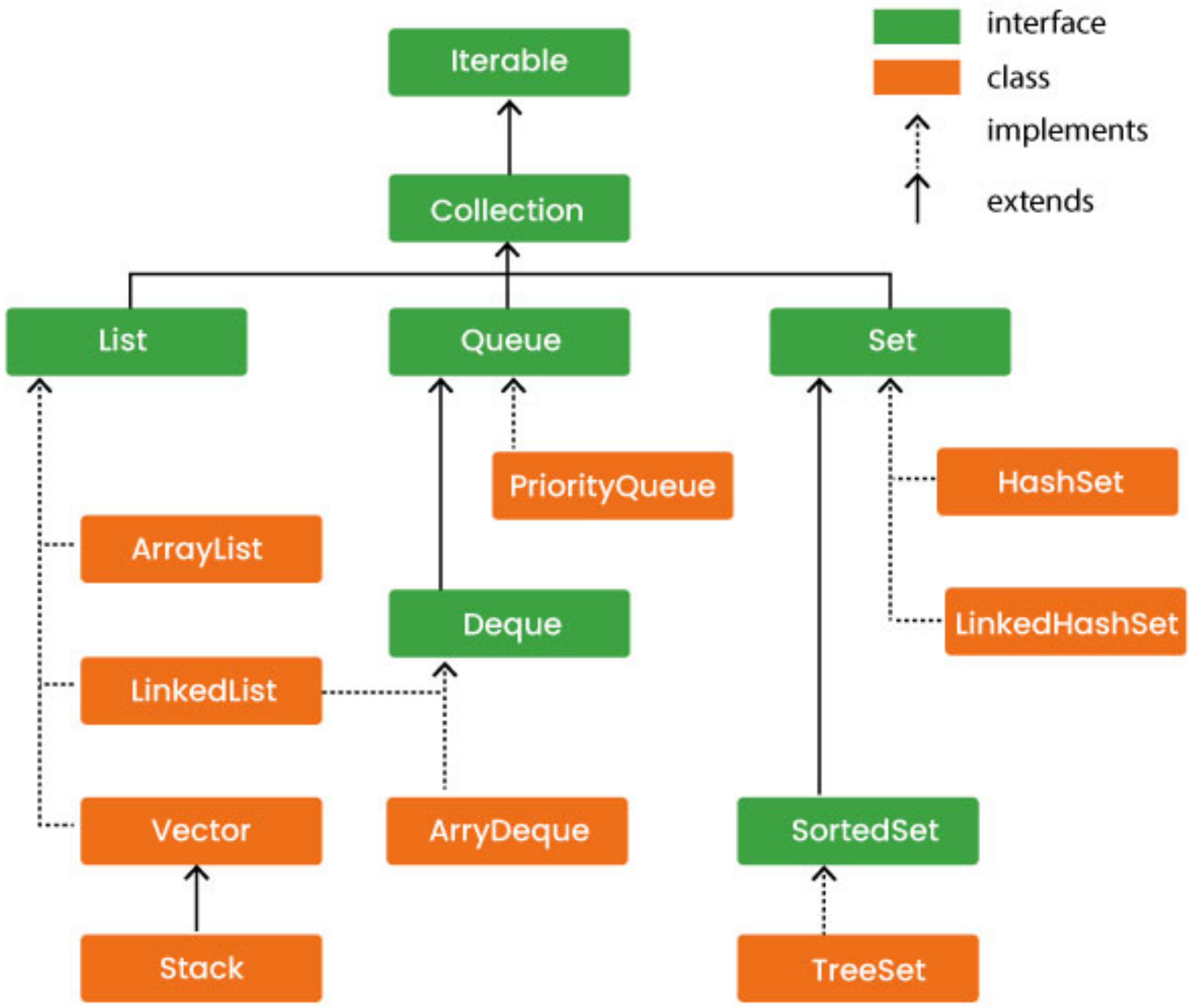
Collections

Java Collection Framework is a **set of classes and interfaces** that provide ready-made data structures to store and manipulate data efficiently.

Hierarchy of Collection Framework



Real-World Applications

List
A playlist of songs.

Set
A collection of student roll numbers.

Queue
A line at a movie ticket counter

Map
A dictionary (word → meaning)

- interface1 -> extends -> interface2
- class1 -> extends -> class2
- class1-> implements -> interface1

- class MyClass implements myInterface extends OtherClass

List Interface

- Part of java.util package
- Maintain insertion order
- Allows duplicate elements
- Index based access
- Support positional operations

Implementation of List Interface

- ArrayList
- LinkedList
- Vector
- Stack

ArrayList

- Implementation of List interface
- Uses **dynamic array** with default inital capacity is 10
- Grows automatically when capacity is full
- Not synchronized (not thread safe)
- Allows null values
- Used when used ?
    - Read operation are more than write
    - When index-based access is required

Imp Methods

1. **add(E e)** → Adds an element to the end of the list
2. **add(int index, E e)** → Inserts an element at a specific position
3. **get(int index)** → Returns the element at the given index
4. **set(int index, E e)** → Replaces the element at the given index
5. **remove(int index)** → Removes the element at the given index
6. **remove(Object o)** → Removes the first matching element
7. **size()** → Returns the number of elements in the list
8. **isEmpty()** → Checks whether the list is empty
9. **contains(Object o)** → Checks if the element exists in the list
10. **clear()** → Removes all elements from the list
11. **indexOf(Object o)** → Returns index of first occurrence
12. **lastIndexOf(Object o)** → Returns index of last occurrence
13. **toArray()** → Converts the list into an array

Iterate ArrayList

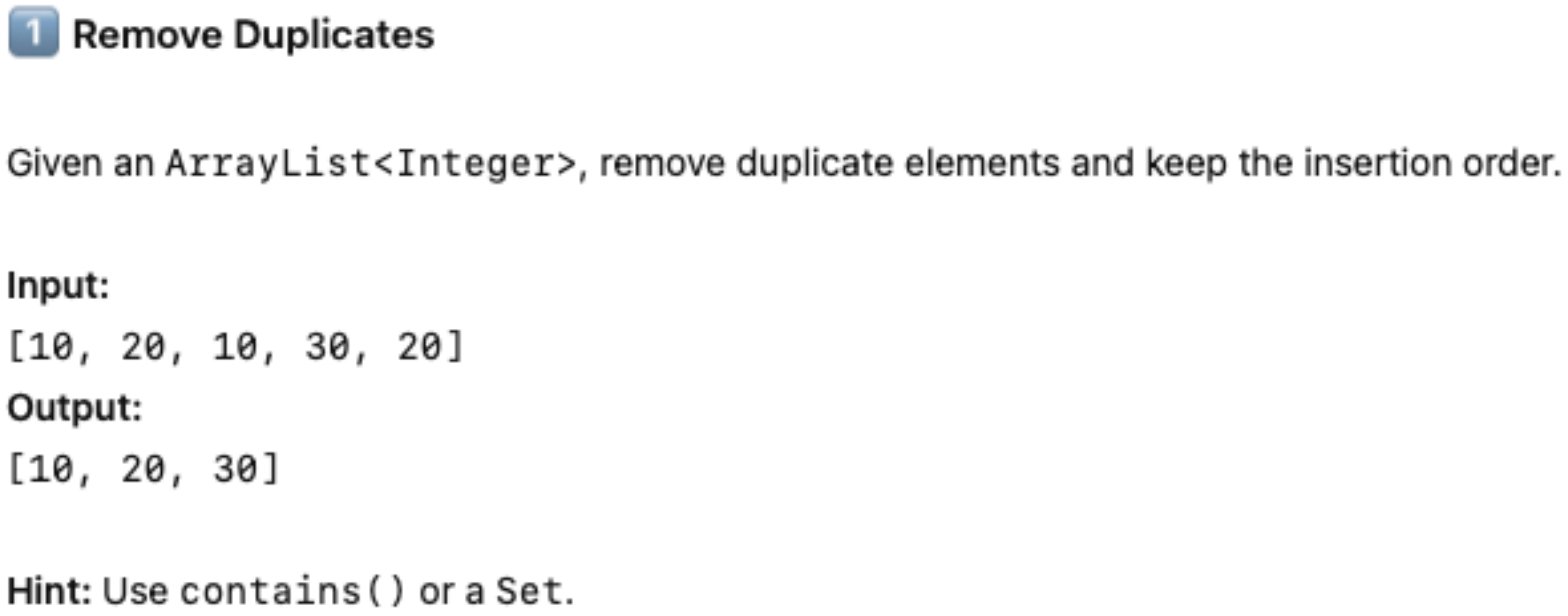
1. **For loop**: used when index access is needed
2. **Enhanced for loop (for-each loop)**: best for read only traversal
3. **While loop**: useful when loop condition is not index-based.
4. **Iterator Interface**: allows safe removal of elements during iteration
5. **forEach() method**: functional programming, uses when concise iteration

Array Sorting

- Collections.sort()
- ArrayList -> sort(null), sort(Collections.reverseOrder())

Questions

Question 1



1️⃣ **Remove Duplicates**

Given an `ArrayList<Integer>`, remove duplicate elements and keep the insertion order.

