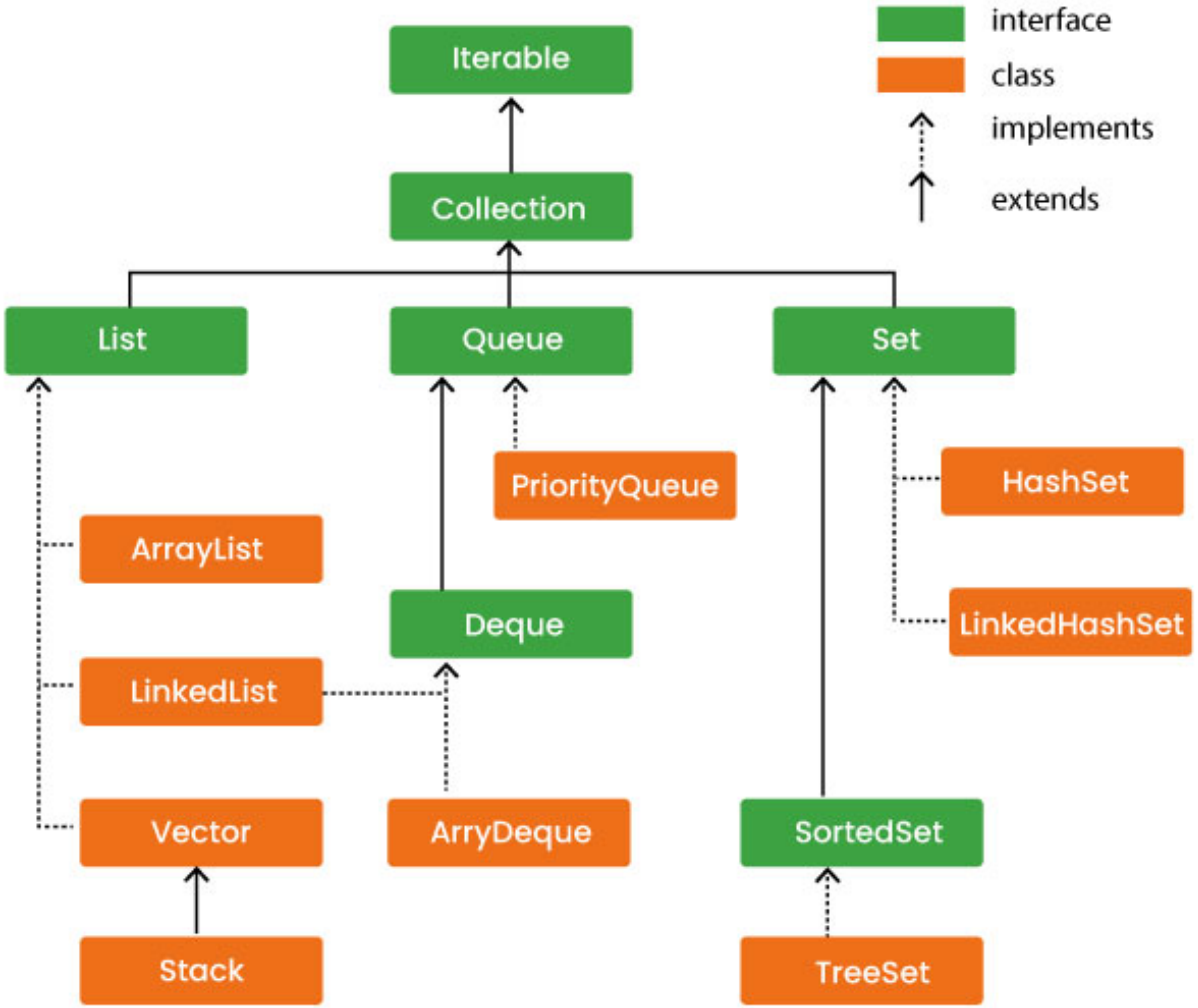
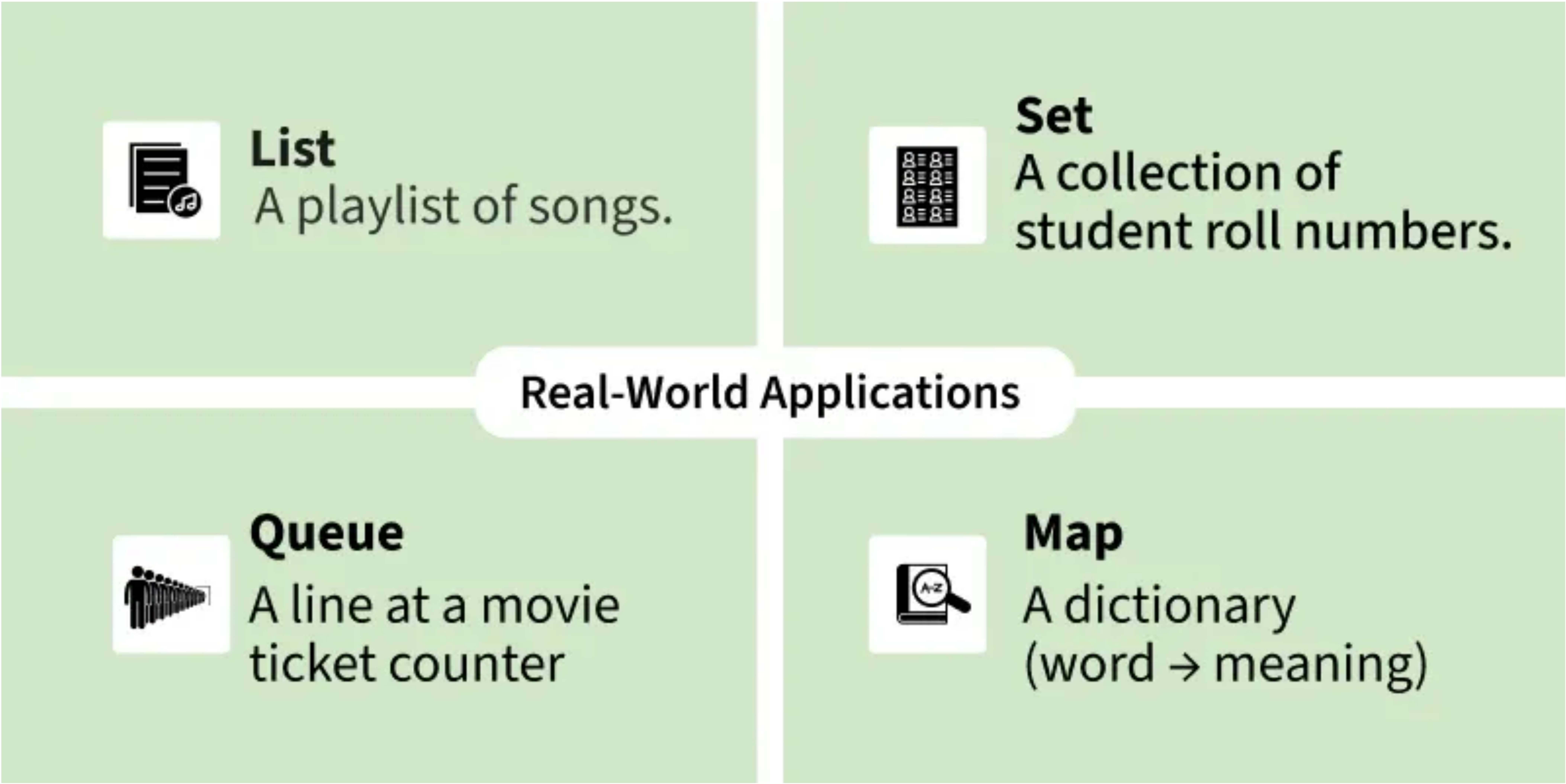
Collections

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

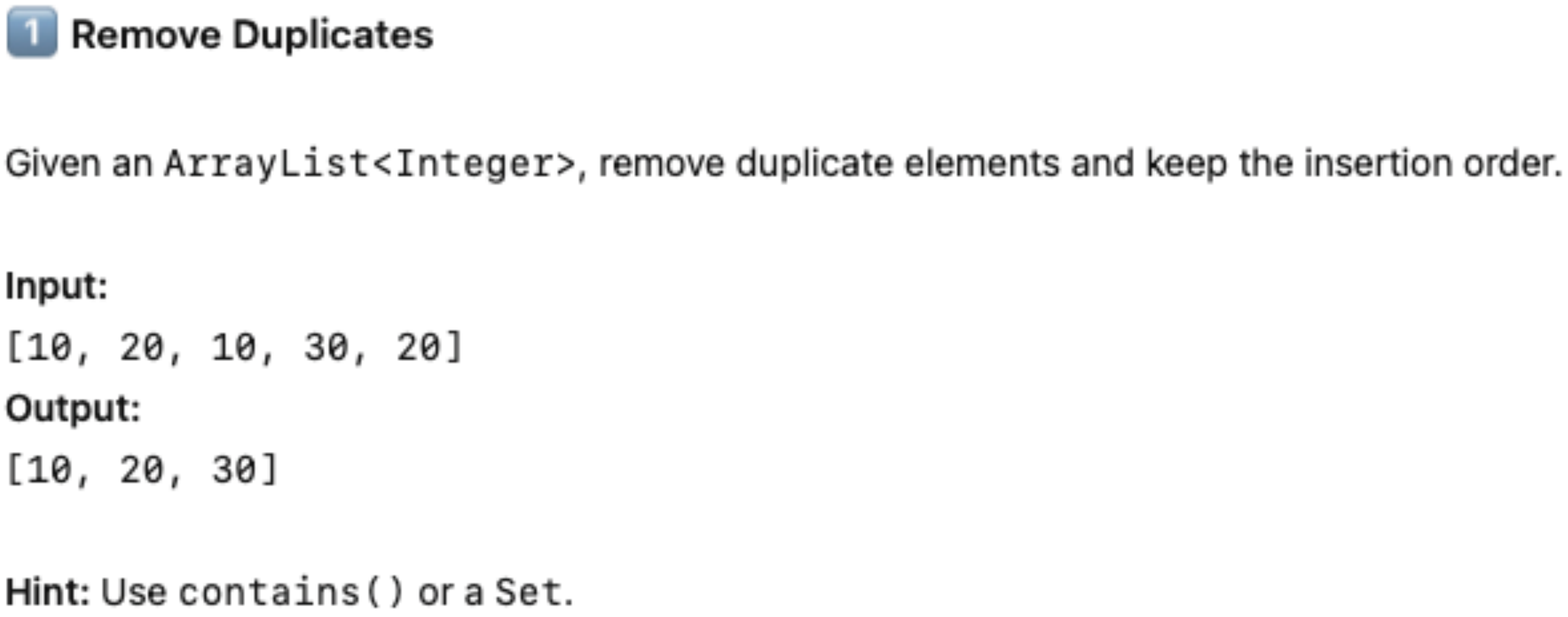
1. **For loop**: used when index access is needed
2. **Enhanced for loop (for-each loop)**: best for read only traversal
3. **forEach() method**: functional programming, uses when concise iteration
4. **While loop**: useful when loop condition is not index-based.
5. **Iterator Interface**: allows safe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

