List Interface

1. **Reverse a List**
   Write a program to reverse the elements of a given List without using built-in reverse methods. Implement it for both ArrayList and LinkedList.
   Example:
   Input: [1, 2, 3, 4, 5] → Output: [5, 4, 3, 2, 1].

```
import java.util.*;

public class ReverseList {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of elements:");
        int n = sc.nextInt();
        ArrayList<Integer> arrayList = new ArrayList<>();
        LinkedList<Integer> linkedList = new LinkedList<>();

        System.out.println("Enter elements:");
        for (int i = 0; i < n; i++) {
            int val = sc.nextInt();
            arrayList.add(val);
            linkedList.add(val);
        }

        System.out.println("Reversed ArrayList:");
        for (int i = n - 1; i >= 0; i--) {
            System.out.print(arrayList.get(i) + " ");
        }

        System.out.println("\nReversed LinkedList:");
        for (int i = n - 1; i >= 0; i--) {
            System.out.print(linkedList.get(i) + " ");
        }
    }
}
```

2. **Find Frequency of Elements**
   Given a list of strings, count the frequency of each element and return the results in a Map<String, Integer>.
   Example:
   Input: ["apple", "banana", "apple", "orange"] → Output: {apple=2, banana=1, orange=1}.

```java
import java.util.*;

public class FrequencyCounter {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of strings:");
        int n = sc.nextInt();
        sc.nextLine();
        List<String> list = new ArrayList<>();
        System.out.println("Enter strings:");
        for (int i = 0; i < n; i++) {
            list.add(sc.nextLine());
        }

        Map<String, Integer> freq = new HashMap<>();
        for (int i = 0; i < list.size(); i++) {
            String s = list.get(i);
            freq.put(s, freq.getOrDefault(s, 0) + 1);
        }
        System.out.println(freq);
    }
}
```

3. **Rotate Elements in a List**
   Rotate the elements of a list by a given number of positions.
   Example:
   Input: [10, 20, 30, 40, 50], rotate by 2 → Output: [30, 40, 50, 10, 20].

```java
import java.util.*;

public class RotateList {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of elements:");
        int n = sc.nextInt();
        List<Integer> list = new ArrayList<>();

        System.out.println("Enter elements:");
        for (int i = 0; i < n; i++) {
            list.add(sc.nextInt());
        }

        System.out.println("Enter rotate positions:");
        int k = sc.nextInt();
        k = k % n;

        List<Integer> rotated = new ArrayList<>();
        for (int i = k; i < n; i++) {
            rotated.add(list.get(i));
        }
        for (int i = 0; i < k; i++) {
            rotated.add(list.get(i));
        }

        System.out.println(rotated);
    }
}
```

4. **Remove Duplicates While Preserving Order**
   Remove duplicate elements from a list while maintaining the original order of elements.
   Example:
   Input: [3, 1, 2, 2, 3, 4] → Output: [3, 1, 2, 4].

```java
import java.util.*;

public class RemoveDuplicates {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of elements:");
        int n = sc.nextInt();
        List<Integer> input = new ArrayList<>();

        System.out.println("Enter elements:");
        for (int i = 0; i < n; i++) {
            input.add(sc.nextInt());
        }

        Set<Integer> seen = new HashSet<>();
        List<Integer> output = new ArrayList<>();

        for (int i = 0; i < input.size(); i++) {
            int val = input.get(i);
            if (!seen.contains(val)) {
                seen.add(val);
                output.add(val);
            }
        }
        System.out.println(output);
    }
}
```

5. **Find the Nth Element from the End**
   Given a singly linked list (use LinkedList), find the Nth element from the end without calculating its size.
   Example:
   Input: [A, B, C, D, E], N=2 → Output: D.

```java
import java.util.*;

public class NthFromEnd {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        LinkedList<String> list = new LinkedList<>();

        System.out.println("Enter number of elements:");
        int n = sc.nextInt();
        sc.nextLine();

        System.out.println("Enter elements:");
        for (int i = 0; i < n; i++) {
            list.add(sc.nextLine());
        }

        System.out.println("Enter N (from end):");
        int k = sc.nextInt();

        int fast = 0, slow = 0;
        while (fast < k) {
            fast++;
            if (fast >= list.size()) {
                System.out.println("Invalid input");
                return;
            }
        }
        while (fast < list.size() - 1) {
            slow++;
            fast++;
        }

        System.out.println(list.get(slow));
    }
}
```

