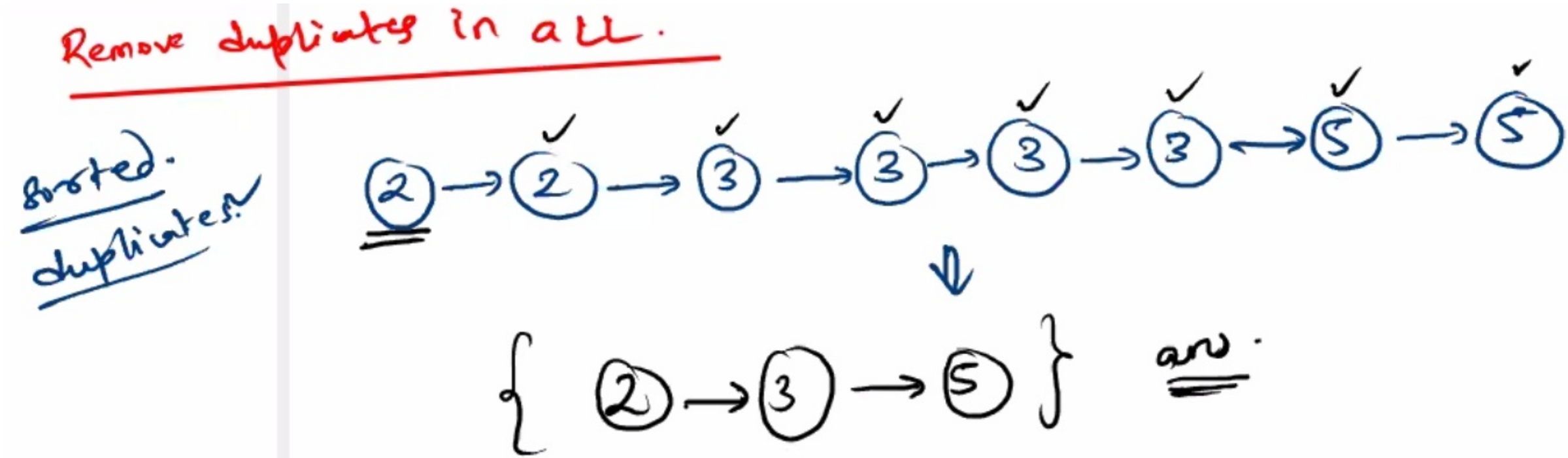
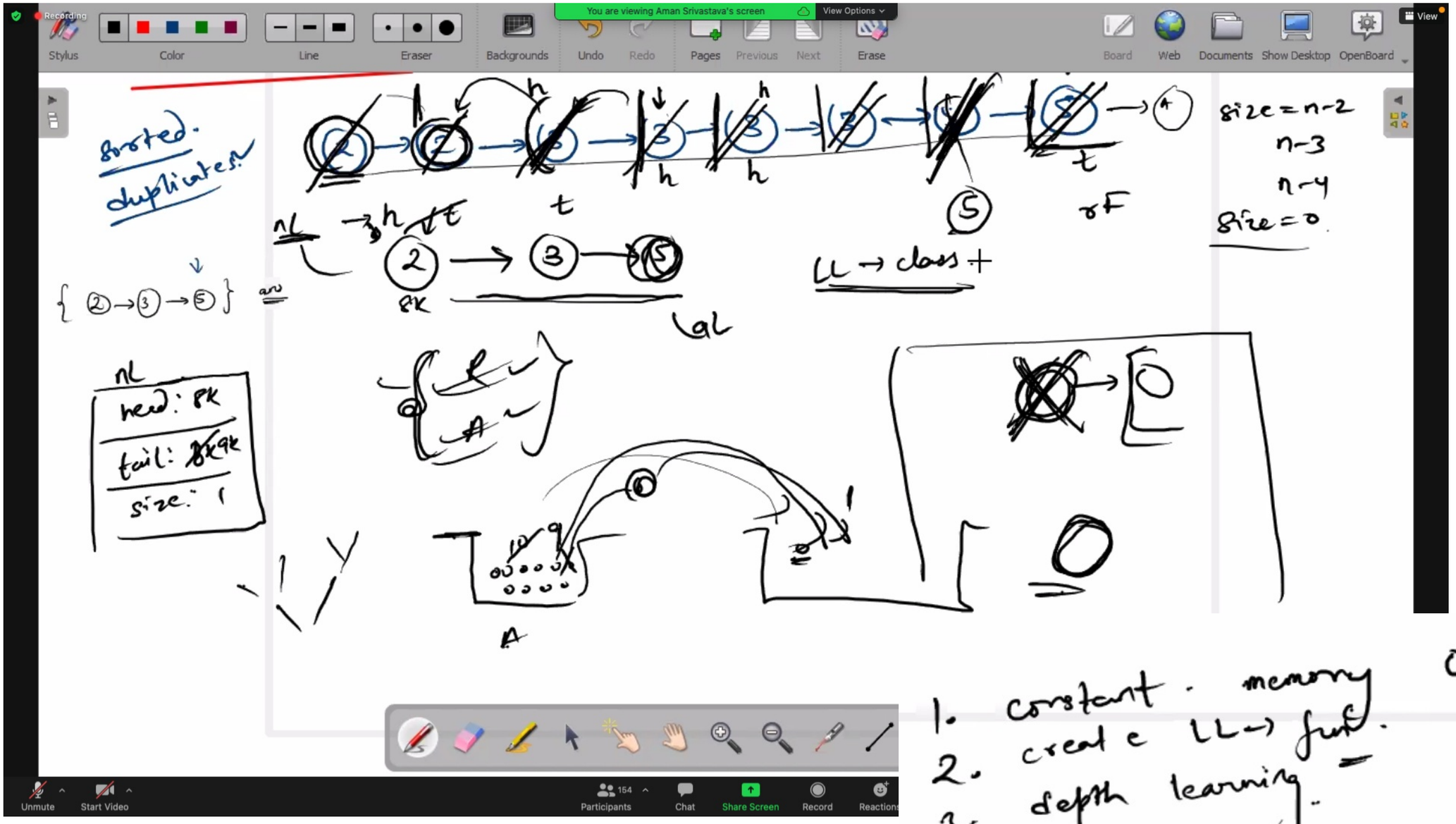
Lecture-038

