الجديدة في بداية القائمة المرتبطة. نقوم بتحديث رابط العقدة الأخيرة في القائمة لتشير إلى العقدة الجديدة، ثم نحدث رابط العقدة الجديدة لتشير إلى الرأس الحالي، وأخيرًا نجعل الرأس يشير إلى العقدة الجديدة.

بهذه الطريقة، يمكنك استخدام طريقة

addFirst() لإضافة عقدة جديدة في بداية القائمة المرتبطة الدائرية.

السؤال الخامس

أ. System.out.println(list.getElement());

هذه العبارة تنقل بطريقة آمنة وتطبع العنصر الحالي في القائمة المرتبطة المدورة التي يسمح لها بالوصول إلى العنصر الحالي. سيتم طباعة العنصر الموجود في العقدة التي يشير إليها الرأس (head) الحالي للقائمة.

ب. System.out.println( A.getElement());

هذه العبارة تنقل بطريقة غير آمنة وتحاول الوصول إلى العنصر الحالي في العقدة المسماة "A". ومع ذلك، لا يوجد سياق للتعامل مع A في العبارة المعطاة، وبالتالي فإن هذه العبارة لن تؤدي إلى نتيجة صحيحة وسيتم إصدار خطأ في وقت التشغيل.

ج. System.out.println( list.getNext().getNext().getElement());

هذه العبارة تنقل بطريقة آمنة وتطبع العنصر الحالي في العقدة التي هي العقدة الموجودة بعد عقدة الـ "next". ستتم الوصول إلى العقدة التي تأتي بعد التالي (next) وستستخرج العنصر الذي يحتويه وتقوم بطباعته.

د. System.out.println( B.getNext().getElement());

هذه العبارة تنقل بطريقة غير آمنة وتحاول الوصول إلى العقدة التالية بعد العقدة المسماة "B" ومن ثم الوصول إلى العنصر الحالي في تلك العقدة. ومع ذلك، لا يوجد سياق للتعامل مع B في العبارة المعطاة، وبالتالي فإن هذه العبارة لن تؤدي إلى نتيجة صحيحة وسيتم إصدار خطأ في وقت التشغيل.