Input:

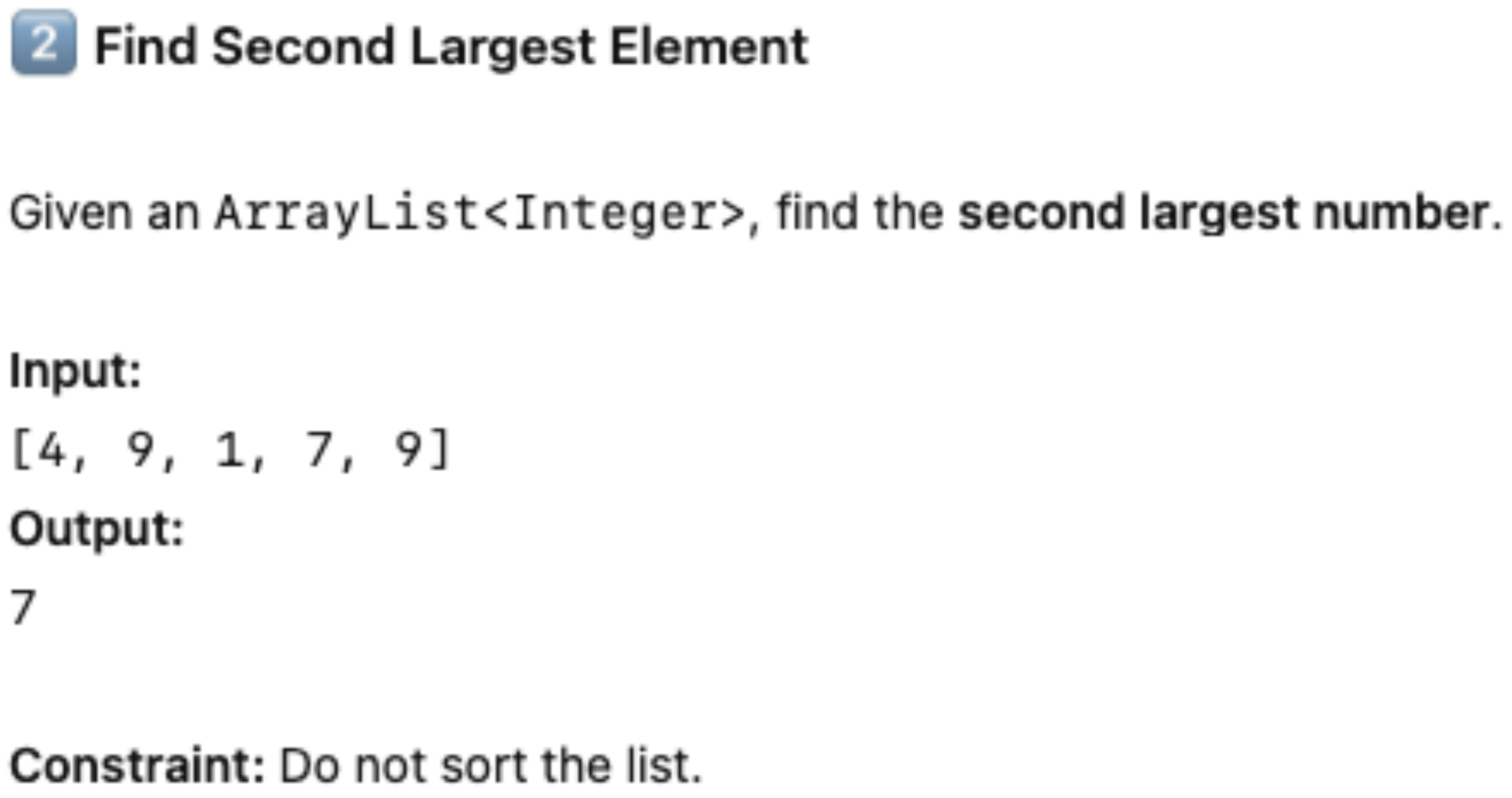
```
[10, 20, 10, 30, 20]
```

Output:

```
[10, 20, 30]
```

Hint: Use `contains()` or a `Set`.

Question 2



2️⃣ Find Second Largest Element

Given an `ArrayList<Integer>`, find the **second largest number**.

Input:

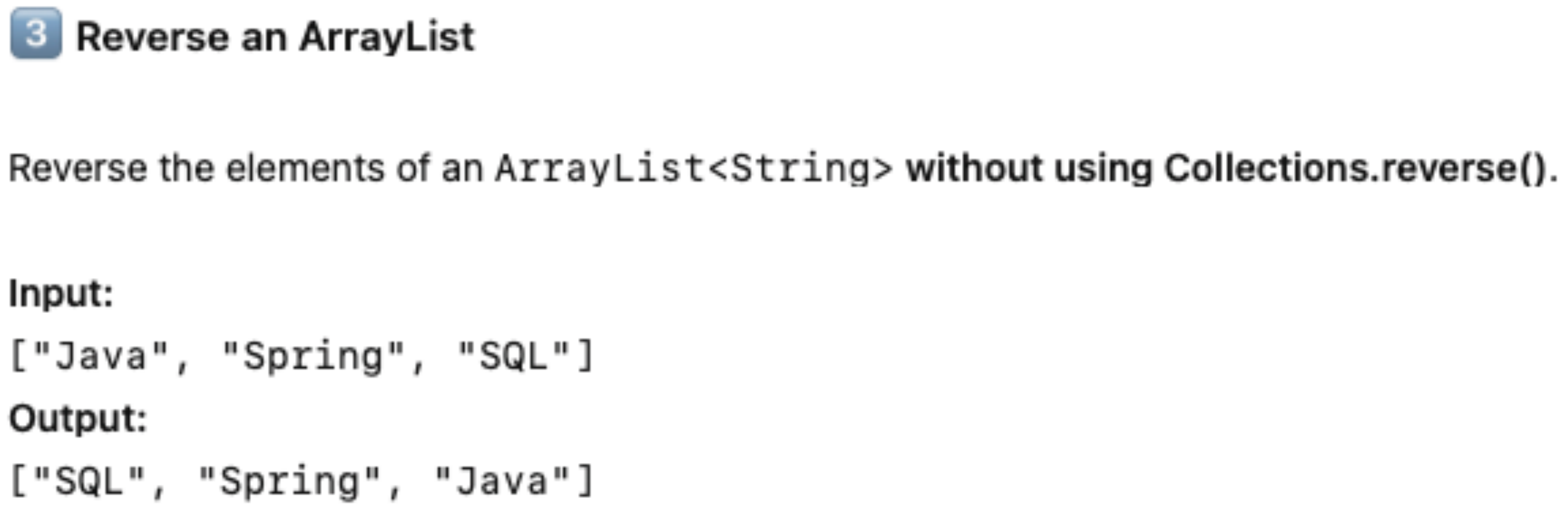
```
[4, 9, 1, 7, 9]
```

Output:

```
7
```

Constraint: Do not sort the list.

Question 3



3️⃣ Reverse an ArrayList

Reverse the elements of an `ArrayList<String>` **without using Collections.reverse()**.