Problem :

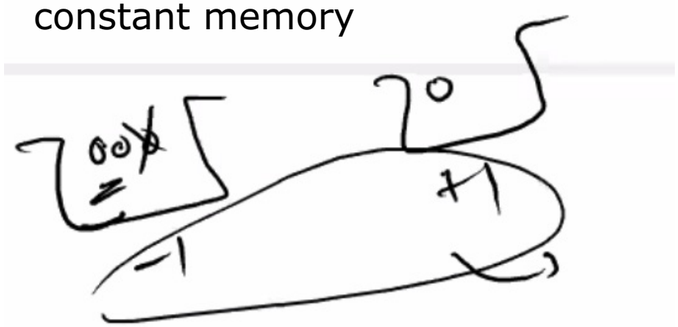


Solution / Approach

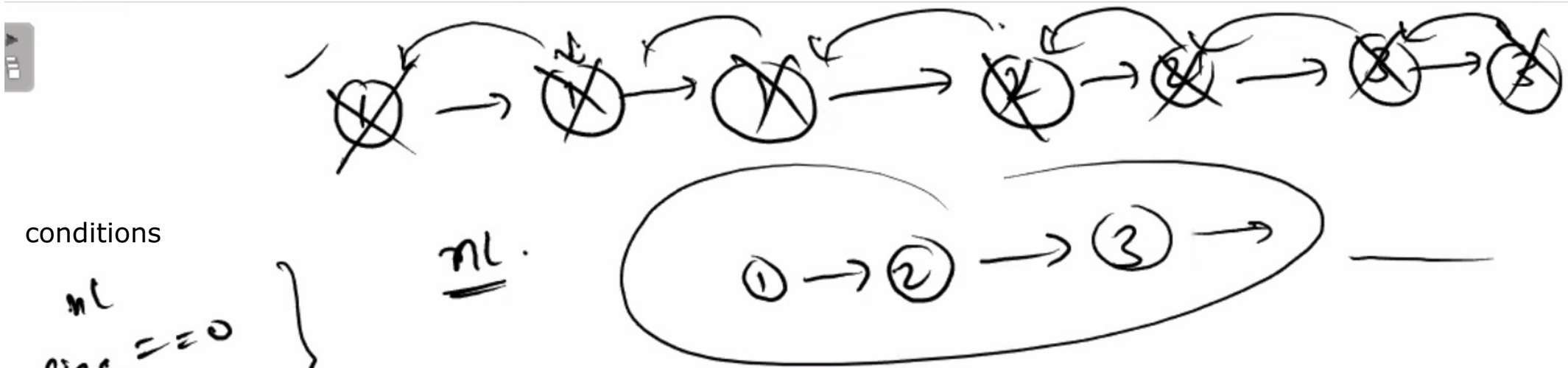


1. constant. memory O(1).
2. create LL → func.
3. depth learning.

