```java
// User of ADT
public static<T> void CLS(List<T> l, int n) {
    for(int i = 0; i < n; i++) {
        l.findFirst();
        T x = l.retrieve();
        l.remove();

        if(!l.empty()) {
            while(!l.last())
                l.findNext();
        }

        l.insert(x);
    }
}

// Implementer of ADT, Linked-List Implementation (extra)
public void CLS(int n) {
    for(int i = 0; i < n; i++) {
        Node<T> temp = head;
        head = head.next;

        while(current.next != null)
            current = current.next;

        current.next = temp;
        temp.next = null;
    }
}

// Implementer of ADT, Array Implemention (extra)
public void CLS(int n) {
    for(int i = 0; i < n; i++) {
        T x = nodes[0];

        for(var j = 1; j < size; j++)
            nodes[j - 1] = nodes[j];

        nodes[size - 1] = x;
    }
}

//-------------------------------------------------------//

// User of ADT
public static<T> T MFE(List<T> l) {
    T mfe = null;
    int max = 0;

    int n = 0;
    if(!l.empty()) {
        l.findFirst();
        while(!l.last()) {
            n++;
            l.findNext();
        }
        n++;
    }
```

```
    for(int i = 0; i < n; i++) {
        l.findFirst();
        for(int j = 0; j < i; j++)
            l.findNext();
        T x = l.retrieve();

        int c = 0;
        while(!l.last()) {
            if(l.retrieve().equals(x))
                c++;
            l.findNext();
        }
        if(l.retrieve().equals(x))
            c++;

        if(c > max) {
            mfe = x;
            max = c;
        }
    }

    return mfe;
}

// Implementer of ADT, Linked-List Implementation (extra)
public T MFE() {
    T mfe = null;
    int max = 0;

    Node<T> temp1 = head;
    while(temp1 != null) {
        Node<T> temp2 = temp1;
        T x = temp1.data;

        int c = 0;
        while(temp2 != null) {
            if(temp2.data.equals(x))
                c++;
            temp2 = temp2.next;
        }

        if(c > max) {
            mfe = x;
            max = c;
        }

        temp1 = temp1.next;
    }

    return mfe;
}

// Implementer of ADT, Array Implemention (extra)
public T MFE() {
    T mfe = null;
    int max = 0;

    for(int i = 0; i < size; i++) {
        T x = nodes[i];
```

```
        int c = 0;
        for(int j = i; j < size; j++) {
            if(nodes[j].equals(x))
                c++;
        }

        if(c > max) {
            mfe = x;
            max = c;
        }
    }

    return mfe
}

//-----------------------------------------------------//

// User of ADT
public static<T> void switch(List<T> l1, List<T> l2) {
    l1.findFirst();
    l2.findFirst();

    if(!l1.last()) {
        l1.findNext();
        while(!l1.last()) {
            T x = l1.retrieve();
            l1.remove();
            l2.insert(x);
        }
        T x = l1.retrieve();
        l1.remove();
        l2.insert(x);
    }

    if(!l2.last()) {
        l2.findNext();
        while(!l2.last()) {
            T x = l2.retrieve();
            l2.remove();
            l1.insert(x);
        }
        T x = l2.retrieve();
        l2.remove();
        l1.insert(x);
    }
}

// Implementer of ADT, Linked-List Implementation (extra)
public void switch(LinkedList<T> l2) {
    Node<T> temp1 = head.next;
    Node<T> temp2 = l2.head.next;
    head.next = temp2;
    l2.head.next = temp1;
}

// Implementer of ADT, Array Implemention (extra)
public void switch(ArrayList<T> l2) {
    for(int i = 1; i < size || i < l2.size; i++) {
```

```
        T x = nodes[i];
        T y = l2.nodes[i];
        nodes[i] = y;
        l2.nodes[i] = x;
    }

    int temp = size;
    size = l2.size;
    l2.size = temp;
}
```