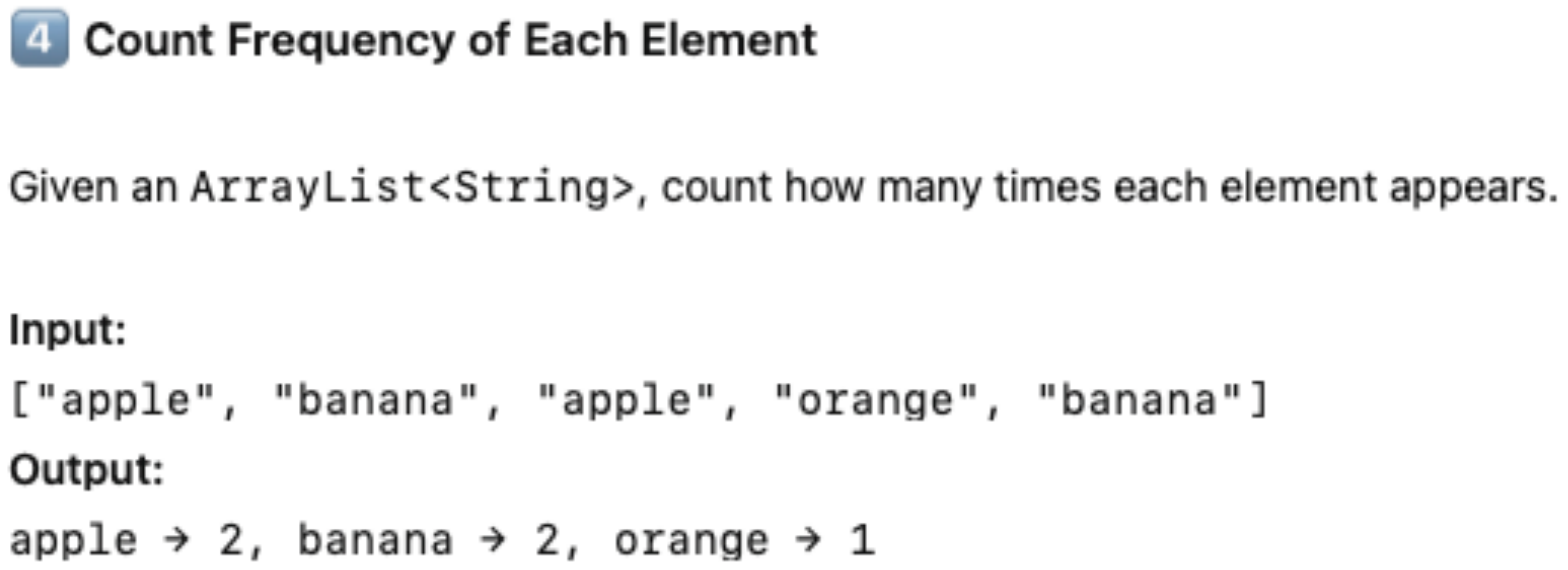
Input:

```
["Java", "Spring", "SQL"]
```

Output:

```
["SQL", "Spring", "Java"]
```

Question 4



4️⃣ Count Frequency of Each Element

Given an `ArrayList<String>`, count how many times each element appears.

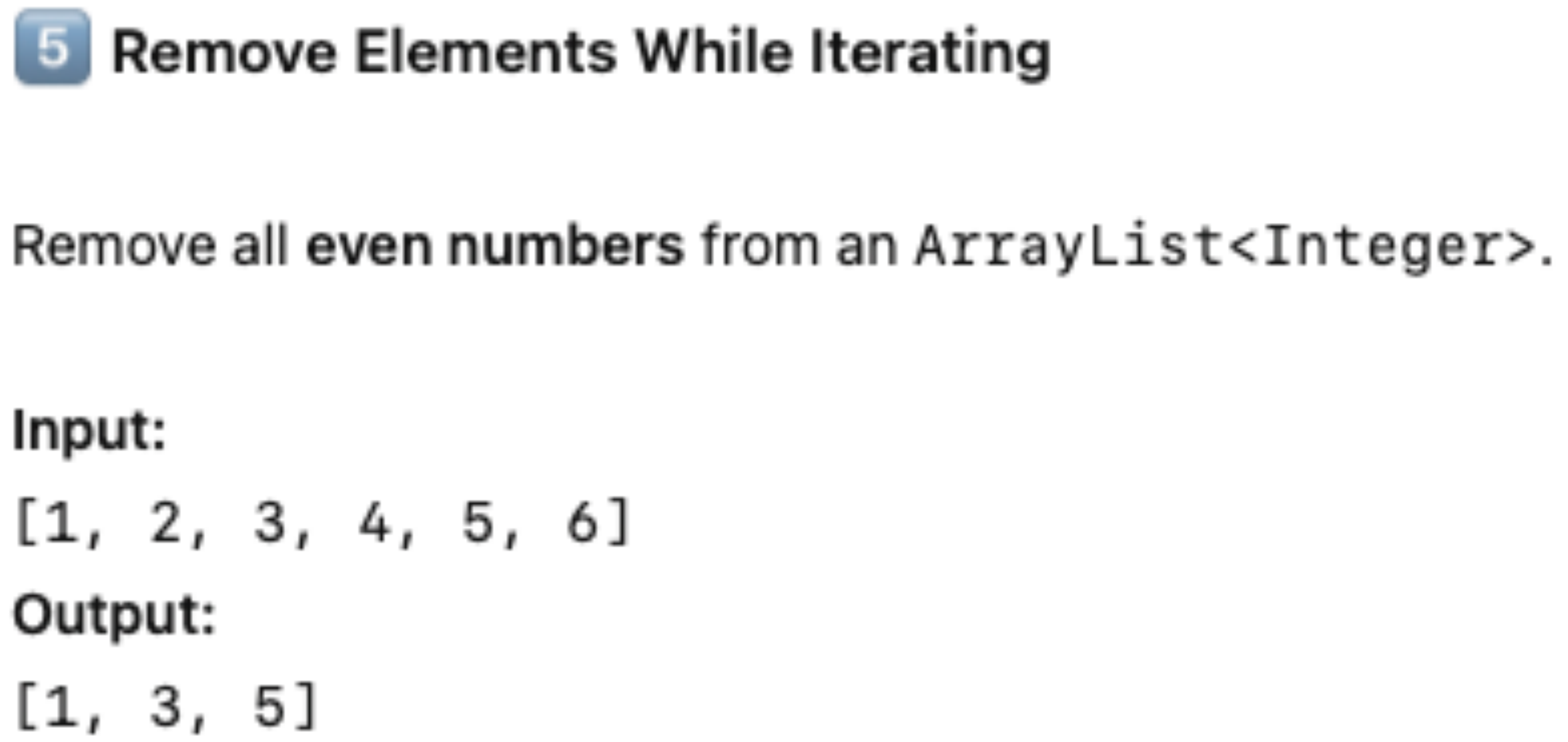
Input:

```
["apple", "banana", "apple", "orange", "banana"]
```

Output:

```
apple → 2, banana → 2, orange → 1
```

Question 5



5️⃣ Remove Elements While Iterating

Remove all **even numbers** from an `ArrayList<Integer>`.