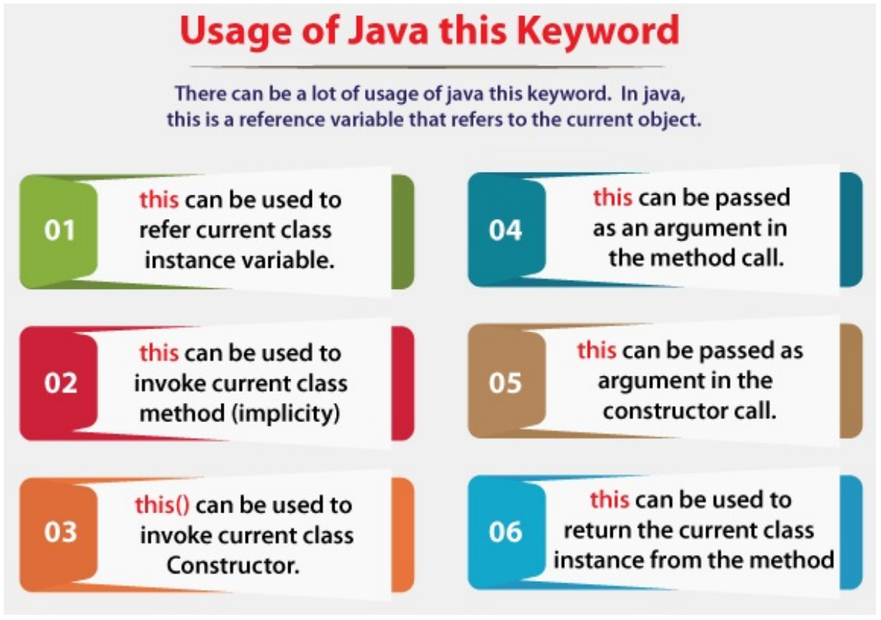
constant memory

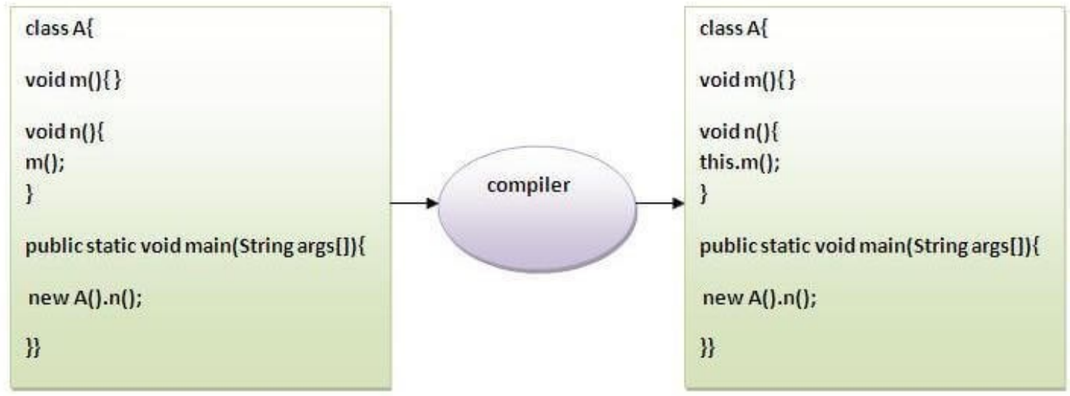


Implementation :



There can be a lot of usage of Java this keyword. In Java, this is a reference variable that refers to the current object.

