1. Describe a method for finding the middle node of a doubly linked list with header and trailer sentinels by “link hopping,” and without relying on explicit knowledge of the size of the list. In the case of an even number of nodes, report the node slightly left of center as the “middle.”

١,٦

لهدف هو إيجاد العقدة الوسطى في قائمة مرتبطة مزدوجة تحتوي على عقدتي رأسية (Header) وذيلية (Trailer) باستخدام تقنية "القفز بين الروابط" دون الاعتماد على معرفة الحجم الكلي للقائمة. إذا كان عدد العقد زوجياً، يتم اختيار

الخطوات:

١. تهيئة مؤشرين:

استخدم مؤشرين، front وback:

المؤشر front يبدأ من العقدة الأولى بعد الرأسية: header.next.

المؤشر back يبدأ من العقدة الأخيرة قبل الذيلية: trailer.prev.

٢. اجتياز القائمة:

قم بإنشاء حلقة تمرر فيها المؤشرين خطوة واحدة في كل مرة:

حرّك المؤشر front للأمام باستخدام: front = front.next.

حرّك المؤشر back للخلف باستخدام: back = back.prev.

٣. شرط الإنهاء:

تتوقف الحلقة عندما يتحقق أحد الشروط التالية:

إذا كان front == back: يعني ذلك أن المؤشرين التقيا عند العقدة الوسطى (عدد العقد فردي).

إذا كان front.next == back: يعني أن المؤشرين أصبحا متجاورين، وبالتالي back يشير إلى العقدة التي تقع على يسار المركز (عدد العقد زوجي).

٤. إرجاع النتيجة:

أرجع العقدة التي يشير إليها المؤشر back باعتبارها العقدة الوسطى.

مثال عملي:

لنفترض قائمة مرتبطة تحتوي على العقد التالية: Header <-> 1 <-> 2 <-> 3 <-> 4 <-> 5 <-> Trailer

الحالة الأولية:

front يشير إلى ١، وback يشير إلى ٥.

التكرار الأول:

حرّك front إلى ٢، وback إلى ٤.

التكرار الثاني:

حرّك front إلى ٣، وback إلى ٣.

الإيقاف:

عندما front == back، العقدة الوسطى هي ٣.

لنفترض قائمة مرتبطة تحتوي على العقد التالية: Header <-> 1 <-> 2 <-> 3 <-> 4 <-> Trailer

الحالة الأولية:

front يشير إلى ١، وback يشير إلى ٤.

التكرار الأول:

حرّك front إلى ٢، وback إلى ٣.

الإيقاف:

عندما front.next == back، العقدة الوسطى هي ٢، وهي العقدة التي تقع على يسار المركز.

النقاط الرئيسية:

تعتمد الطريقة فقط على التنقل بين العقد ولا تتطلب معرفة مسبقة بحجم القائمة.

الكفاءة الزمنية للطريقة هي O(n) لأن كل مؤشر يمر عبر نصف القائمة فقط.

في القوائم المرتبطة المزدوجة، هذه الطريقة فعّالة بسبب وجود المؤشرين next وprev.

```
A: instance variable
} ()A: public int size
int count = 0;
Node current = header.next

} while (current != trailer)
count++;
current = current.next;
{
return count;
{
```


:Algorithm Concatenate(L, M)

