```java
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;
    private Node tail;

    // Representation of the empty list.

    public OrderedList() {
        head = null;
        tail = null;
        // Your code here.
        //throw new UnsupportedOperationException( "not implemented yet!" );
    }

    // Calculates the size of the list

    public int size() {
          Node p = head;
          int compteur = 0;
          while(p!=null){
              p = p.next;
              compteur++;
          }
          return compteur;
    }


    public Object get( int pos ) {
        if (pos < 0) {
            throw new IndexOutOfBoundsException( Integer.toString( pos ) );
        }

        Node p = head;

        for ( int i=0; i<pos; i++ ) {
            if ( p == null ) {
                throw new IndexOutOfBoundsException( Integer.toString( pos ) );
            } else {
                p = p.next;
            }
        }

        return p.value;
    }
    // Adding an element while preserving the order

    public boolean add( Comparable o ) {
        if ( o == null ) {
            throw new IllegalArgumentException( "null" );
        }

        if ( head == null ) { // special case: empty list

            head = tail = new Node( o, null, null );

        } else if ( head.value.compareTo(o) >= 0 ) { // special case: add before
first node

            head = new Node( o, null, head );

            head.next.previous = head;

        } else {

            Node p = head;


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

            if ( p.next == null ) { // adding at the end of the list

                tail.next = new Node( o, tail, null );

                tail = tail.next;

            } else { // intermediate position

                Node q = p.next; // the node that follows

                p.next = new Node( o, p, q );

                q.previous = p.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 p = head;

        if ( pos == 0 ) {

            if ( head == null ) {
                throw new IndexOutOfBoundsException( Integer.toString( pos ) );
            }

            head = head.next;
            if ( head == null ) {
                tail = null; // this was the last element
            } else {
                head.previous = null;
            }

            p.value = null;
            p.next = null;

        } else {

            for ( int i=0; i<pos; i++ ) // traversing pos nodes
                if ( p == null ) {
                    throw new IndexOutOfBoundsException(Integer.toString(pos));
                } else {
                    p = p.next;
                }

            Node del = p;  // the node to delete

            p = p.previous; // p designates de previous node

            Node q = del.next; // q designates the node that follows

            p.next = q;

            if ( del == tail ) {
                tail = p;
            } else {
                q.previous = p;
            }

            del.value = null;
            del.next = null;
            del.previous = null;

        }

    }

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

    public void merge( OrderedList other ) {
        Node p = head;
        Node q = other.head;

        while ( q != null ) {
            if ( p == null ) {  // special case this list was empty
                head = tail = new Node( q.value, null, null );
                p = head;
                q = q.next;
            } else if ( q.value.compareTo( p.value ) < 0) {
                // insert before
                if ( p == head ) {
                    head = new Node( q.value, null, head );
                    p.previous = head;
                } else {
                    p.previous.next = new Node( q.value, p.previous, p );
                    p.previous = p.previous.next;
                }
                q = q.next;
            } else if ( p.next == null ) {
                // insert after
                p.next = new Node( q.value, p, null );
                tail = p.next;
                p = p.next;
                q = q.next;
            } else {
                p = p.next;
            }
        }
    }
}
```