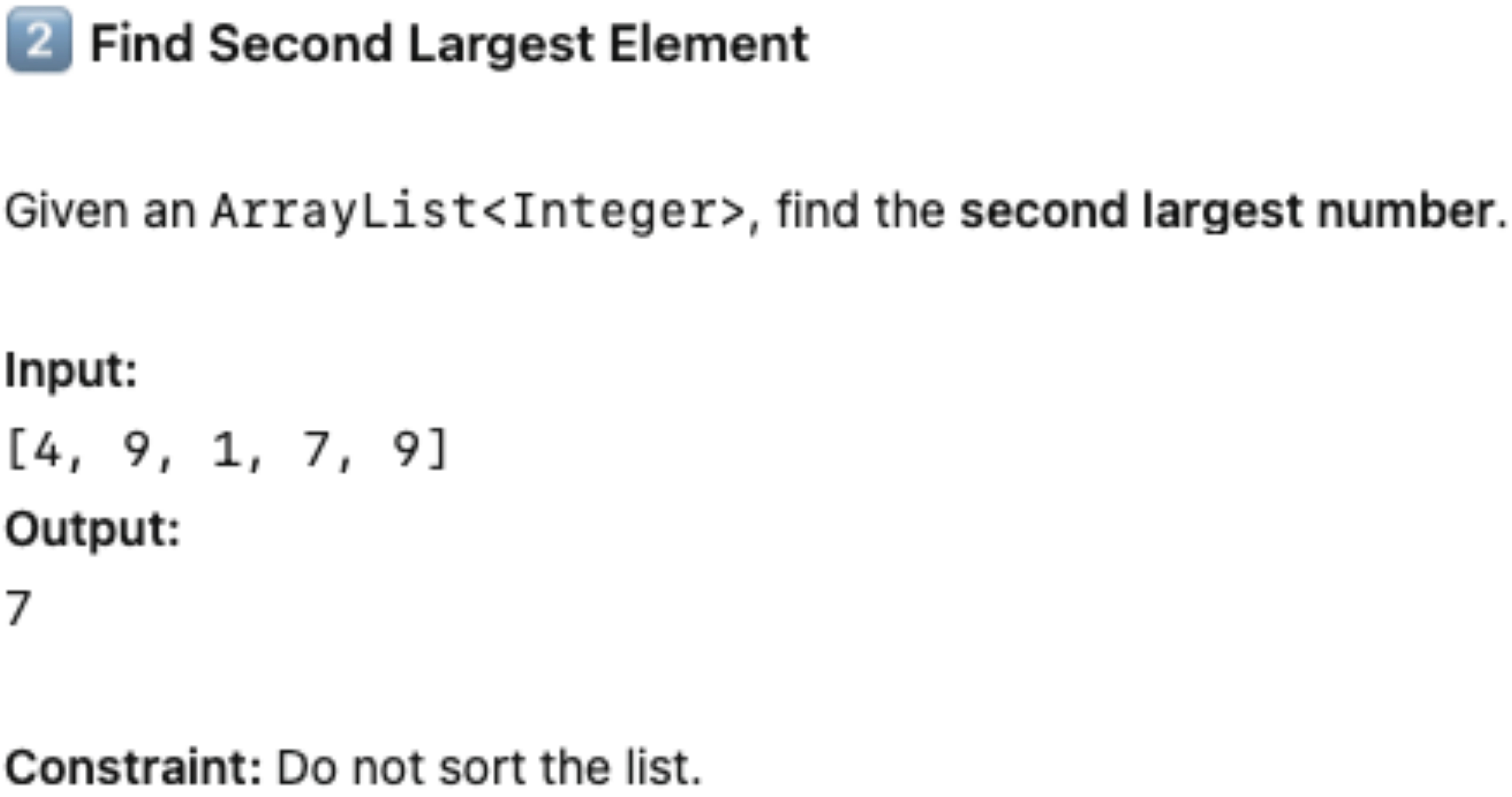
```
[10, 20, 10, 30, 20]
```

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:

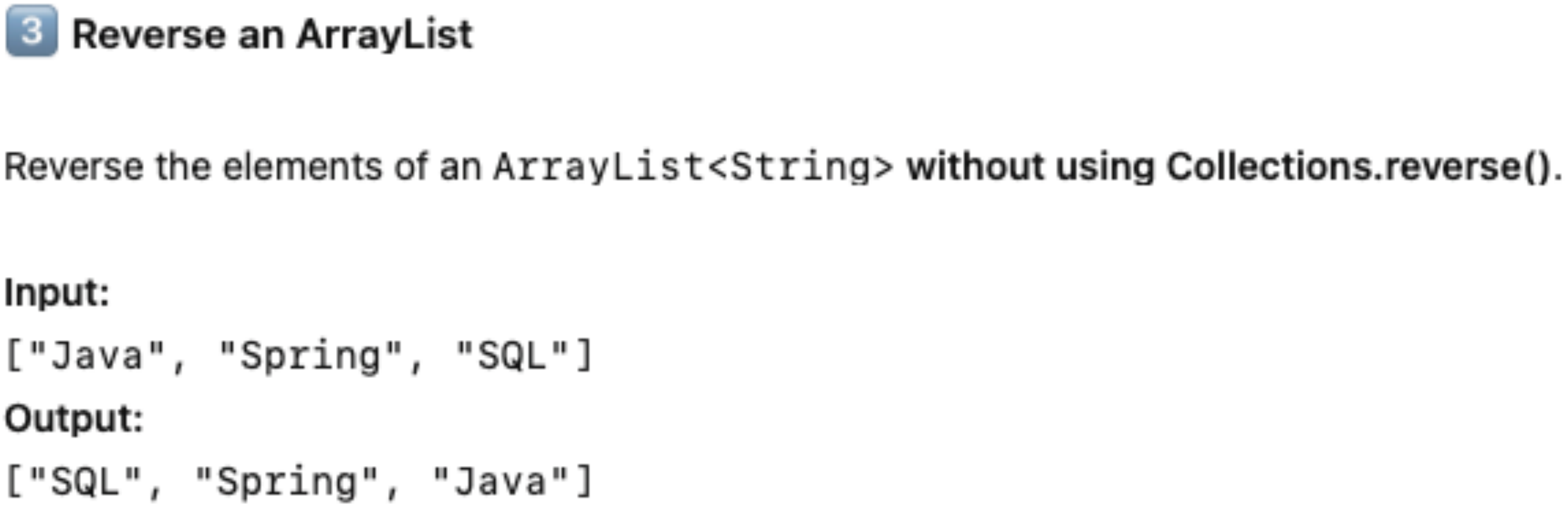
```
[4, 9, 1, 7, 9]
```

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()**.