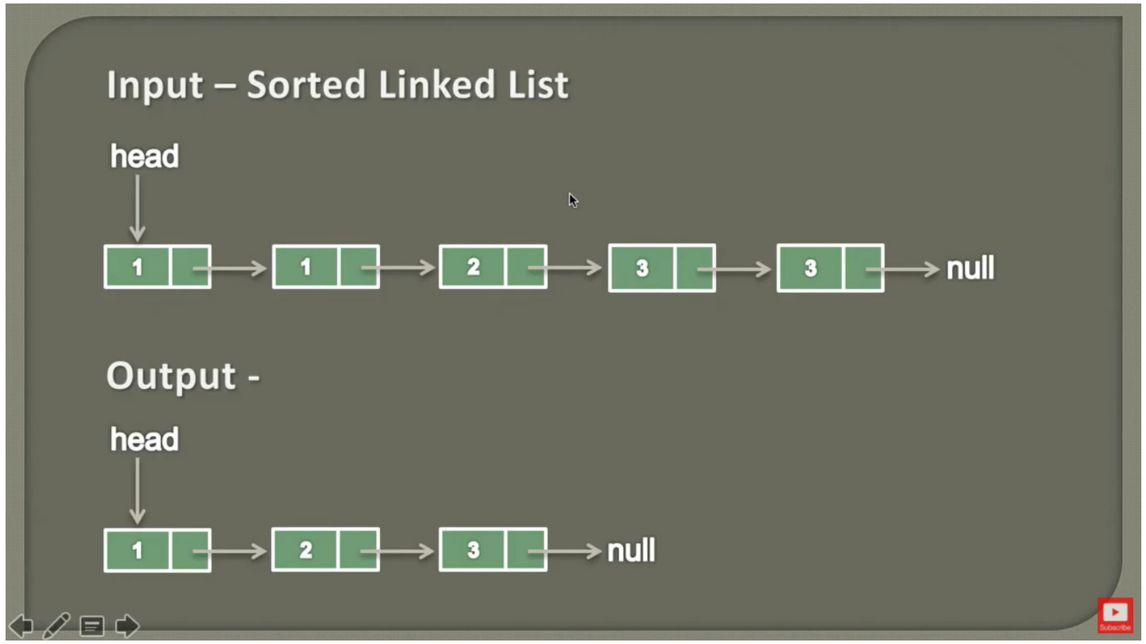




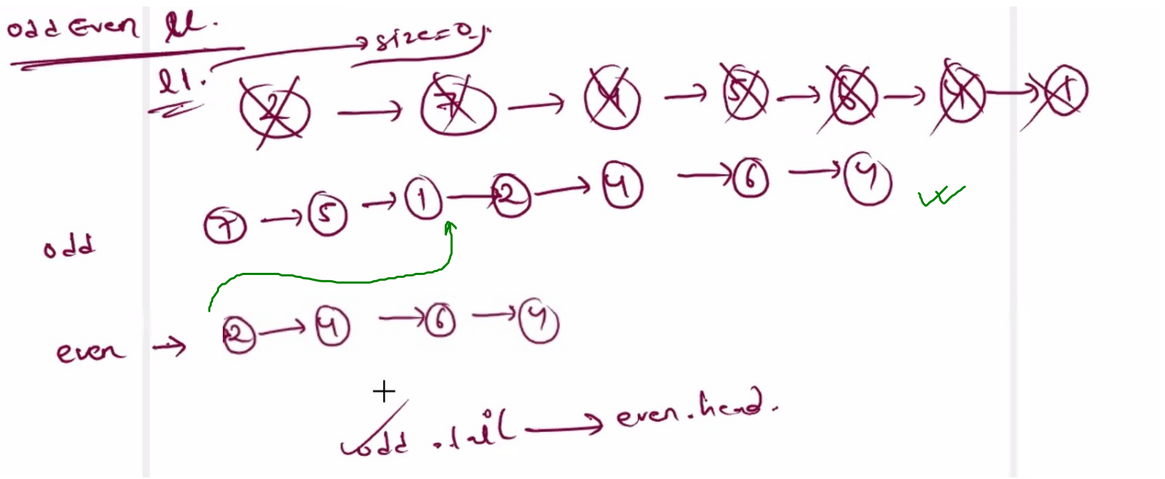
take refrence from javaTpoint

Remove Duplicates In A Sorted Linked List

```
public void removeDuplicates(){
  LinkedList nl = new LinkedList();
  while(this.size > 0){
      int val = this.getFirst();
      this.removeFirst();

      if(nl.size == 0 || nl.tail.data != val){
        nl.addLast(val);
      }
  }
  this.head = nl.head;
  this.tail = nl.tail;
  this.size = nl.size;
}
```