:if L is empty

L'.header.next = M.header.next

M.header.next.prev = L'.header

L'.trailer.prev = M.trailer.prev

M.trailer.prev.next = L'.trailer

return

:if M is empty

L'.header.next = L.header.next

L.header.next.prev = L'.header

L'.trailer.prev = L.trailer.prev

L.trailer.prev.next = L'.trailer

return

Link the last node of L to the first node of M //

```
L.trailer.prev.next = M.header.next
M.header.next.prev = L.trailer.prev

Update the trailer of L' to point to M's trailer //
L'.trailer.prev = M.trailer.prev
M.trailer.prev.next = L'.trailer

Update the header of L' to point to L's header //
L'.header.next = L.header.next
L.header.next.prev = L'.header
                                                    ٥
} <public class DoublyLinkedList<E
    } <private class Node<E
        ;E data
        ;Node<E> next
        ;Node<E> prev

} Node(E data, Node<E> next, Node<E> prev)
        ;this.data = data
        ;this.next = next
        ;this.prev = prev
                                {
                                  {
    ;private Node<E> sentinel
        ;private int size

    } ()public DoublyLinkedList
;sentinel = new Node<>(null, null, null)
        ;sentinel.next = sentinel
        ;sentinel.prev = sentinel
            ;size = 0
```

```
{

} ()public int size
    ;return size
{
} ()public boolean isEmpty
    ;return size == 0
{
} public void addFirst(E data)
;Node<E> newNode = new Node<>(data, sentinel.next, sentinel)
;sentinel.next.prev = newNode
;sentinel.next = newNode
;++size
{

} public void addLast(E data)
;Node<E> newNode = new Node<>(data, sentinel, sentinel.prev)
;sentinel.prev.next = newNode
;sentinel.prev = newNode
;++size
{
} ()public E removeFirst
;if (isEmpty()) throw new IllegalStateException("List is empty")
;Node<E> first = sentinel.next
;sentinel.next = first.next
;first.next.prev = sentinel
;--size
;return first.data
{
```

```
    } ()public E removeLast
;if (isEmpty()) throw new IllegalStateException("List is empty")
;Node<E> last = sentinel.prev
;sentinel.prev = last.prev
;last.prev.next = sentinel
;--size
;return last.data
    {

    } ()public E getFirst
;if (isEmpty()) throw new IllegalStateException("List is empty")
;return sentinel.next.data
    {

    } ()public E getLast
;if (isEmpty()) throw new IllegalStateException("List is empty")
;return sentinel.prev.data
    {
      {

} <public class DoublyLinkedList<E
    } <private class Node<E
;E data
;Node<E> next
;Node<E> prev

} Node(E data, Node<E> next, Node<E> prev)
;this.data = data
;this.next = next
;this.prev = prev
    {
```

```
{
    ;private Node<E> sentinel
    ;private int size

    } ()public DoublyLinkedList
        ;sentinel = new Node<>(null, null, null)
        ;sentinel.next = sentinel
        ;sentinel.prev = sentinel
        ;size = 0
    {

    } ()public int size
        ;return size
    {
    } ()public boolean isEmpty
        ;return size == 0
    {
    } public void addFirst(E data)
        ;Node<E> newNode = new Node<>(data, sentinel.next, sentinel)
        ;sentinel.next.prev = newNode
        ;sentinel.next = newNode
        ;++size
    {

    } public void addLast(E data)
        ;Node<E> newNode = new Node<>(data, sentinel, sentinel.prev)
        ;sentinel.prev.next = newNode
        ;sentinel.prev = newNode
        ;++size
    {
    } ()public E removeFirst
```

```
;if (isEmpty()) throw new IllegalStateException("List is empty")
                                                ;Node<E> first = sentinel.next
                                                    ;sentinel.next = first.next
                                                    ;first.next.prev = sentinel
                                                                ;--size
                                                        ;return first.data
                                                                    {


                                                    } ()public E removeLast
;if (isEmpty()) throw new IllegalStateException("List is empty")
                                                ;Node<E> last = sentinel.prev
                                                    ;sentinel.prev = last.prev
                                                    ;last.prev.next = sentinel
                                                                ;--size
                                                        ;return last.data
                                                                    {


                                                     } ()public E getFirst
;if (isEmpty()) throw new IllegalStateException("List is empty")
                                                        ;return sentinel.next.data
                                                                    {


                                                      } ()public E getLast
;if (isEmpty()) throw new IllegalStateException("List is empty")
                                                        ;return sentinel.prev.data
                                                                    {
                                                                      {


                                                } <public class CircularDoublyLinkedList<T
                                                        } private class Node
```

```
                                        ;T data
                                    ;Node next
                                    ;Node prev

                                } Node(T data)
                            ;this.data = data
                                        {
                                          {

                                ;private Node head
                                    ;private int size
                                } public void add(T data)
                        ;Node newNode = new Node(data)

                                } if (head == null)
                                ;head = newNode
                                ;head.next = head
                                ;head.prev = head
                                        } else {
                            ;Node tail = head.prev