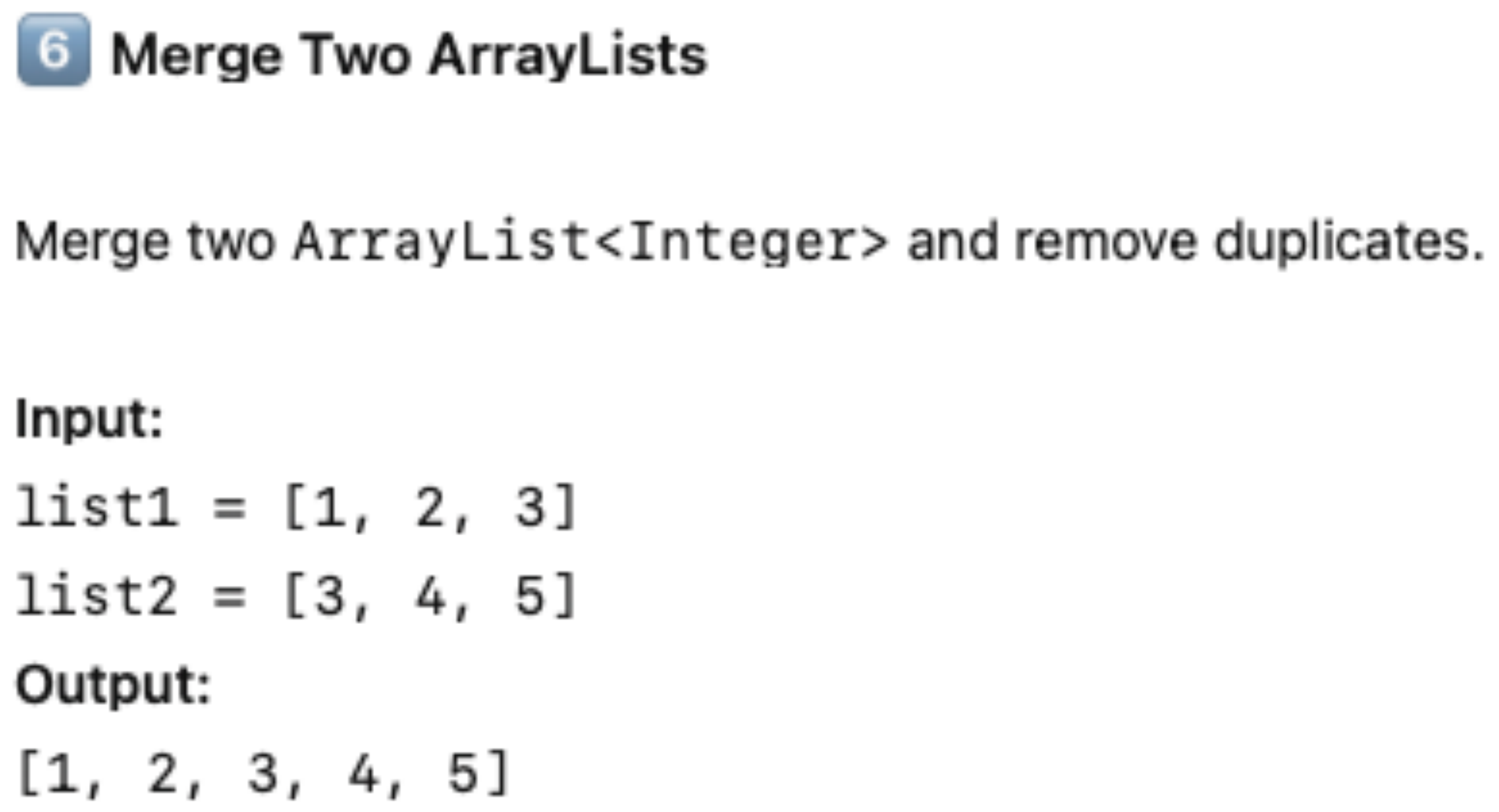
Input:

```
[1, 2, 3, 4, 5, 6]
```

Output:

```
[1, 3, 5]
```

Question 6



6️⃣ Merge Two ArrayLists

Merge two `ArrayList<Integer>` and remove duplicates.

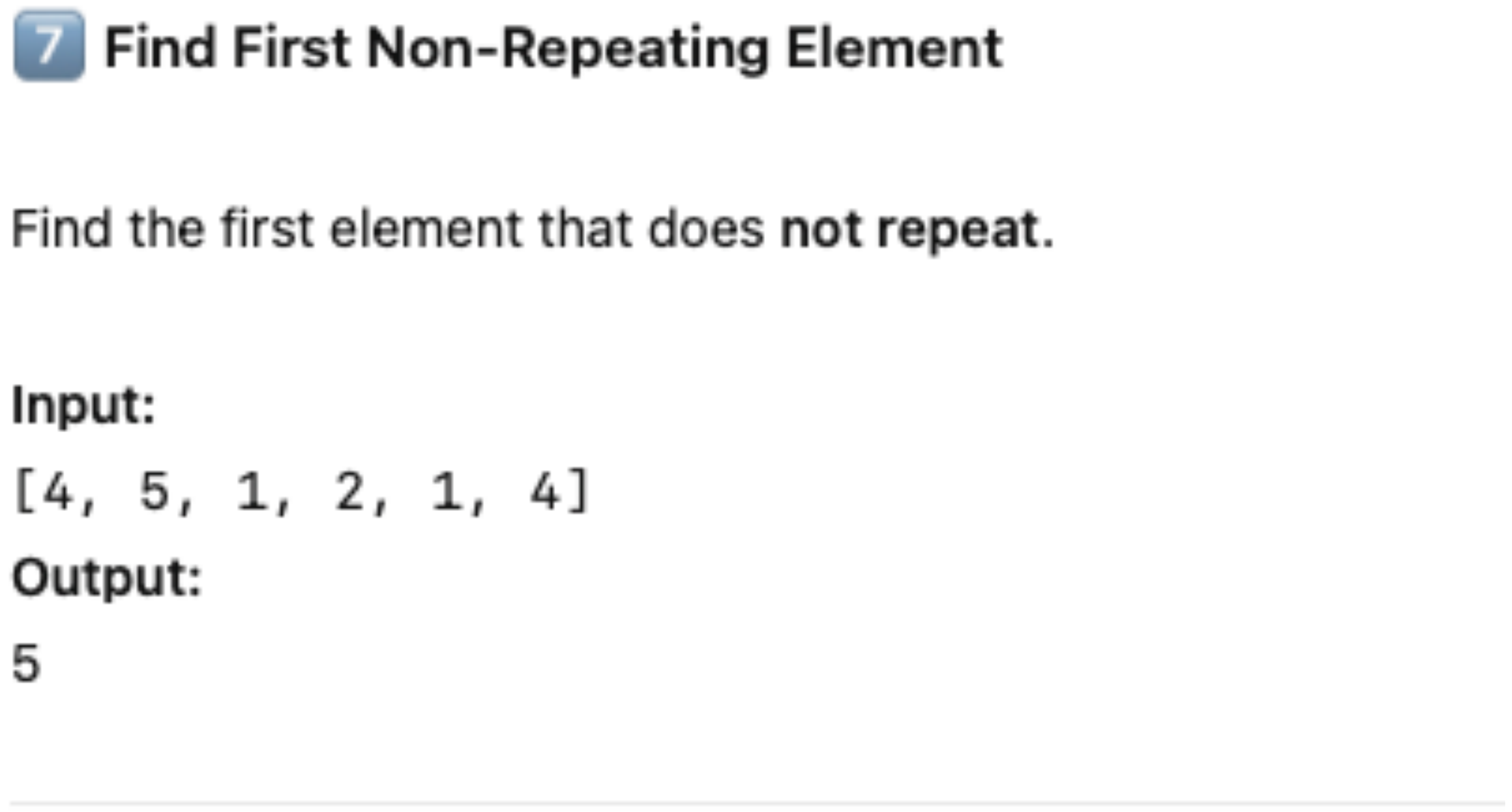
Input:

```
list1 = [1, 2, 3]
list2 = [3, 4, 5]
```

Output:

```
[1, 2, 3, 4, 5]
```

Question 7



7️⃣ Find First Non-Repeating Element

Find the first element that does **not repeat.**

Input:

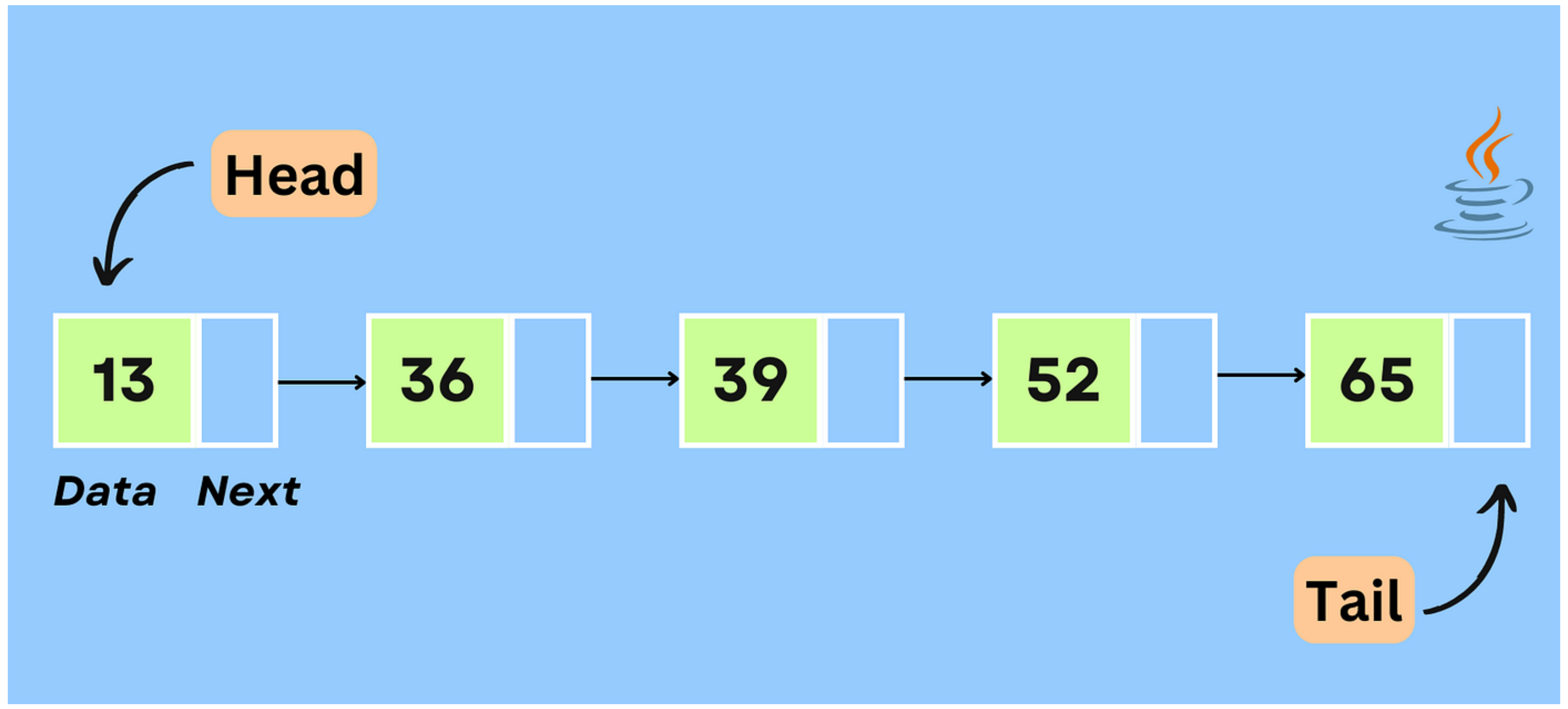
```
[4, 5, 1, 2, 1, 4]
```

Output:

5

LinkedList

- A linked list is a sequence of nodes where each node contains:
  - Data (the value to store).
  - A reference (pointer) to the next node in the sequence.
- The list is accessed starting from a special pointer called the head, and the last node usually points to null.



Imp Method

From List interface (common to ArrayList etc)

- **add(E e)** : Append element at end; returns true on success.
- **add(int index, E e)** : Insert element at specific position; shifts later elements.
- **addAll(Collection<? extends E> c)** : Append all elements from another collection.
- **addAll(int index, Collection<? extends E> c)** : Insert a collection starting at given index.
- **get(int index)** : Return element at given index (O(n) for linked list).
- **set(int index, E element)** : Replace element at index; returns old value.
- **remove(int index)** : Remove element at index; returns removed value.
- **remove(Object o)** : Remove first matching element; returns true if found.
- **clear()** : Remove all elements from list.
- **clear()** : Remove all elements from list.
- **size()** : Number of elements currently stored.
- **isEmpty()** : true if list has no elements.
- **contains(Object o)** : true if list has at least one matching element.
- **indexOf(Object o)** : Index of first occurrence or -1 if absent.
- **lastIndexOf(Object o)** : Index of last occurrence or -1 .
- **iterator()** : Forward iterator over elements.
- **listIterator() , listIterator(int index)** : Bidirectional iterator, optionally starting at index.
- **toArray() , toArray(T[] a)** : Copy elements into an array.

Specific Method of LinkedList

- **addFirst(E e)**: Insert at beginning; efficient O(1) for linked list.
- **addLast(E e)**: Insert at end explicitly.
- **getFirst()**: Return first element; throws if list empty.
- **getLast()**: Return last element; throws if list empty.
- **removeFirst()**: Remove and return first element; throws if empty.
- **removeLast()**: Remove and return last element; throws if empty.
- **offer(E e)**: Add at tail, returns true/false instead of throwing.
- **offerFirst(E e)**: Add at head, queue-friendly (returns boolean).
- **offerLast(E e)**: Add at tail, queue-friendly.
- **peek()**: Return head without removing; null if empty.
- **peekFirst()**: Return first or null if empty.
- **peekLast()**: Return last or null if empty.
- **poll()**: Remove and return head; null if empty.
- **pollFirst()**: Remove first or null if empty.
- **pollLast()**: Remove last or null if empty.
- **descendingIterator()**: Iterator that traverses from tail to head.
- **clone()**: Returns shallow copy of the list.

LinkedList Iteration

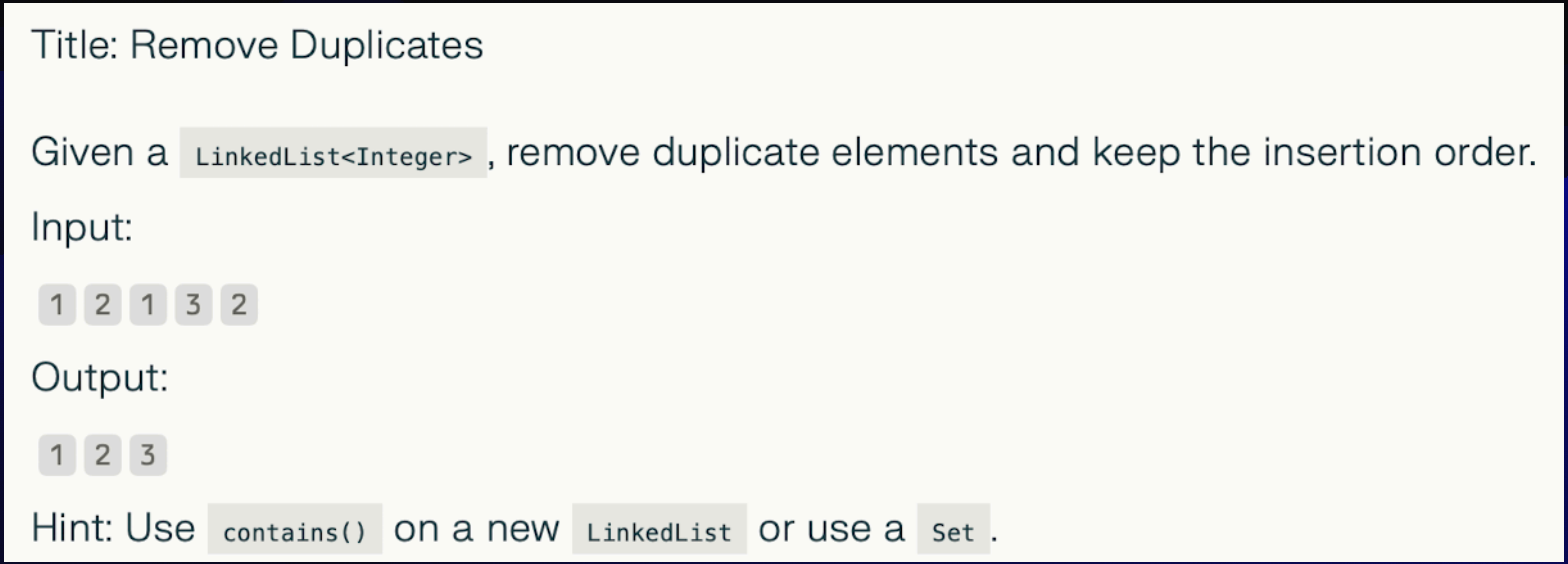
1. **For loop** simple index based iteration
2. **Inhanced for loop** (for each loop)
3. **Iterator** - forward iteration only
4. **ListIterator** - for bidirectional iteration (hasPrevious/previous), descendingIterator reverse order iteration
    1. ListIterator allows modification during iteration (safe concurrent changes)

Sorting Linked List

- Collections.sort(list)
- Collections.sort(list, Comparator)
- LinkedList -> sort(null), sort(Comparator)
-

Question

QUESTION 1



Title: Remove Duplicates

Given a `LinkedList<Integer>`, remove duplicate elements and keep the insertion order.