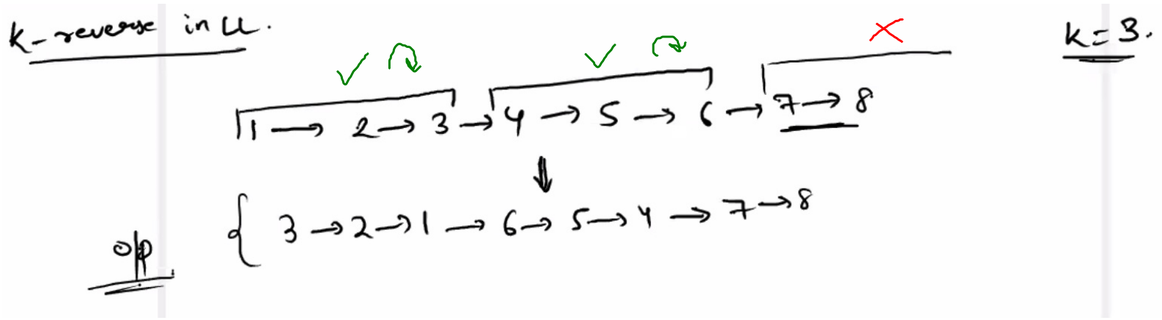
Problem :Odd Even Linked List



Solution :

```java
public void oddEven() {
    LinkedList odd = new LinkedList(); // list of odd
    LinkedList even = new LinkedList(); // list of even
    while (this.size > 0) { // while list is not empty
        int val = this.getFirst(); // get first element
        this.removeFirst(); // remove first element
        if (val % 2 == 0) { // if even
            even.addLast(val); // add to even list
        } else { // if odd
            odd.addLast(val); // add to odd list
        }
    }
    /*
     * corner case : if odd/even list is empty or present
     * odd even (combinations)
     * 1 1
     * 1 0
     * 0 1
     */
    if (odd.size > 0 && even.size > 0) { // if both lists are present
        odd.tail.next = even.head; // connect odd list to even list
        this.head = odd.head; // list starts with odd list
        this.tail = even.tail; // list ends with even list
        this.size = odd.size + even.size; // size of new list
    } else if (odd.size > 0) { // if only odd list is present
        this.head = odd.head; // list starts with odd list
        this.tail = odd.tail; // list ends with odd list
        this.size = odd.size; // size of new list is the size of odd list
    } else if (even.size > 0) { // if only even list is present
        this.head = even.head; // list starts with even list
        this.tail = even.tail; // list ends with even list
        this.size = even.size; // size of new list is the size of even list
    }
}
```