                                ;tail.next = newNode
                                ;newNode.prev = tail

                            ;newNode.next = head
                            ;head.prev = newNode
                                        {

                                        ;++size
                                            {
```

```
} public void remove(T data)
    ;if (head == null) return

    ;Node current = head
    ;int count = 0

    } do
} if (current.data.equals(data))
    } if (size == 1)
    ;head = null
    } else {
;Node prevNode = current.prev
;Node nextNode = current.next

;prevNode.next = nextNode
;nextNode.prev = prevNode

} if (current == head)
;head = nextNode
    {
        {
    ;--size
    ;return
        {
;current = current.next
    ;++count
;(while (current != head && count < size {
                {

} ()public void rotate
} if (head != null)
```

```
                                        ;head = head.next
                                                    {
                                                      {


                                    } ()public void rotateBackward
                                          } if (head != null)
                                        ;head = head.prev
                                                    {
                                                      {
                                     } ()public void display
                                       } if (head == null)
                                ;System.out.println("List is empty.")
                                                ;return
                                                    {

                                        ;Node current = head
                                                  } do
                                ;System.out.print(current.data + " ")
                                        ;current = current.next
                                        ;(while (current != head {
                                            ;()System.out.println
                                                    {


                                                } ()public int size
                                                   ;return size
                                                    {


                                 } public static void main(String[] args)
                ;()<>CircularDoublyLinkedList<Integer> list = new CircularDoublyLinkedList
```

```
;(١)list.add
;(٨)list.add
;(٦)list.add
;(٤)list.add
;System.out.println("Original List:")
;()list.display
;()list.rotate
;System.out.println("After rotate():")
;()list.display
;()list.rotateBackward
;System.out.println("After rotateBackward():")
;()list.display
;(٣)list.remove
;System.out.println("After removing 3:")
;()list.display
                                                    {
                                                      {
} <public class CircularlyLinkedList<E
    ;private Node<E> tail = null
        ;private int size = 0

    {} ()public CircularlyLinkedList


    } ()public int size
        ;return size
                                                    {



    } ()public boolean isEmpty
        ;return size == 0
                                                    {
```

```
} ()public E first
;if (isEmpty()) return null
;()return tail.getNext().getElement
{

} ()public E last
;if (isEmpty()) return null
;()return tail.getElement
{
} ()public void rotate
;()if (tail != null) tail = tail.getNext
{
} public void addFirst(E e)
} if (size == 0)
;tail = new Node<>(e, null)
;tail.setNext(tail)
} else {
;tail.setNext(new Node<>(e, tail.getNext()))
{
;++size
{
} public void addLast(E e)
;addFirst(e)
;()tail = tail.getNext
{
} ()public E removeFirst
;if (isEmpty()) return null
;()Node<E> head = tail.getNext
;if (head == tail) tail = null
;else tail.setNext(head.getNext())
;--size
```

```
        ;()return head.getElement
                                                    {


                                        } ()public void printList
                                            } if (isEmpty())
                                    ;("[]")System.out.println
                                                ;return
                                                    {
    البداية عند الرأس // ;()Node<E> current = tail.getNext
                                                ;("]")System.out.print
                                                    } do
System.out.print(current.getElement() + (current.getNext() != tail.getNext() ? ", " :
                                                        ;""))
                                            ;()current = current.getNext
                                        ;(()while (current != tail.getNext {
                                            ;("[")System.out.println
                                                    {


                                                Node الفئة الداخلية //
                                        } <private static 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 Node<E> getNext
        ;return next
    {

    } public void setNext(Node<E> next)
        ;this.next = next
    {
{

    } public static void main(String[] args)
;()<>CircularlyLinkedList<Integer> list = new CircularlyLinkedList

// إضافة عناصر إلى القائمة
;(١٨)list.addLast
;(٢٩)list.addLast
;(٣٧)list.addLast

;("القائمة الأصلية:")System.out.println
;()list.printList

// إضافة عنصر في البداية
;(٠)list.addFirst
;("بعد إضافة ٠ في البداية:")System.out.println
;()list.printList

// إزالة العنصر الأول
;("إزالة العنصر الأول: " + list.removeFirst())System.out.println
;("بعد الإزالة:")System.out.println
;()list.printList
```

```
// تدوير القائمة
list.rotate();
System.out.println("بعد التدوير:");
list.printList();
{
{
} <public class CircularlyLinkedList<E
Node<E> tail;


} <static class Node<E
E element;
Node<E> next;

} public Node(E element, Node<E> next)
this.element = element;
this.next = next;
{
{

} ()public CircularlyLinkedList
tail = null;
{


} ()public int size
} if (tail == null)
return 0;
{
```

```