Input:

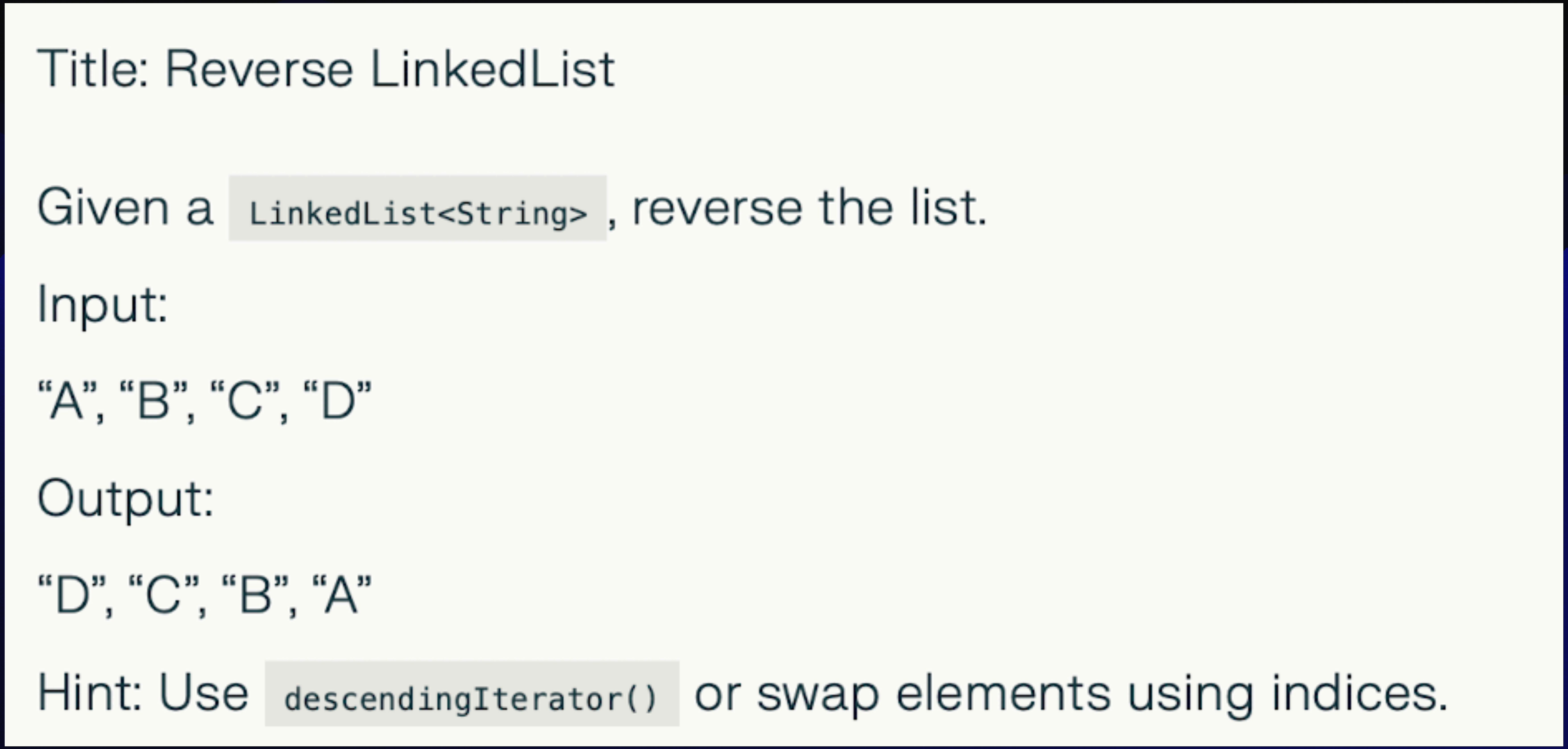
`1` `2` `1` `3` `2`

Output:

`1` `2` `3`

Hint: Use `contains()` on a new `LinkedList` or use a `Set`.

QUESTION 2



Title: Reverse LinkedList

Given a `LinkedList<String>` , reverse the list.

Input:

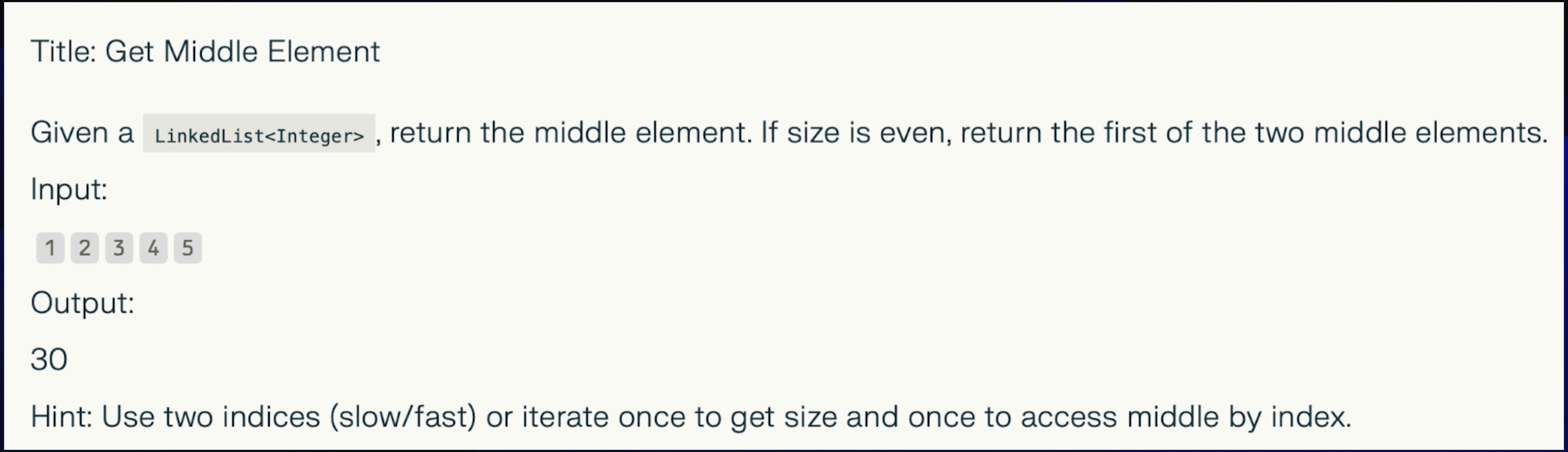
“A”, “B”, “C”, “D”

Output:

“D”, “C”, “B”, “A”

Hint: Use `descendingIterator()` or swap elements using indices.

QUESTION 3



Title: Get Middle Element

Given a `LinkedList<Integer>`, return the middle element. If size is even, return the first of the two middle elements.

Input:

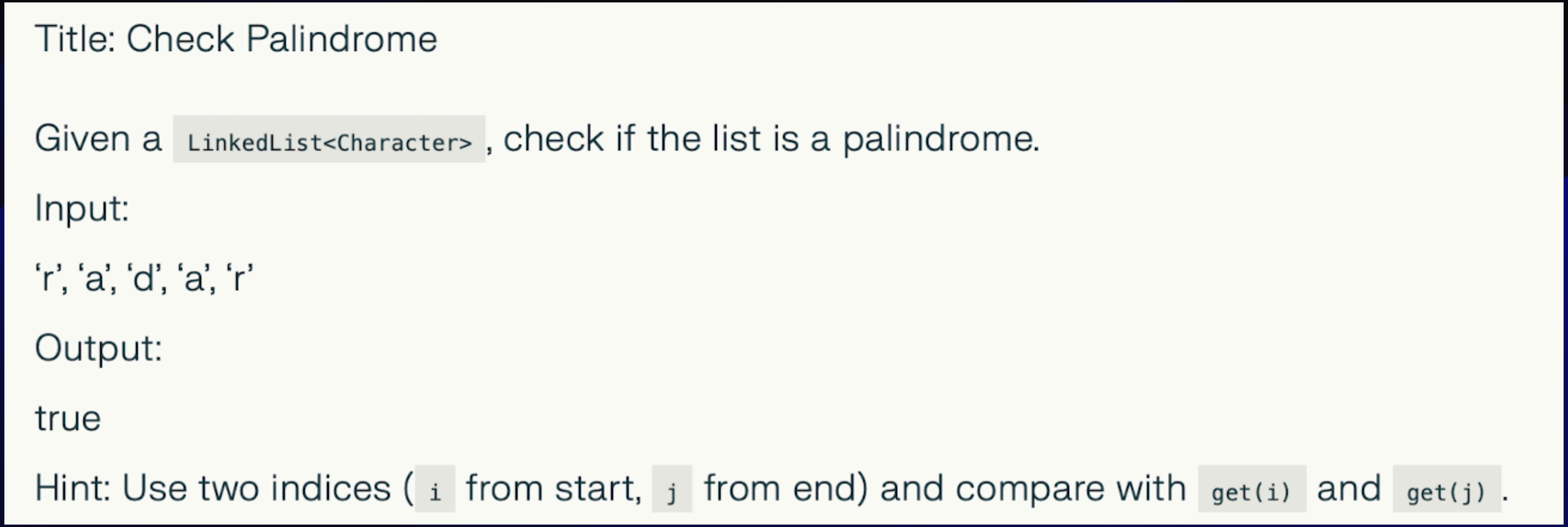
1 2 3 4 5

Output:

30

Hint: Use two indices (slow/fast) or iterate once to get size and once to access middle by index.