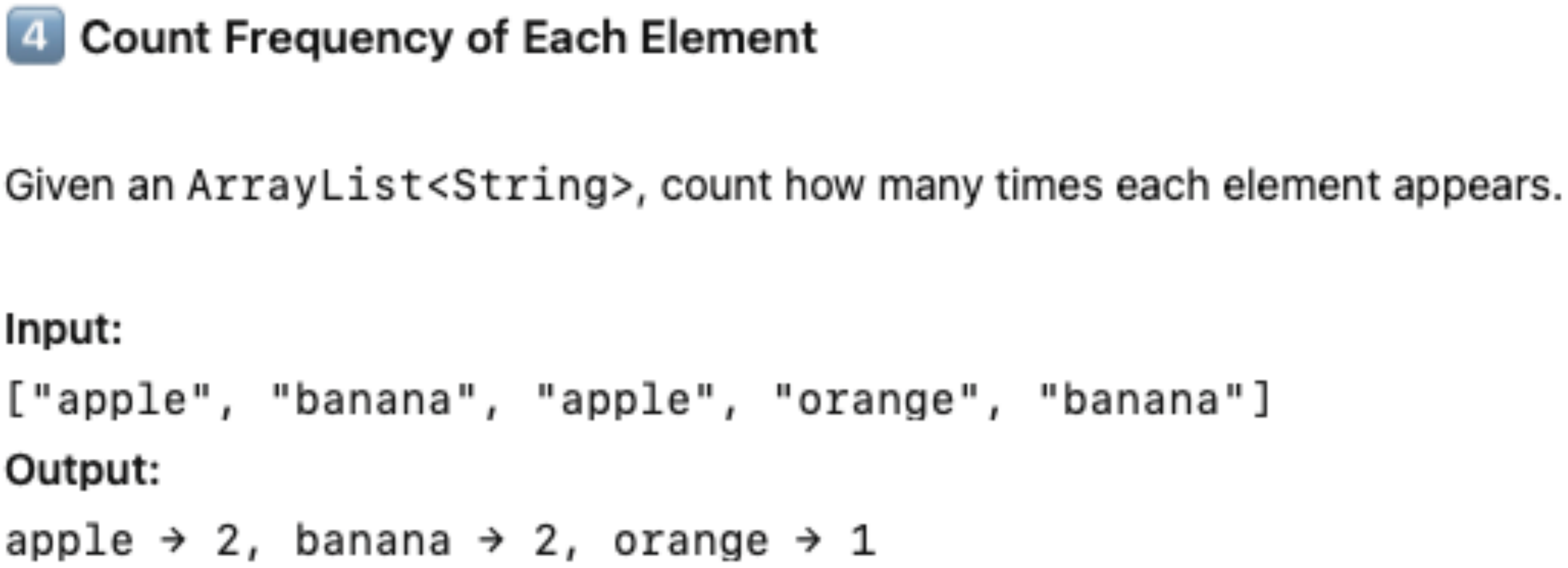
Input:

```
["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.

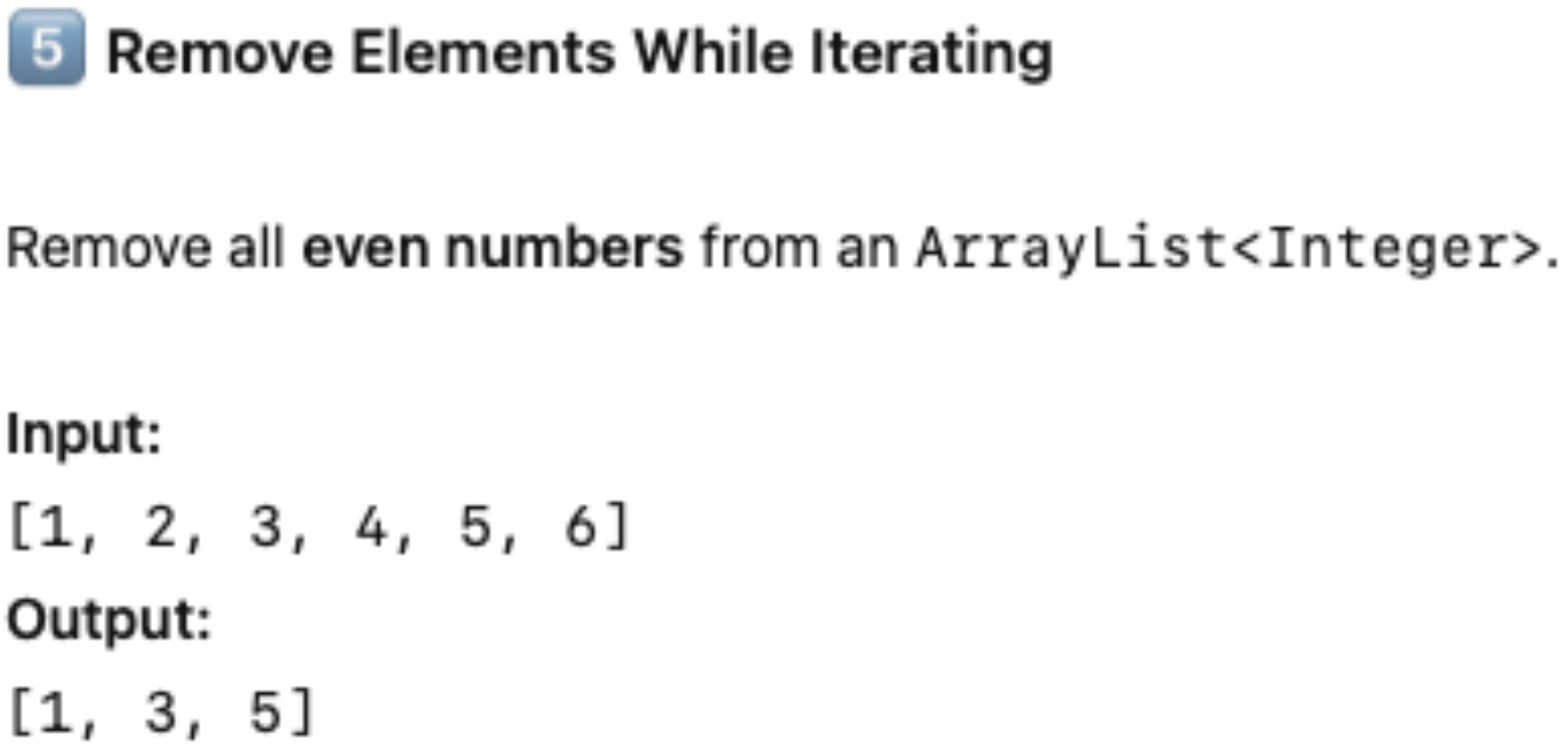
Input:

```
["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>`.

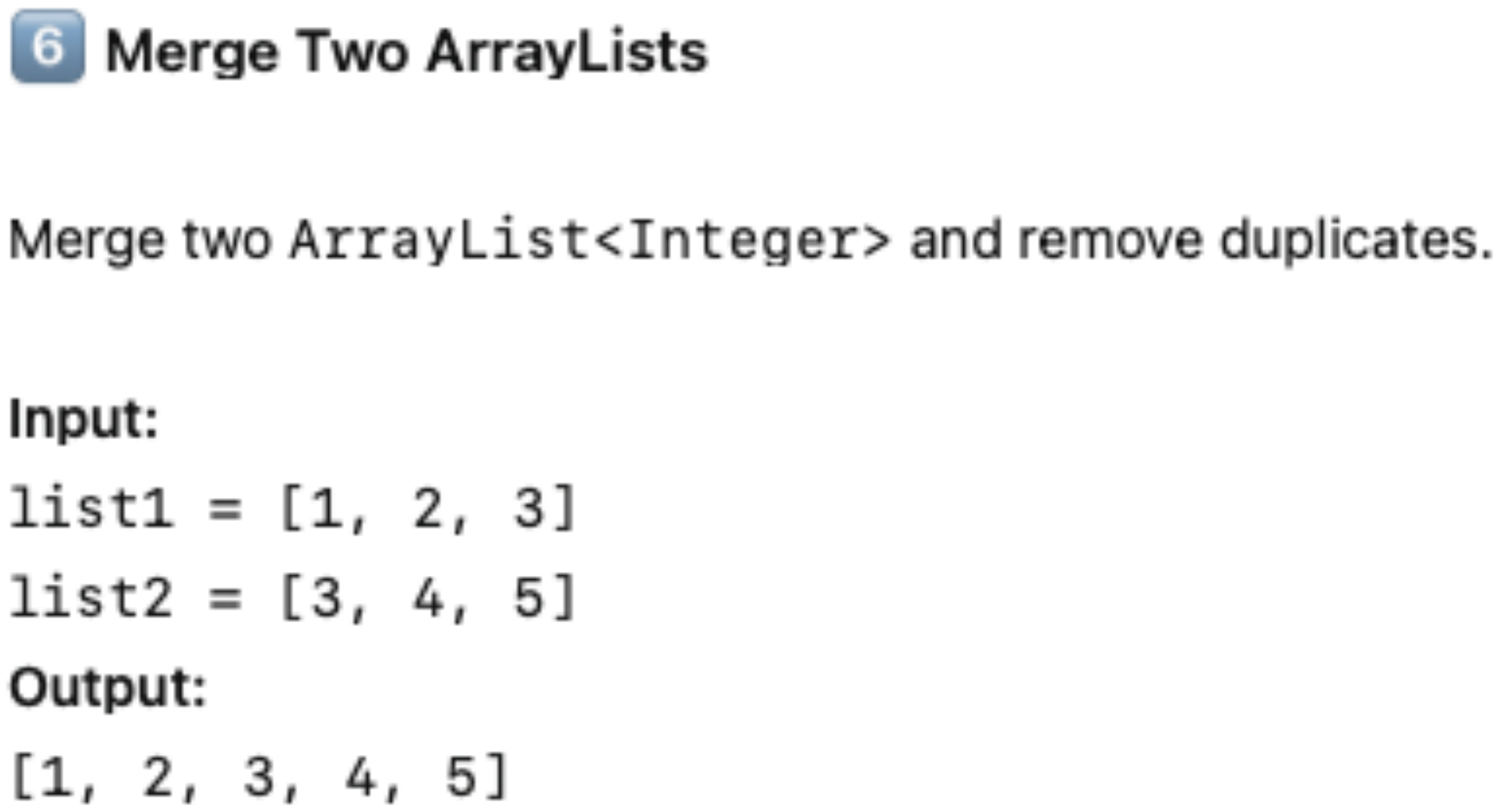
Input:

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

Output:

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

Question 6



6️⃣ Merge Two ArrayLists

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

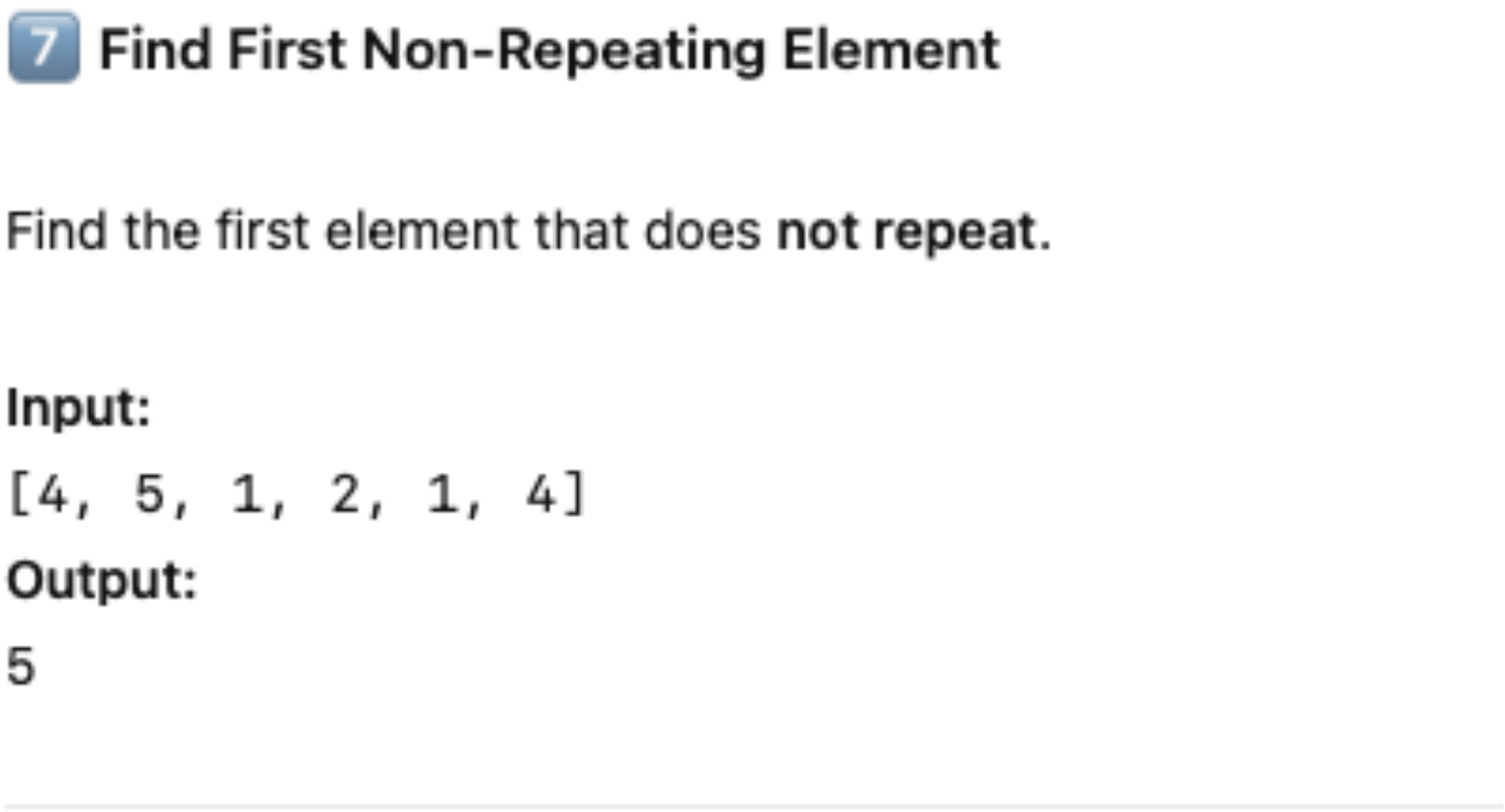
Input:

```
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:

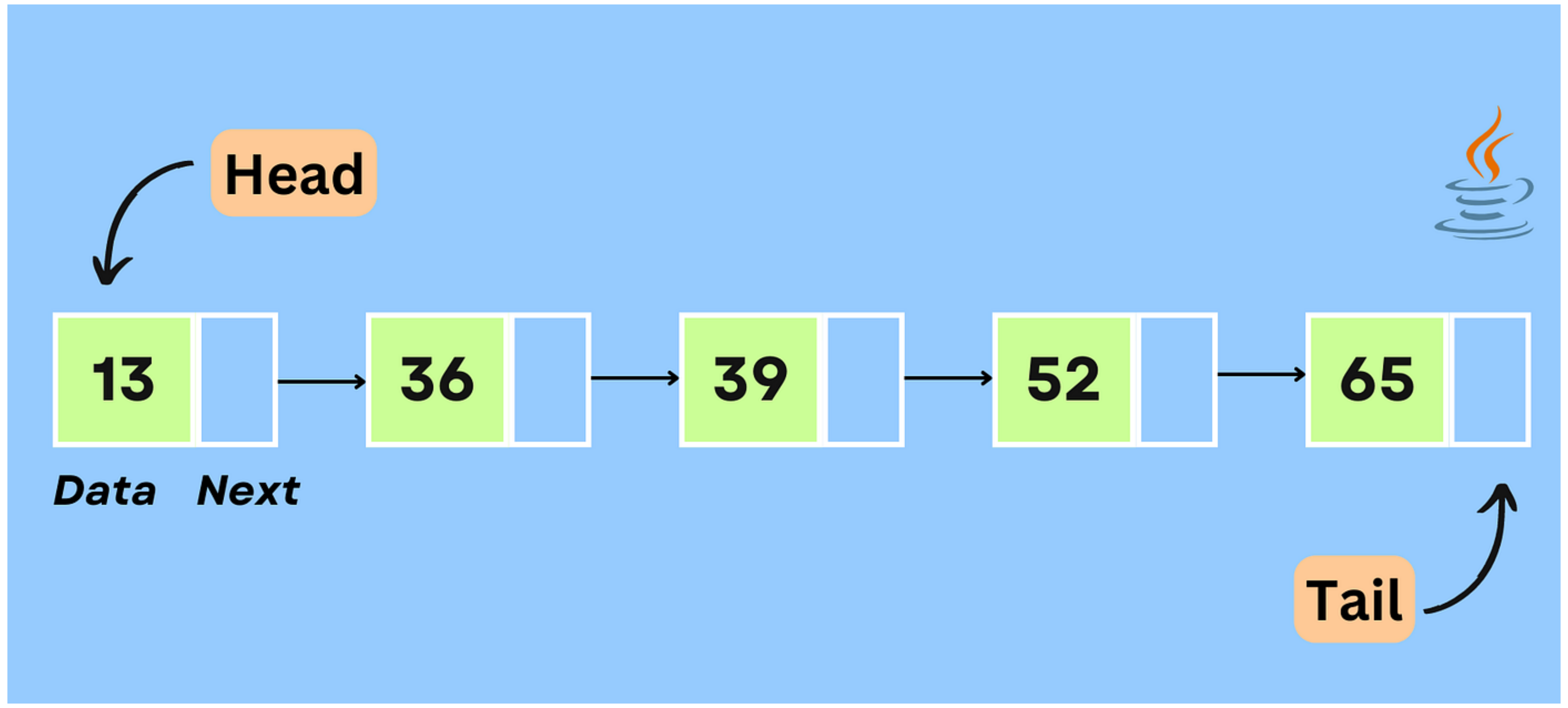
```
[4, 5, 1, 2, 1, 4]
```

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.
- **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. **While loop**

4. **Iterator** - forward iteration only

5. **ListIterator** - for bidirectional iteration (hasPrevious/previous), descendingIterator reverse order iteration

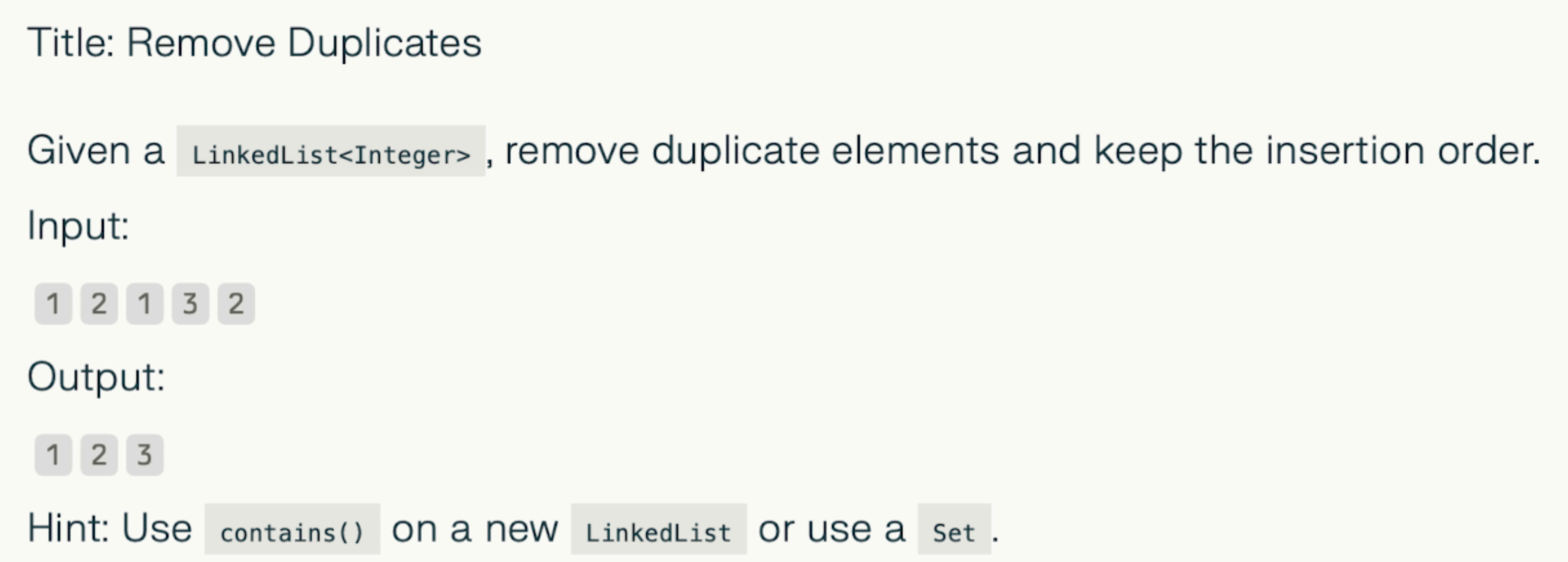
    1. ListIterator allows modification during iteration (safe concurrent changes)

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:

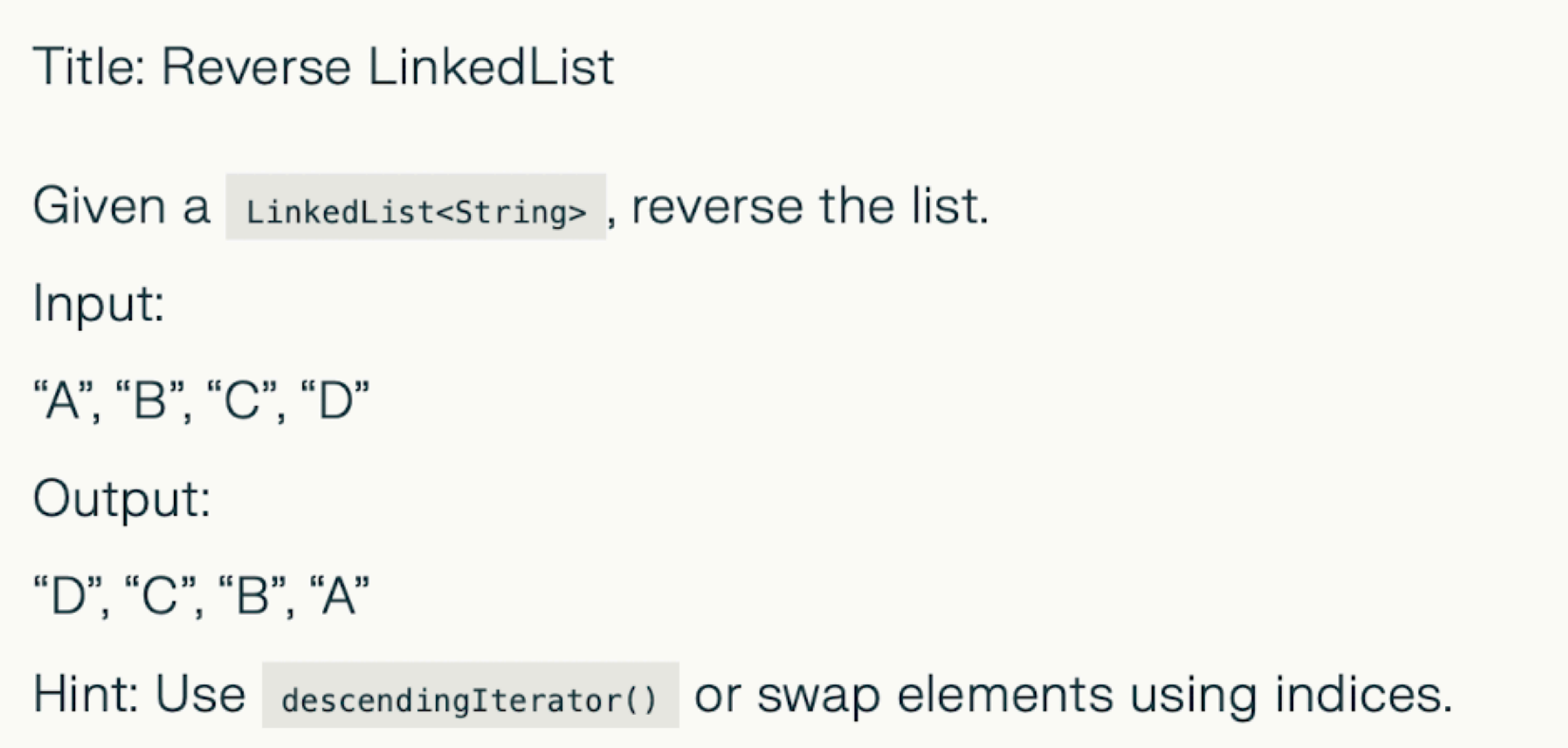
`1` `2` `1` `3` `2`

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:

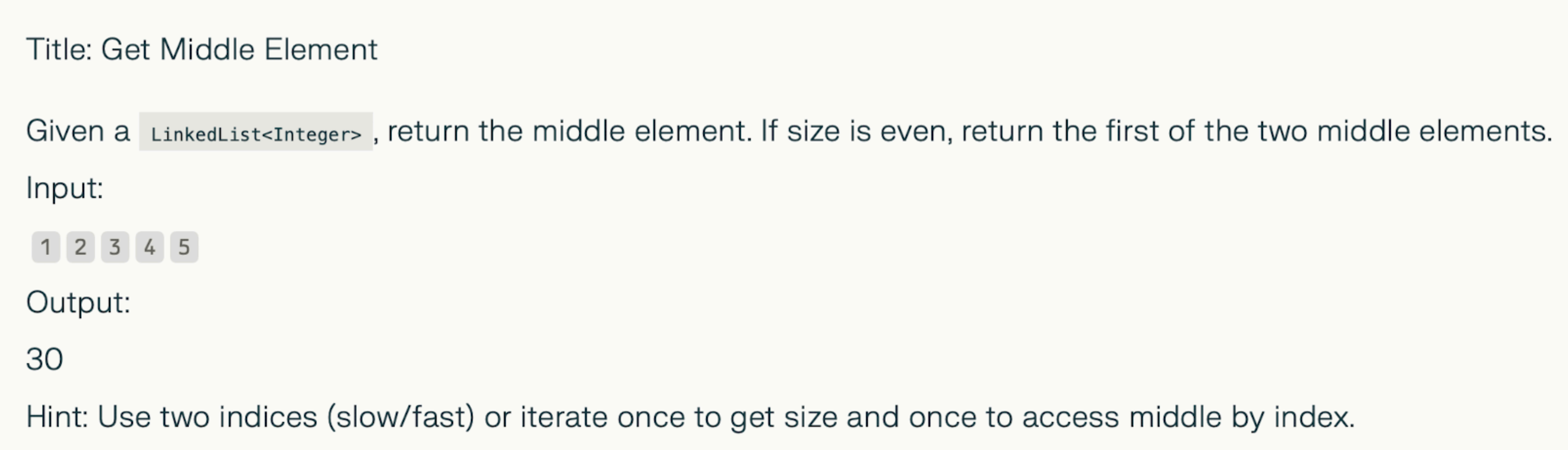
“A”, “B”, “C”, “D”

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:

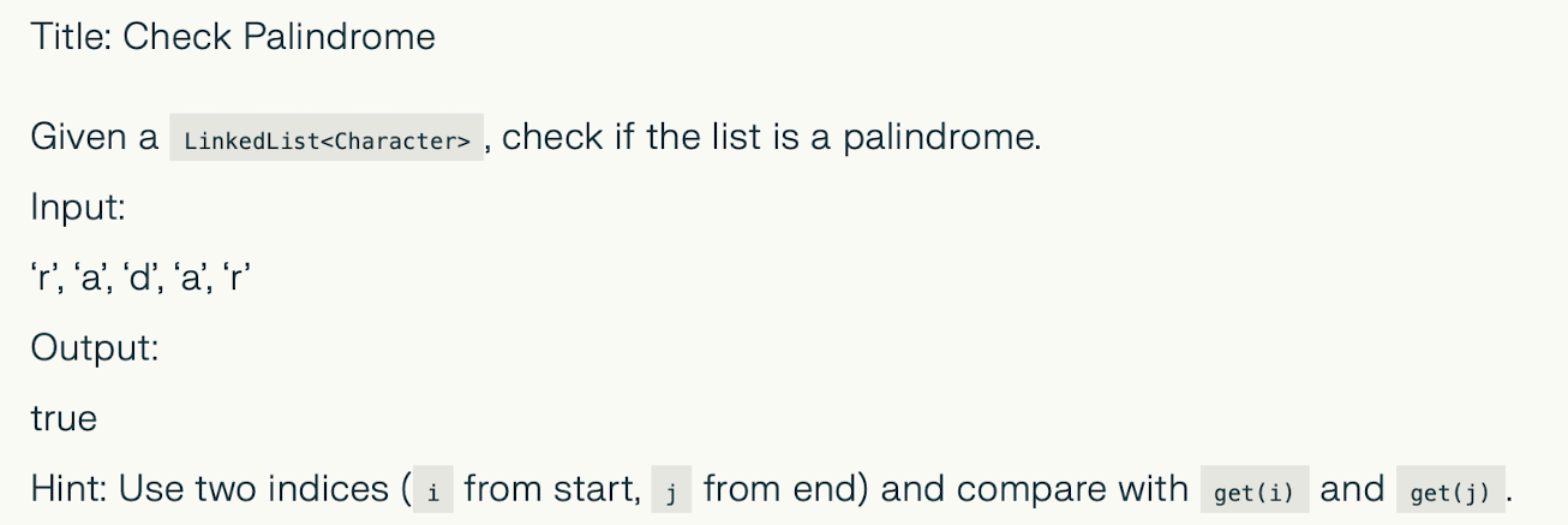
1 2 3 4 5

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:

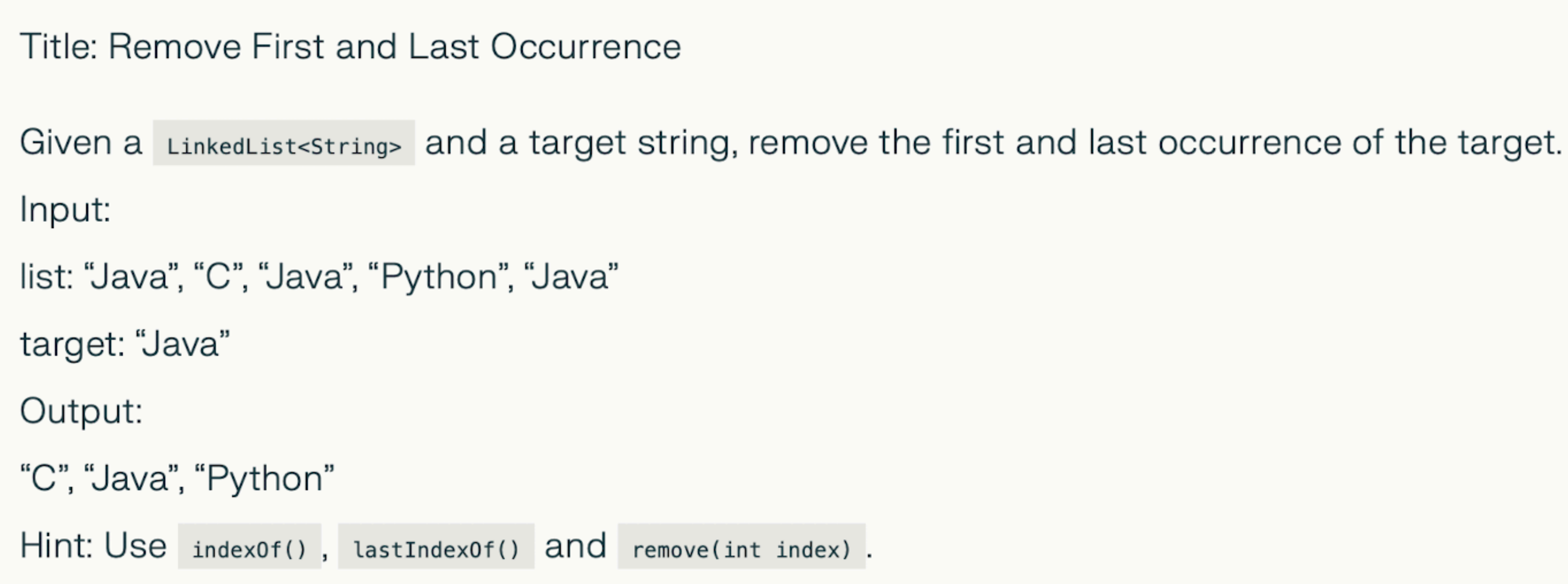
‘r’, ‘a’, ‘d’, ‘a’, ‘r’

Output:

true

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

QUESTION 5



Title: 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)`.

Vector

- Vector is dynamic and ordered collections of elements like ArrayList
- Vector is synchronized (thread safe) where as ArrayList is not synchronize
- Use Vector only if you specifically need synchronized list

Stack

- A **stack** is a linear data structure that follows **LIFO** — *Last In, First Out*.
- You can **add (push)** and **remove (pop)** elements only from the **top**.
- Key operations:
    - Push → add element
    - Pop → remove top element
    - Peek → see top element without removing
- In java Stack extends Vector class and implements List interface
- As Stack extends Vector so it’s synchronized (thread safe)

Imp Method

- **push(E item)** – adds an element to the top of the stack
- **pop()** – removes and returns the top element
- **peek()** – returns the top element without removing it
- **empty()** – checks whether the stack is empty
- **size()** – returns number of elements