                                        ;int count = 1
                                    ;Node<E> current = tail.next



                                    } while (current != tail)
                                            ;++count
                                    ;current = current.next
                                                {

                                            ;return count
                                                {



                                } public void addLast(E element)
                            ;Node<E> newNode = new Node<>(element, null)
                                            } if (tail == null)
                                        ;newNode.next = newNode
                                            ;tail = newNode
                                                } else {
                                        ;newNode.next = tail.next
                                            ;tail.next = newNode
                                            ;tail = newNode
                                                {
                                                    {



                                            } ()public void printList
                                                } if (tail == null)
                                        ;("[]")System.out.println
                                                ;return
                                                {
```

```
                                                    ;Node<E> current = tail.next
                                                        ;("]")System.out.print
                                                                    } do
;System.out.print(current.element + (current.next != tail.next ? ", " : ""))
                                                        ;current = current.next
                                                    ;(while (current != tail.next {
                                                        ;("[")System.out.println
                                                                    {

                                                } public static void main(String[] args)
                                        ;()<>CircularlyLinkedList<Integer> list = new CircularlyLinkedList
                                                                ;(٣)list.addLast
                                                                ;(٩)list.addLast
                                                                ;(٨)list.addLast

                                                    ;System.out.println("List:")
                                                                ;()list.printList

                                                ;System.out.println("Size: " + list.size())
                                                                    {
                                                                      {
                                        } public class CircularlyLinkedList<E> implements Cloneable
                                                                ;Node<E> tail
                                                            ;private int size



                                                        } <static class Node<E
                                                                ;E element
                                                            ;Node<E> next

                                                } public Node(E element, Node<E> next)
```

```
                ;this.element = element
                    ;this.next = next
                                {
                                  {

                } ()public CircularlyLinkedList
                            ;tail = null
                              ;size = 0
                                  {


            } public void addLast(E element)
        ;Node<E> newNode = new Node<>(element, null)
                            } if (tail == null)
                        ;newNode.next = newNode
                            ;tail = newNode
                                } else {
                     ;newNode.next = tail.next
                        ;tail.next = newNode
                            ;tail = newNode
                                {
                              ;++size
                                  {


                    } ()public int size
                       ;return size
                                  {

                              Override@
                } ()public CircularlyLinkedList<E> clone
```

```
;()<>CircularlyLinkedList<E> clonedList = new CircularlyLinkedList
                                                    } if (size == 0)
                                                ;return clonedList
                                                            {

                                                ;Node<E> current = tail.next
                                                            } do
                                            ;clonedList.addLast(current.element)
                                                ;current = current.next
                                            ;(while (current != tail.next {

                                                ;return clonedList
                                                        {

                                                } ()public void printList
                                                    } if (tail == null)
                                                ;("[]")System.out.println
                                                        ;return
                                                        {
                                                ;Node<E> current = tail.next
                                                    ;("]")System.out.print
                                                            } do
;System.out.print(current.element + (current.next != tail.next ? ", " : ""))
                                                ;current = current.next
                                            ;(while (current != tail.next {