QUESTION 4



Title: Check Palindrome

Given a `LinkedList<Character>`, check if the list is a palindrome.

Input:

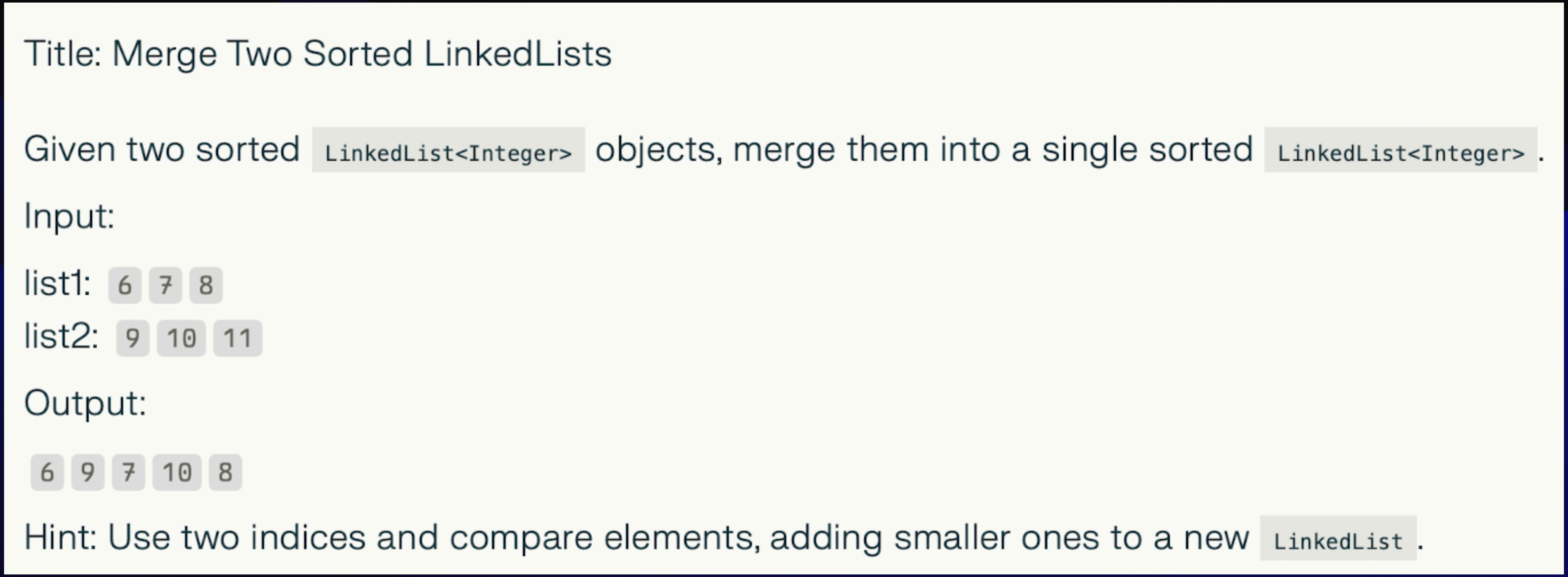
'r', 'a', 'd', 'a', 'r'

Output:

true

Hint: Use two indices (`i` from start, `j` from end) and compare with `get(i)` and `get(j)`.

QUESTION 5



Title: Merge Two Sorted LinkedLists

Given two sorted `LinkedList<Integer>` objects, merge them into a single sorted `LinkedList<Integer>`.

Input:

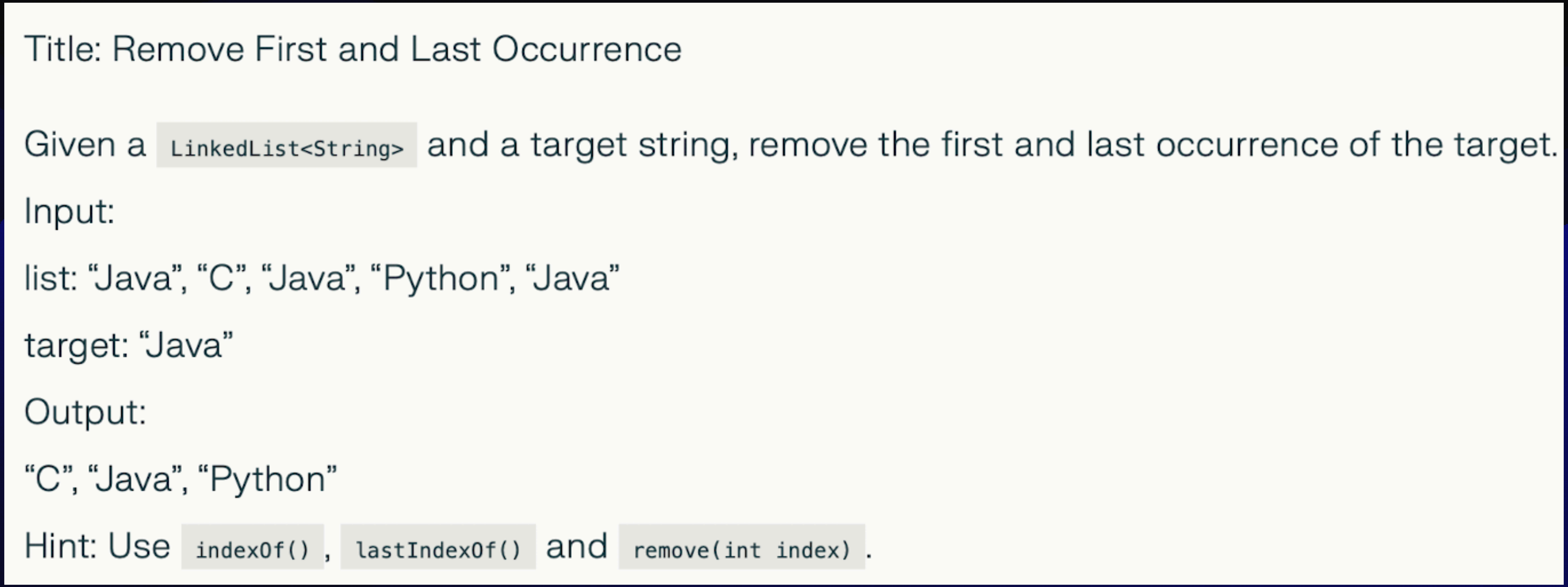
list1: `6` `7` `8`
list2: `9` `10` `11`

Output:

`6` `9` `7` `10` `8`

Hint: Use two indices and compare elements, adding smaller ones to a new `LinkedList`.

QUESTION 6



Title: Remove First and Last Occurrence

Given a `LinkedList<String>` and a target string, remove the first and last occurrence of the target.

Input:

list: “Java”, “C”, “Java”, “Python”, “Java”

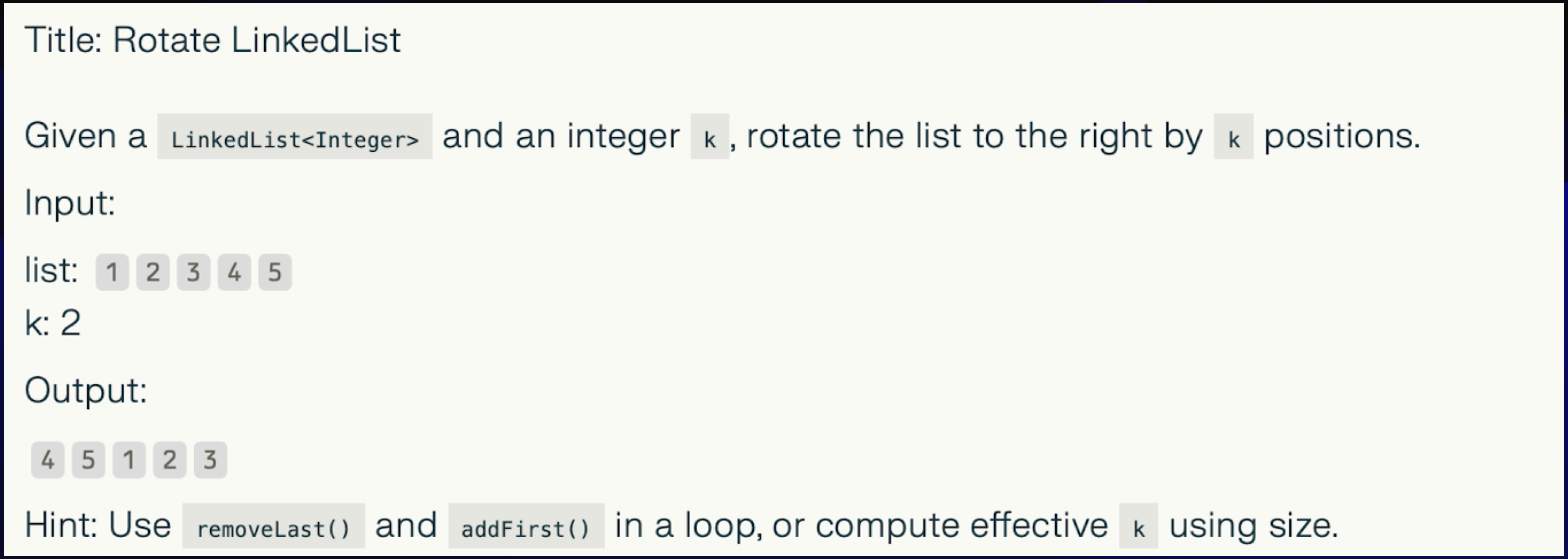
target: “Java”