- **get(int index)** – gets element at given index
- **firstElement()** – bottom element
- **lastElement()** – top element
- **remove(int index)** – removes element at index
- **contains(Object o)** – checks if element exists

\

Iterate Stack

- Same way as we iterate any implementation of List interface

Question

1: Valid Parentheses

Question Statement

Given a string containing only (), {}, [], check whether the brackets are balanced.

Input: “{[()]}"

Output: true

Hint:

Use a Stack. Push opening brackets, pop when closing bracket appears.

2: Next Greater Element

Question Statement

Given an array, for each element find the **next greater element to its right**.

If none exists, print -1.

Input: [4,5,2,10]

Output: [5, 10, 10, -1]

Hint:

Use a **monotonic stack** to keep elements in decreasing order.

3: Reverse a String

Question Statement

Given a string, reverse it using a Stack

Input: “hello”

Output: “olleh”

Hint:

Push each character into Stack, then pop to build reversed string.

4: Evaluate Postfix Expression

Question Statement

Given a postfix expression, evaluate it using Stack.

Input: "2 3 1 * + 9 -“

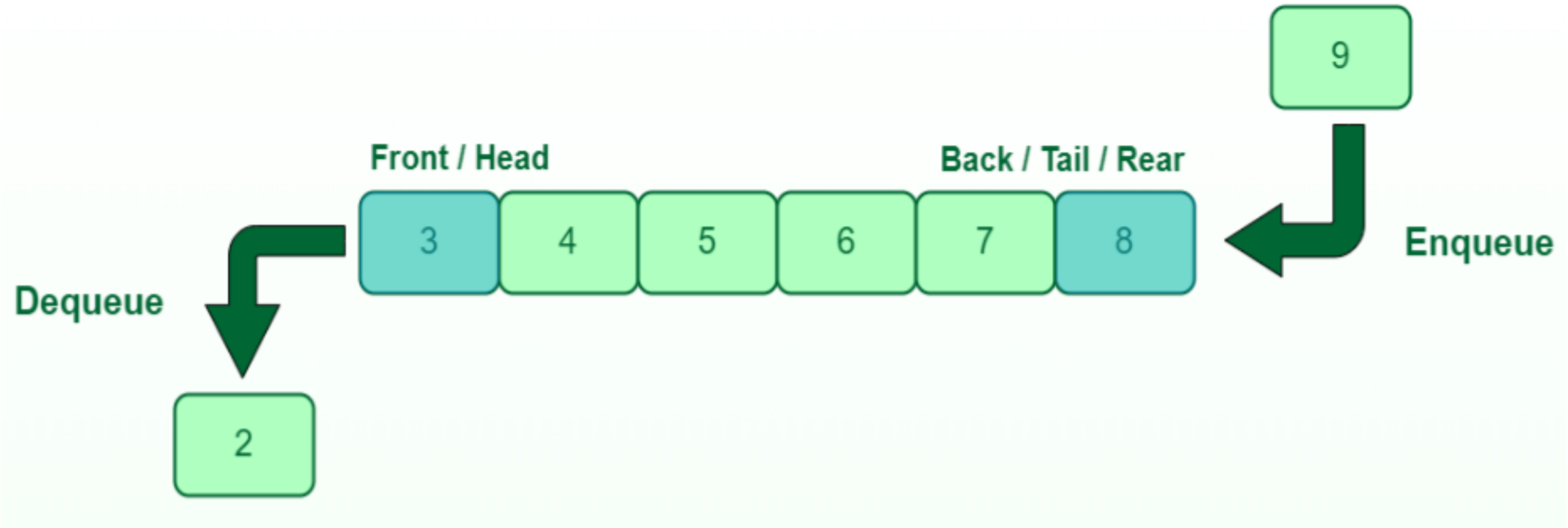
Output: -4

Hint:

Push operands. On operator, pop two values and apply operation.

Queue Interface

- Queue is interface in Java, available in the Util package
- **Queue** is a linear data structure that follows **FIFO (First In First Out)**
- Basic operations of Queue are
  - **Enqueue** → add element at rear
  - **Dequeue** → remove element from front



Implementation of Queue Interface

- ArrayDeque
- PriorityQueue

ArrayDeque

- ArrayDeque is a resizable-array implementation of Deque (Double-Ended Queue)

- It allows insertion and removal from **both front and rear**

-

Imp Method

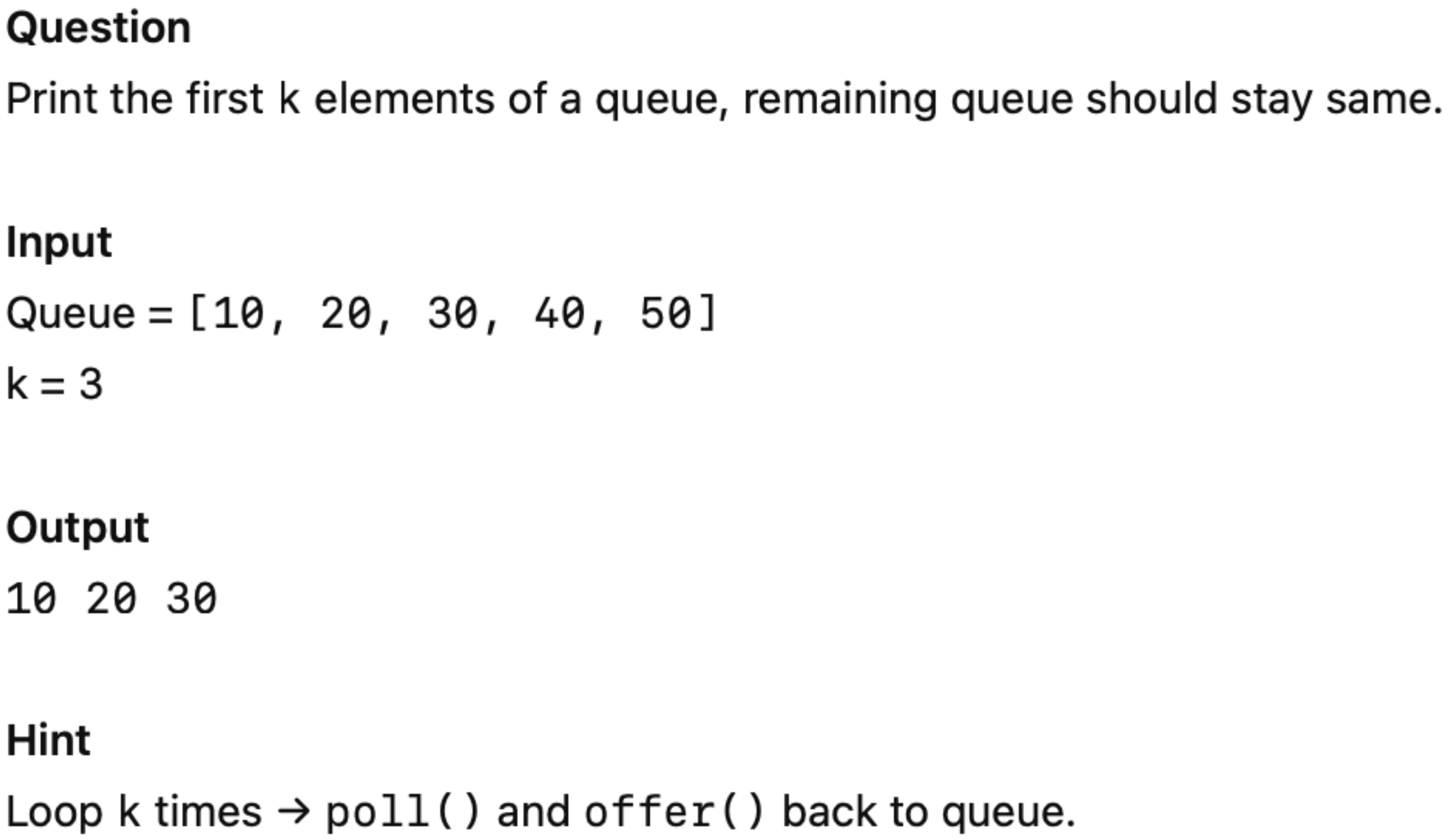
1. offer(e) - enqueue element, add element at rear
2. poll() - dequeue, remove element from front
3. peek() - view front element

Ways to Iterate Queue

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

Question

1: Print First K Elements of Queue



Question

Print the first k elements of a queue, remaining queue should stay same.

Input

Queue = `[10, 20, 30, 40, 50]`

k = 3

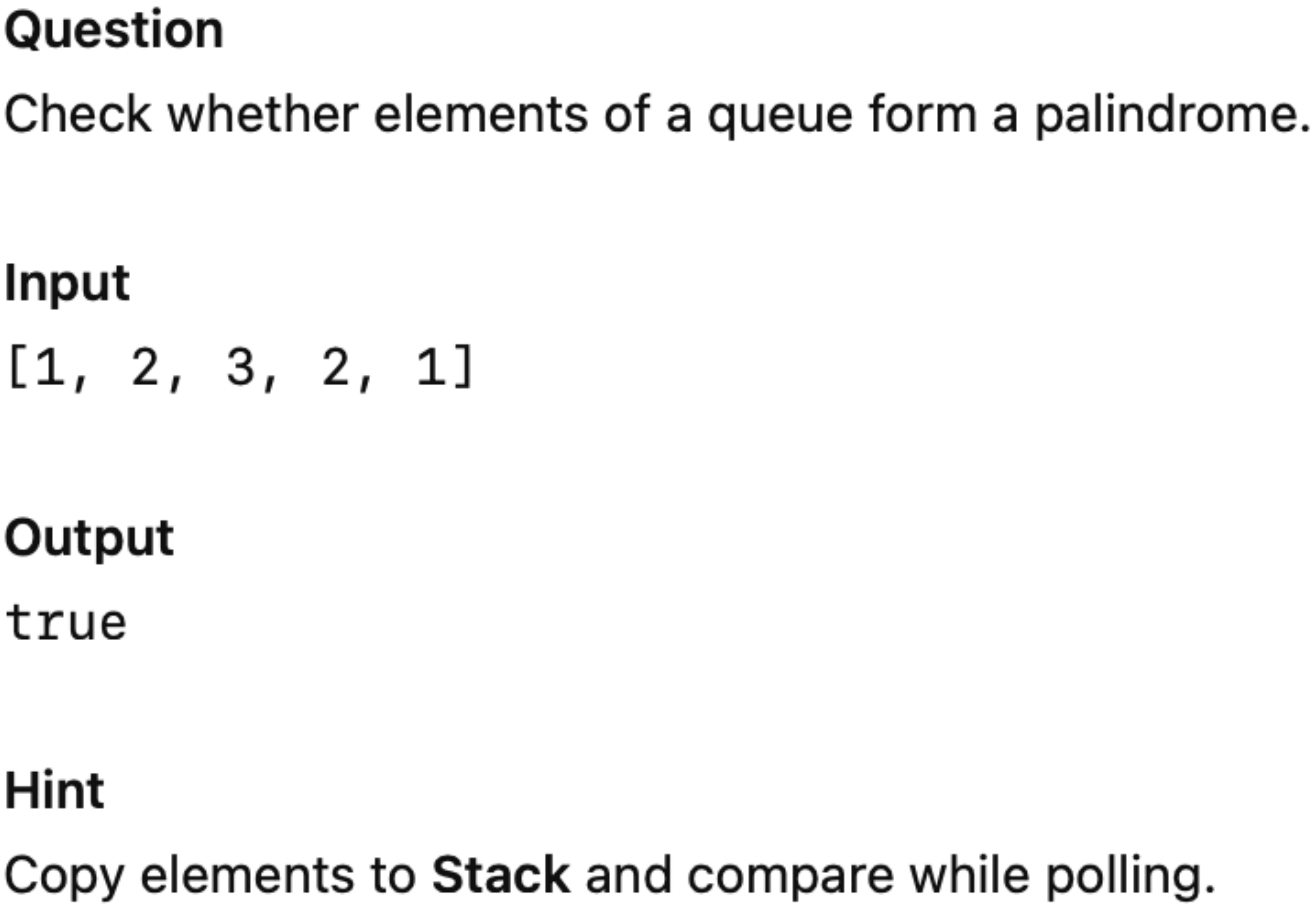
Output