                                                ;("[")System.out.println
                                                        {

                                            } public static void main(String[] args)
                                    ;()<>CircularlyLinkedList<Integer> list = new CircularlyLinkedList
                                                ;(١)list.addLast
```

```
;(٢)list.addLast
;(٣)list.addLast

;System.out.println("Original List:")
;()list.printList

;()CircularlyLinkedList<Integer> clonedList = list.clone

;System.out.println("Cloned List:")
;()clonedList.printList



;(٤)list.addLast
;System.out.println("Modified Original List:")
;()list.printList

;System.out.println("Cloned List After Modification:")
;()clonedList.printList
{
{
} <public class CircularlyLinkedList<E
Node<E> tail; // Reference to the tail node
private int size; // Size of the list

Inner Node class //
} <static class Node<E
;E element
;Node<E> next

} public Node(E element, Node<E> next)
;this.element = element
```

```
                                        ;this.next = next
                                                {
                                                  {

                                    } ()public CircularlyLinkedList
                                                ;tail = null
                                                  ;size = 0
                                                {

                                Method to add an element at the end of the list //
                                } public void addLast(E element)
                            ;Node<E> newNode = new Node<>(element, null)
                                                } if (tail == null)
                                            ;newNode.next = newNode
                                                ;tail = newNode
                                                } else {
                                            ;newNode.next = tail.next
                                                ;tail.next = newNode
                                                ;tail = newNode
                                                        {
                                                    ;++size
                                                        {

                                        Method to get the size of the list //
                                                } ()public int size
                                                  ;return size
                                                {

                                Method to split the list into two circularly linked lists //
                                } ()public CircularlyLinkedList<E>[] splitList
                                                } if (size % 2 != 0)
```

```
;throw new IllegalArgumentException("The list size must be even to split.")
                                                                                {

                                                    ;()<>CircularlyLinkedList<E> L1 = new CircularlyLinkedList
                                                    ;()<>CircularlyLinkedList<E> L2 = new CircularlyLinkedList

                                                        Node<E> slow = tail.next; // Start at head
                                                        ;Node<E> fast = tail.next

                                                                    Find the midpoint //
                                                    } while (fast.next != tail.next && fast.next.next != tail.next)
                                                                        ;slow = slow.next
                                                                    ;fast = fast.next.next
                                                                                {

                                                                    Split the list //
                                                        L1.tail = slow; // Tail of the first half
                                                        L2.tail = tail; // Tail of the second half

                                                    Update next pointers to make two circular lists //
                                                    L1.tail.next = tail.next; // First half points to its head
                                                    L2.tail.next = slow.next; // Second half points to its head

                                                Update tail of the original list to point to the new L1 head //
                                                        ;slow.next = L1.tail.next

                                                                Calculate sizes for each list //
                                                                    ;L1.size = size / 2
                                                                    ;L2.size = size / 2

                                                                    Return the two lists //
```

```
        return new CircularlyLinkedList[]{L1, L2};
    }

    // Helper method to print the list
    public void printList() {
        if (tail == null) {
            System.out.println("[]");
            return;
        }
        Node<E> current = tail.next; // Start at the head
        System.out.print("[");
        do {
            System.out.print(current.element + (current.next != tail.next ? ", " : ""));
            current = current.next;
        } while (current != tail.next);
        System.out.println("]");
    }

    public static void main(String[] args) {
        CircularlyLinkedList<Integer> list = new CircularlyLinkedList<>();
        list.addLast(١);
        list.addLast(٢);
        list.addLast(٣);
        list.addLast(٤);
        list.addLast(٥);
        list.addLast(٦);

        System.out.println("Original List:");
        list.printList();

        CircularlyLinkedList<Integer>[] splitLists = list.splitList();
```

```
;System.out.println("First Half:")
;()splitLists[0].printList

;System.out.println("Second Half:")
;()splitLists[1].printList
{
{
} <public class CircularlyLinkedList2<E