Output:

“C”, “Java”, “Python”

Hint: Use `indexOf()`, `lastIndexOf()` and `remove(int index)`.

QUESTION 7



Title: Rotate LinkedList

Given a `LinkedList<Integer>` and an integer `k`, rotate the list to the right by `k` positions.

Input:

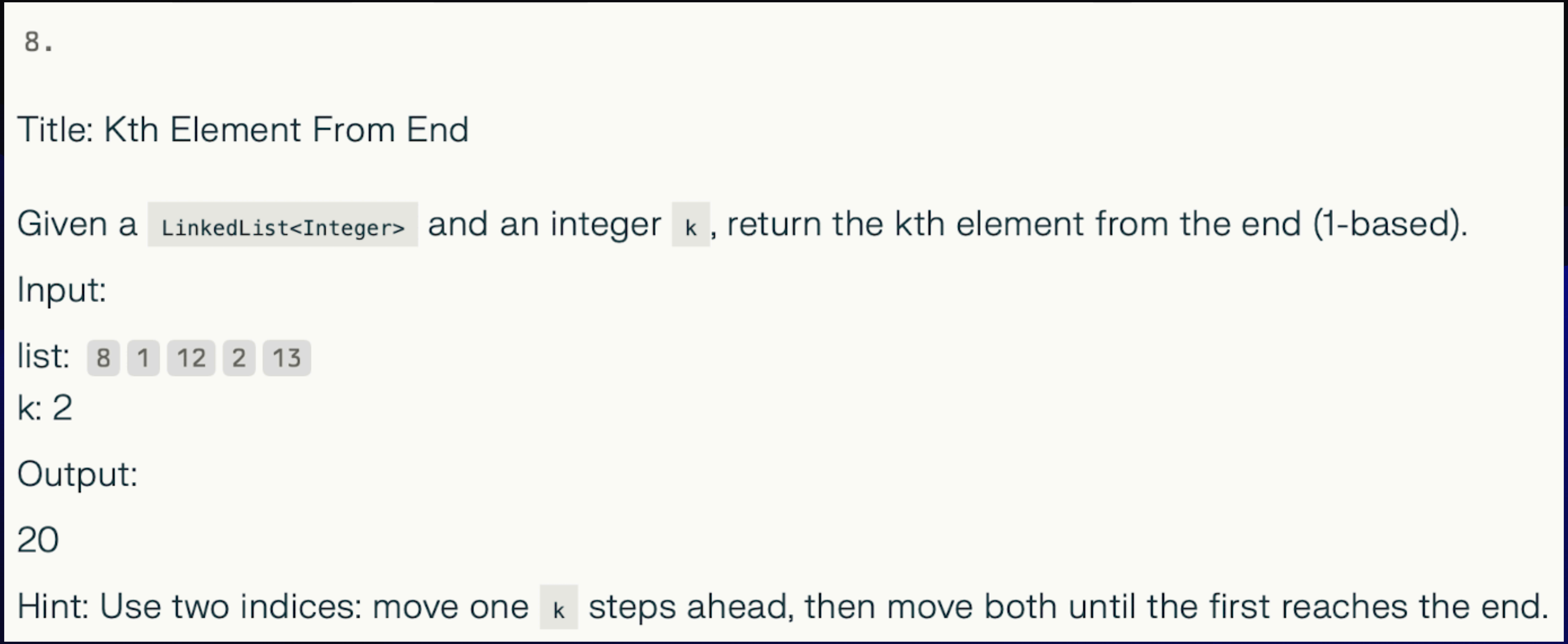
list: 1 2 3 4 5
k: 2

Output:

4 5 1 2 3

Hint: Use `removeLast()` and `addFirst()` in a loop, or compute effective `k` using size.

QUESTION 8



8.

Title: Kth Element From End

Given a `LinkedList<Integer>` and an integer `k`, return the kth element from the end (1-based).

Input:

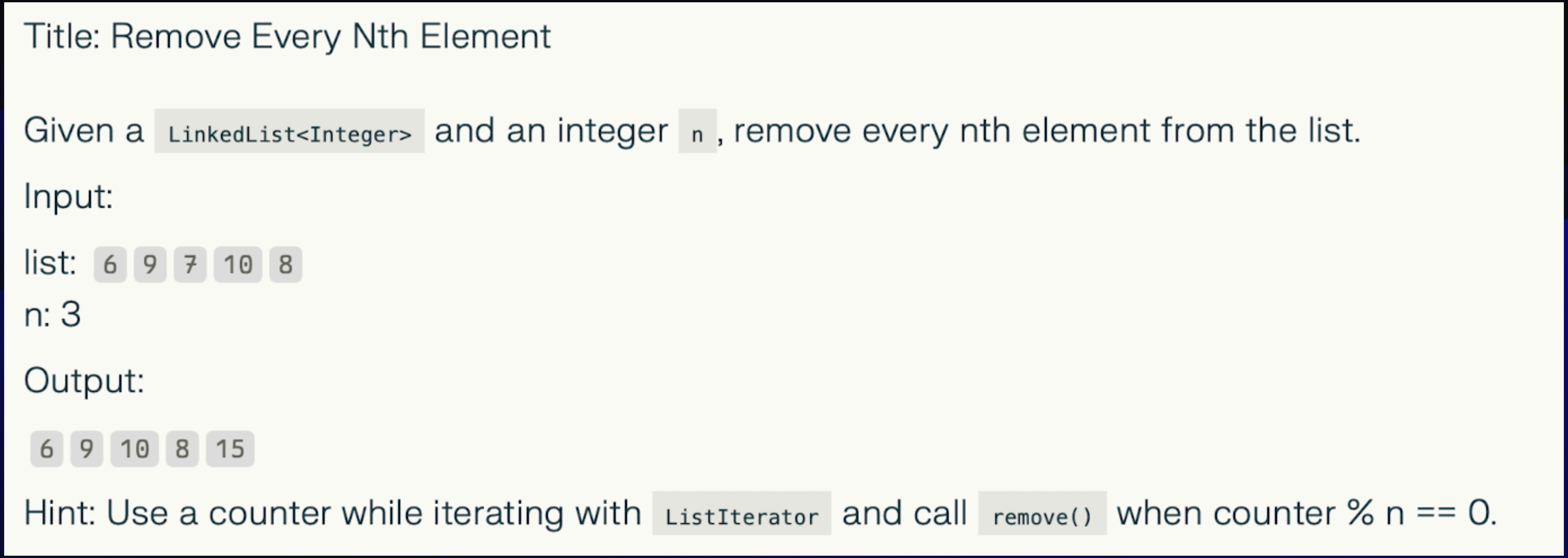
list: 8 1 12 2 13
k: 2

Output:

20

Hint: Use two indices: move one `k` steps ahead, then move both until the first reaches the end.

QUESTION 9



Title: Remove Every Nth Element

Given a `LinkedList<Integer>` and an integer `n`, remove every nth element from the list.

Input:

list: `6` `9` `~~7~~` `10` `8`

n: 3

Output:

`6` `9` `10` `8` `15`

Hint: Use a counter while iterating with `ListIterator` and call `remove()` when counter % n == 0.