```
10 20 30
```

Hint

Loop k times → `poll()` and `offer()` back to queue.

2: Check if a Queue is Palindrome



Question

Check whether elements of a queue form a palindrome.

Input

```
[1, 2, 3, 2, 1]
```

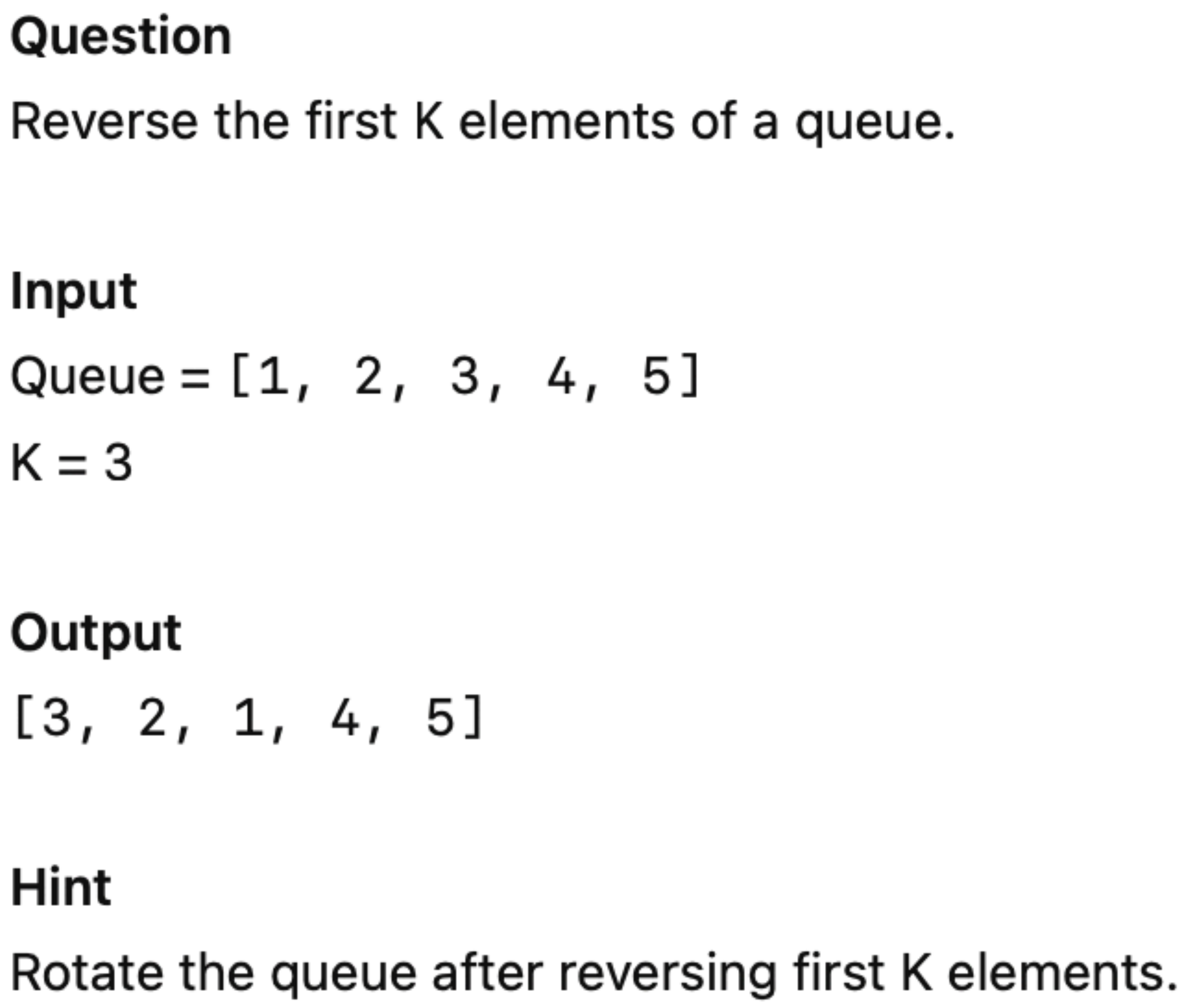
Output

```
true
```

Hint

Copy elements to **Stack** and compare while polling.

3: Reverse First K Elements of Queue



Question

Reverse the first K elements of a queue.

Input

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

K = 3

Output

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

Hint

Rotate the queue after reversing first K elements.

PriorityQueue

- **PriorityQueue** is a special type of Queue where elements are processed (add & remove) **based on priority**, not FIFO

Set Interface

- **Set** is a collection that stores **unique elements.**
- **Set** does not allow duplicate and does does not gurantee order
- In java Set is interface, defined in Util package

Implementation of Set

1. HashSet
2. LinkedHashSet
3. TreeSet

HashSet

- HashSet is class in Util package that implement Set interface.
- It store unique element and does not mantain insertion order.
- HashSet internally uses HashMap.
- Each element we store in HashSet is stored as key in HashMap with constant dummy value.

Imp Method

- add(E e) → adds element if not already present
- addAll(Collection<? extends E> c) → adds multiple elements
- remove(Object o) → removes an element
- removeAll(Collection<?> c) → removes multiple elements

Ways to Iterate HashSet

1. ~~**For loop**: used when index access is needed~~

2. **Enhanced for loop (for-each loop)**: best for read only traversal

3. **forEach() method**: functional programming, uses when concise iteration

4. ~~**While loop**: useful when loop condition is not index-based.~~

5. **Iterator Interface**: allows safe removal of elements during iteration

6. **ListIterator** - for bidirectional iteration (hasPrevious/previous), descendingIterator reverse order iteration

Question

1: Find duplicate elements in an array

Question Statement

Given an integer array, print all duplicate elements.

Input

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

Output

2 1

Hint

Try inserting elements into a HashSet. If insertion fails, it is a duplicate.

2: Intersection of two arrays

Question Statement

Print common elements from two arrays.

Input

Array1: [1, 2, 3, 4]

Array2: [3, 4, 5]

Output

3 4

Hint

Store first array in HashSet and check elements of second array.

3: Longest substring without repeating characters

Question Statement

Find the length of the longest substring without repeating characters.

Input

“abcabcbb"

“abcaaxyzabcbb”

Output

3

6

Hint

Use sliding window technique with HashSet.

LinkedHashSet

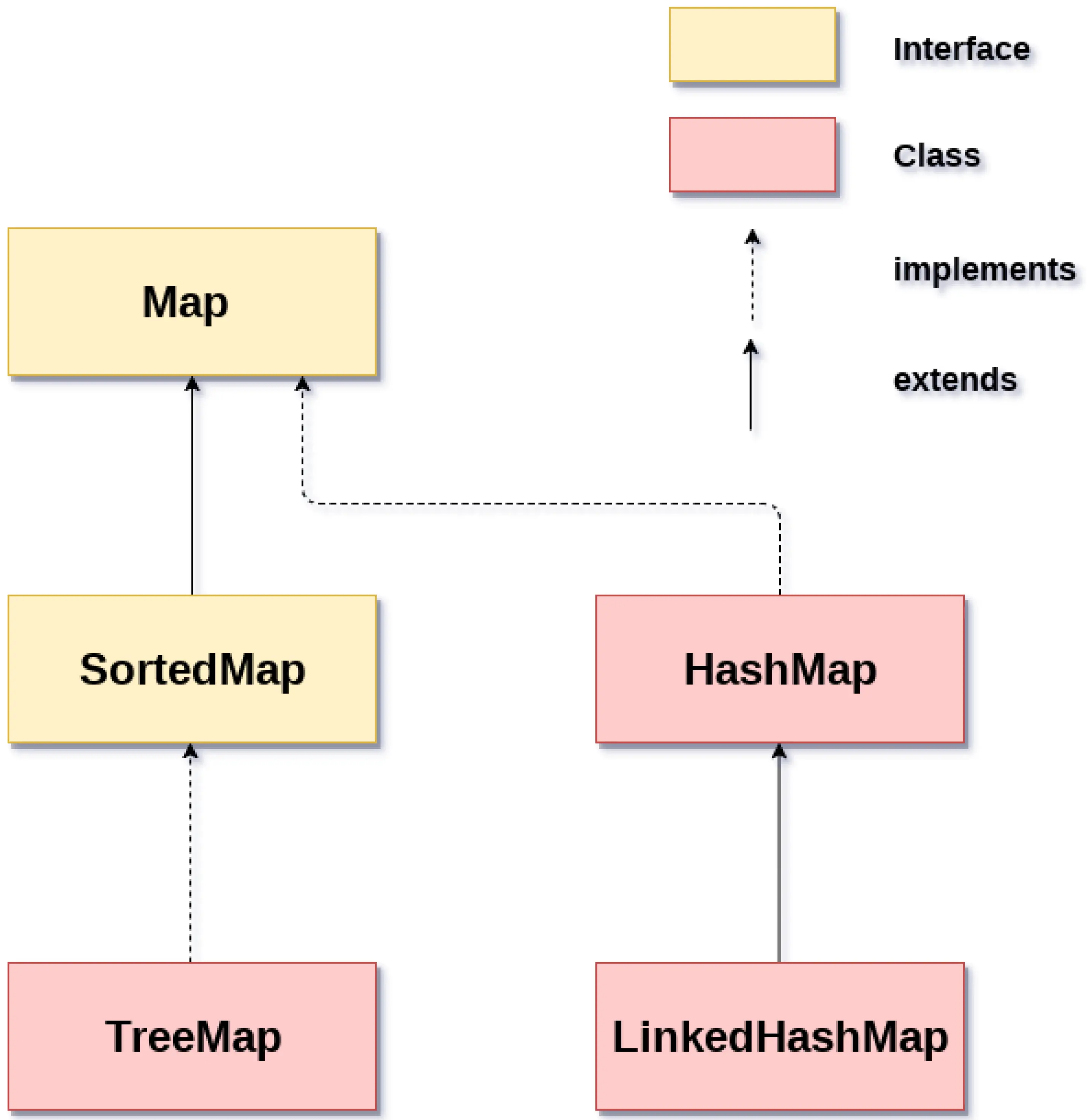
- LinkedHashSet stores unique elements and **maintains insertion order**
- LinkedHashSet uses double linked list to maintain insertion order

TreeSet

- TreeSet stores unique elements in **sorted order**
- TreeSet internally uses a Red Black Tree (self balancing BST)

Map Interface

- Map is an **interface** in java.util.
- It is used to store data in **key–value pairs.**
- Keys are unique, values can be duplicated.



Implementation of Map

- **HashMap** → hash table, no order
- **LinkedHashMap** → insertion order
- **TreeMap** → sorted by key (Red-Black Tree)
- **HashTable** → synchronized, legacy
- **ConcurrentHashMap** → thread-safe (modern)

HashMap

- Part of the Util package and stores key-value pairs.
- It is based on a hash table.
- Allows one null key, and values can be multiple nulls
- Unordered and not synhcronized.

Imp Method

- **put(K key, V value)** → insert/update pair in hash map
- **putIfAbsent(K key, V value)** → insert only
- **get(Object key)** → get value
- **getOrDefault(Object key, V defaultValue)** → get or return defaultValue
- **remove(Object key)** → delete entry
- **containsKey(Object key)** → search key and return boolean
- **containsValue(Object value)** → search value and return boolean
- **keySet()** → returns all keys
- **values()** → returns all values
- **entrySet()** → key–value pairs

Ways to Iterate HashMap

1. ~~**For loop**: used when index access is needed~~

2. **For each loop**: using entrySet method of HashMap [mostly used]

3. **forEach() method**: functional programming, uses when concise iteration

4. ~~**While loop**: useful when loop condition is not index-based.~~

5. ~~**Iterator Interface**: similar like for each loop again we have to iterate on entrySet~~

6. ~~**ListIterator** - no use~~

Questions

1: Frequency of elements

Problem Statement

Count frequency of each element in an array.

Input

[1, 2, 2, 3, 1, 4]

Output

1=2, 2=2, 3=1, 4=1

Hint

Use element as key and count as value.

2: First non-repeating character

Problem Statement

Find first character that does not repeat in a string.

Input

"swiss"

Output

w

Hint

Store character count using HashMap, then traverse string again.

3: Two Sum problem (VERY FAMOUS)

Problem Statement

Find indices of two numbers whose sum equals target.

Input

Array: [2, 7, 11, 15]

Target: 9

Output

[0, 1]

Hint

Store number and index in HashMap.

Generics Programming

- **Generics** is a style of programming in which programs are written in terms of types to be specified later.

- It allows writing type-independent (generic) code.

- It enables functions, classes, and data structures to work with **any data type**

Generic Method

1. Get a Parameter in Generic Style - method can accept any parameter type

2. Return a generic variable - method decides return type dynamically

3. Do Both (Accept and Return Generic Type)

Generic Class

1. How to write a generic class

2. How java collection framework use generic programming

Question

1: Generic Method – Print Any Type

Problem Statement

Write a generic method that accepts a value of any type and prints it.

Input

```
printValue(10)

printValue("Java")
```

Expected Output

```
10
Java
```

2: Generic Method – Return Same Type

Problem Statement

Create a generic method that takes a value and returns the same value.

Input

getValue(5)

getValue("Hello")

Expected Output

```
5
Hello
```

3: Generic Class – Store and Retrieve Value

Problem Statement

Create a generic class Box<T> that stores a value in a private global variable and returns it.

Input

Box<Integer> → 100

Box<String> → "Java"

Expected Output

```
100
Java
```