```
/**
*
* @Modified Vahdat Abdelzad
*/
import java.util.NoSuchElementException;

public class OrderedList implements OrderedStructure {

    // Implementation of the doubly linked nodes (nested-class)

    private static class Node {

        private Comparable value;
        private Node previous;
        private Node next;

        private Node(Comparable value, Node previous, Node next) {
            this.value = value;
            this.previous = previous;
            this.next = next;
        }
    }

    // Instance variables

    private Node head;

    // Representation of the empty list.

    public OrderedList() {
        head = new Node(null, null, null); // dummy node
        head.next = head; // circular to the right
        head.previous = head; // circular to left
    }

    // Calculates the size of the list

    public int size() {
        Node current = head.next; // also works for the empty list
        int count = 0;
        while (current != head) {
            current = current.next;
            count++;
        }
        return count;
    }

    // This implementation does not call size() to determine
    // if pos is valid; therefore, the list is only traversed
    // once.

    public Object get(int pos) {

        if (pos < 0) {
            throw new IndexOutOfBoundsException(Integer.toString(pos));
        }

        Node current = head.next;
```

```
    for (int i=0; i<pos; i++) {
        if (current.next == head) {
            throw new IndexOutOfBoundsException(Integer.toString(pos));
        }
        current = current.next;
    }

    return current.value;
}

// Adding an element while preserving the order

@SuppressWarnings("unchecked")

public boolean add(Comparable o) {

    if (o == null) {
        throw new IllegalArgumentException("null");
    }

    Node before = head;

    while (before.next != head && before.next.value.compareTo(o) < 0) {
        before = before.next;
    }

    Node after = before.next; // the node that follows

    before.next = new Node(o, before, after);

    after.previous = before.next;

    return true;
}

//Removes one item from the position pos.

public void remove(int pos) {

    if (pos < 0) {
        throw new IndexOutOfBoundsException(Integer.toString(pos));
    }

    Node before = head;
    for (int i=0; i<pos; i++) { // visiting pos nodes
        if (before.next == head) {
            throw new IndexOutOfBoundsException(Integer.toString(pos));
        }
        before = before.next;
    }

    Node after = before.next.next;

    before.next = after;

    after.previous = before;

}
```

```
    // Knowing that both lists store their elements in increasing
    // order, both lists can be traversed simultaneously.

    @SuppressWarnings("unchecked")

    public void merge(OrderedList other) {

        Node thisCurrent = head.next;
        Node otherCurrent = other.head.next;

        while (otherCurrent != other.head) {

            if (thisCurrent == head) {
                thisCurrent.next = new Node(otherCurrent.value, thisCurrent,
thisCurrent.next);
                thisCurrent = thisCurrent.next;
                thisCurrent.next.previous = thisCurrent;
                otherCurrent = otherCurrent.next;
            } else if (otherCurrent.value.compareTo(thisCurrent.value) < 0) {
                //insert before
                thisCurrent.previous = new Node(otherCurrent.value,
thisCurrent.previous, thisCurrent);
                thisCurrent.previous.previous.next = thisCurrent.previous;
                otherCurrent = otherCurrent.next;
            } else if (thisCurrent.next == head) {
                //insert after
                thisCurrent.next = new Node(otherCurrent.value, thisCurrent,head);
                thisCurrent = thisCurrent.next;
                thisCurrent.next.previous = thisCurrent;
                otherCurrent = otherCurrent.next;
            } else {
                thisCurrent=thisCurrent.next;
            }

        }
    }
}
```