Node<E> tail; // الإشارة إلى العقدة الأخيرة
private int size; // عدد العقد في القائمة

// الفئة الداخلية Node لتعريف العقدة
} <static class Node<E
;E element
;Node<E> next

} public Node(E element, Node<E> next)
;this.element = element
;this.next = next
{
{

} ()public CircularlyLinkedList2
;tail = null
;size = 0
{

// طريقة لإضافة عنصر إلى نهاية القائمة
} public void addLast(E element)
;Node<E> newNode = new Node<>(element, null)
```

```
(tail == null) { // إذا كانت القائمة فارغة
    newNode.next = newNode;
    tail = newNode;
} else {
    newNode.next = tail.next;
    tail.next = newNode;
    tail = newNode;
}
size++;
}

// طريقة للحصول على حجم القائمة
public int size() {
    return size;
}

// دالة للتحقق من تساوي قائمتين دائريتين
public static <E> boolean areCircularListsEqual(CircularlyLinkedList<E> L,
                                                CircularlyLinkedList<E> M) {
    // خطوة ١: تحقق من الأحجام
    if (L.size() != M.size()) {
        return false; // الأحجام مختلفة، لا يمكن أن تكون القوائم متساوية
    }

    // خطوة ٢: التعامل مع القوائم الفارغة
    if (L.size() == 0) {
        return true; // كلا القائمتين فارغتين
    }

    // خطوة ٣: تحديد نقطة البداية في L
    CircularlyLinkedList2.Node<E> startL = L.tail.next; // رأس القائمة L
```

```
// خطوة ٤: مقارنة التسلسلات مع جميع التدويرات الممكنة في M
CircularlyLinkedList2.Node<E> currentM = M.tail.next; // البداية من رأس القائمة M
} for (int i = 0; i < M.size(); i++)
} if (compareSequences(startL, currentM, L.size()))
return true; // تم العثور على تطابق
{
currentM = currentM.next; // تجربة التدوير التالي في M
{

// خطوة ٥: لم يتم العثور على تطابق
return false;
{

// دالة مساعدة لمقارنة التسلسلات
private static <E> boolean compareSequences(CircularlyLinkedList2.Node<E> startL,
CircularlyLinkedList2.Node<E> startM,
int size) {
CircularlyLinkedList2.Node<E> currentL = startL;
CircularlyLinkedList2.Node<E> currentM = startM;

} for (int i = 0; i < size; i++)
if ((currentL.element == null && currentM.element != null)
|| ((currentL.element != null && !currentL.element.equals(currentM.element)) {
return false; // العناصر ليست متساوية
{
currentL = currentL.next;
currentM = currentM.next;
{

return true; // كل العناصر متطابقة
{
```

```
// دالة main لاختبار القوائم
} public static void main(String[] args)
;()<>CircularlyLinkedList2<Integer> list1 = new CircularlyLinkedList2
;()<>CircularlyLinkedList2<Integer> list2 = new CircularlyLinkedList2

// إضافة العناصر إلى القائمة الأولى
;(١)list1.addLast
;(٢)list1.addLast
;(٣)list1.addLast

// إضافة العناصر إلى القائمة الثانية
;(٢)list2.addLast
;(٣)list2.addLast
;(١)list2.addLast

// مقارنة القائمتين
(areCircularListsEqual(list1, list2)); // المخرجات: true

// إضافة عنصر جديد إلى القائمة الثانية
;(٤)list2.addLast
(areCircularListsEqual(list1, list2)); // المخرجات: false
{
{

} <public class CircularlyLinkedList<E
;private Node<E> tail
;private int size

} <private static class Node<E
;private E element
```

```
                                        ;private Node<E> next

                                    } public Node(E element, Node<E> next)
                                            ;this.element = element
                                                ;this.next = next
                                                        {
                                                          {

                                    } ()public CircularlyLinkedList
                                                    ;tail = null
                                                     ;size = 0
                                                        {


                                  } public void addLast(E element)
                            ;Node<E> newNode = new Node<>(element, null)
                                                  } if (tail == null)
                                               ;newNode.next = newNode
                                                    ;tail = newNode
                                                       } else {
                                             ;newNode.next = tail.next
                                                ;tail.next = newNode
                                                    ;tail = newNode
                                                        {
                                                    ;++size
                                                          {


                                                   Override@
                                    } public boolean equals(Object obj)
                                          ;if (this == obj) return true
```

```
if (obj == null || getClass() != obj.getClass()) return false; // incompatible types

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

;if (this.size != other.size) return false

;if (size == 0) return true

;Node<E> current1 = this.tail.next
;Node<?> current2 = other.tail.next
} for (int i = 0; i < size; i++)

;if (current1.element == null && current2.element != null) return false
} if (current1.element != null && !current1.element.equals(current2.element))
;return false
{
;current1 = current1.next
;current2 = current2.next
{

;return true
{

} public static void main(String[] args)
;()<>CircularlyLinkedList<Integer> list1 = new CircularlyLinkedList
;()<>CircularlyLinkedList<Integer> list2 = new CircularlyLinkedList

;(١)list1.addLast
```

```
                                                    ;(٢)list1.addLast
                                                    ;(٣)list1.addLast

                                                    ;(١)list2.addLast
                                                    ;(٢)list2.addLast
                                                    ;(٣)list2.addLast

                                          ;System.out.println(list1.equals(list2))

                                                    ;(٤)list2.addLast
                                          ;System.out.println(list1.equals(list2))
                                                                  {
                                                                    {
```