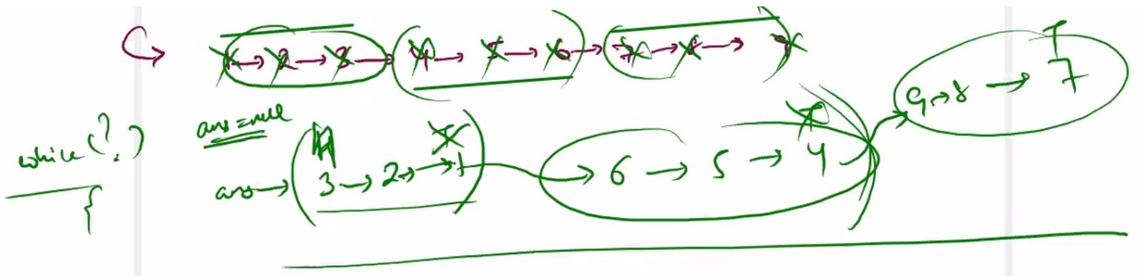
Problem : K Reverse In Linked List



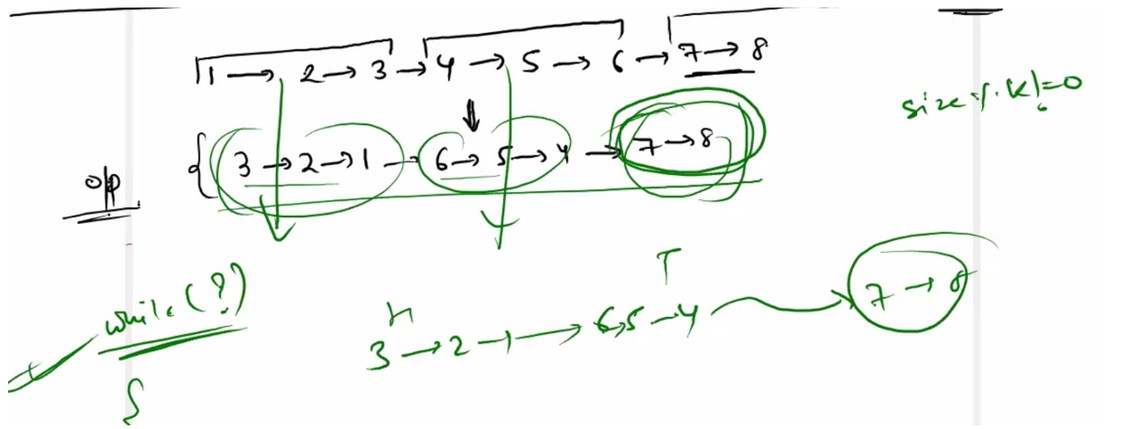
Just:
Thinking out Loud:
make reversals from 0 to size - (size%.k)
keep revering in sets of 'k'.

Approach : removeFirst -> addFirst => combine LL

case when size%k ==0



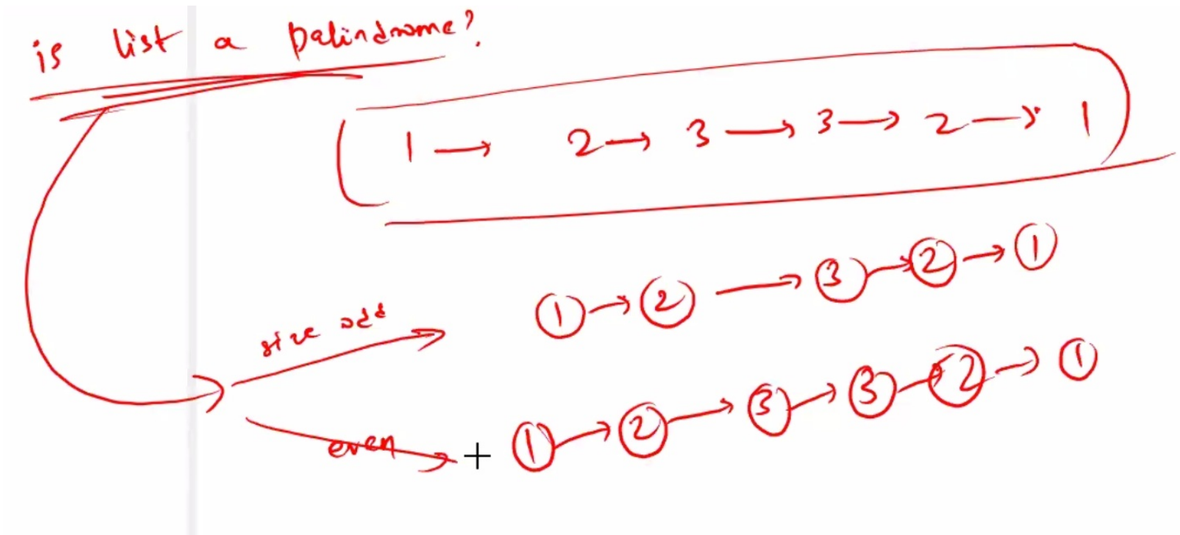
case when size%k != 0



Solution：

```java
public void kReverse(int k) {
    if (k > this.size) {
        return;
    }
    LinkedList ans = new LinkedList();
    while (this.size ≥ k) {
        LinkedList tmp = new LinkedList(); // temp list
        for (int i = 0; i < k; i++) {
            int val = this.getFirst(); // get first element
            this.removeFirst(); // remove first element
            tmp.addFirst(val); // add to temp list (addFirst ensures that the list is reversed)
        }

        if (ans.size == 0) {
            ans = tmp; // shallow copy of temp list
        } else {
            ans.tail.next = tmp.head; // connect ans list to temp list
            ans.tail = tmp.tail; // update tail of ans list
            ans.size += tmp.size; // update size of ans list
        }

    }

    // left elements
    if (this.size > 0) {
        ans.tail.next = this.head; // connect ans list to left elements
        ans.tail = this.tail; // update tail of ans list
        ans.size += this.size; // update size of ans list
    }

    this.head = ans.head; // deep copy
    this.tail = ans.tail;
    this.size = ans.size;
}
```

H/W : Is Linked List A Palindrome?



Approach (as discussed)