Set Interface

1. **Check if Two Sets Are Equal**
   Compare two sets and determine if they contain the same elements, regardless of order.
   Example:
   Set1: {1, 2, 3}, Set2: {3, 2, 1} → Output: true.

```
import java.util.*;

public class EqualSets {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Set<Integer> set1 = new HashSet<>();
        Set<Integer> set2 = new HashSet<>();

        System.out.println("Enter number of elements in Set1:");
        int n1 = sc.nextInt();
        System.out.println("Enter elements of Set1:");
        for (int i = 0; i < n1; i++) {
            set1.add(sc.nextInt());
```

```java
        }

        System.out.println("Enter number of elements in Set2:");
        int n2 = sc.nextInt();
        System.out.println("Enter elements of Set2:");
        for (int i = 0; i < n2; i++) {
            set2.add(sc.nextInt());
        }

        System.out.println(set1.equals(set2));
    }
}
```

2. **Union and Intersection of Two Sets**
   Given two sets, compute their union and intersection.
   Example:
   Set1: {1, 2, 3}, Set2: {3, 4, 5} → Union: {1, 2, 3, 4, 5}, Intersection: {3}.

```java
import java.util.*;

public class UnionIntersection {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Set<Integer> set1 = new HashSet<>();
        Set<Integer> set2 = new HashSet<>();

        System.out.println("Enter number of elements in Set1:");
        int n1 = sc.nextInt();
        System.out.println("Enter elements of Set1:");
        for (int i = 0; i < n1; i++) {
            set1.add(sc.nextInt());
        }

        System.out.println("Enter number of elements in Set2:");
        int n2 = sc.nextInt();
        System.out.println("Enter elements of Set2:");
        for (int i = 0; i < n2; i++) {
            set2.add(sc.nextInt());
        }

        Set<Integer> union = new HashSet<>(set1);
        union.addAll(set2);

        Set<Integer> intersection = new HashSet<>(set1);
        intersection.retainAll(set2);

        System.out.println("Union: " + union);
        System.out.println("Intersection: " + intersection);
    }
}
```

3. **Symmetric Difference**
   Find the symmetric difference (elements present in either set but not in both) of two sets.
   Example:
   Set1: {1, 2, 3}, Set2: {3, 4, 5} → Output: {1, 2, 4, 5}.

```
import java.util.*;

public class SymmetricDifference {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Set<Integer> set1 = new HashSet<>();
        Set<Integer> set2 = new HashSet<>();

        System.out.println("Enter elements for Set1:");
        int n1 = sc.nextInt();
        for (int i = 0; i < n1; i++) {
            set1.add(sc.nextInt());
        }

        System.out.println("Enter elements for Set2:");
        int n2 = sc.nextInt();
        for (int i = 0; i < n2; i++) {
            set2.add(sc.nextInt());
        }

        Set<Integer> symDiff = new HashSet<>(set1);
        symDiff.addAll(set2);
        Set<Integer> temp = new HashSet<>(set1);
        temp.retainAll(set2);
        symDiff.removeAll(temp);

        System.out.println(symDiff);
    }
}
```

4. **Convert a Set to a Sorted List**
   Convert a HashSet of integers into a sorted list in ascending order.
   Example:
   Input: {5, 3, 9, 1} → Output: [1, 3, 5, 9].

```
import java.util.*;

public class SetToSortedList {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Set<Integer> set = new HashSet<>();
        System.out.println("Enter number of elements:");
        int n = sc.nextInt();
        System.out.println("Enter elements:");
        for (int i = 0; i < n; i++) {
            set.add(sc.nextInt());
        }

        List<Integer> sortedList = new ArrayList<>(set);
        Collections.sort(sortedList);
        System.out.println(sortedList);
    }
}
```

5. **Find Subsets**
   Check if one set is a subset of another.
   Example:
   Set1: {2, 3}, Set2: {1, 2, 3, 4} → Output: true.

```
import java.util.*;

public class SubsetCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Set<Integer> set1 = new HashSet<>();
        Set<Integer> set2 = new HashSet<>();

        System.out.println("Enter elements of Set1:");
        int n1 = sc.nextInt();
        for (int i = 0; i < n1; i++) {
            set1.add(sc.nextInt());
        }

        System.out.println("Enter elements of Set2:");
        int n2 = sc.nextInt();
        for (int i = 0; i < n2; i++) {
            set2.add(sc.nextInt());
        }

        System.out.println(set2.containsAll(set1));
    }
}
```

Insurance Policy Management System

Each policy has the following attributes:

- Policy Number (unique identifier)
- Policyholder Name
- Expiry Date
- Coverage Type (e.g., Health, Auto, Home)
- Premium Amount

Requirements:

1. Store Unique Policies: Implement methods to store policies using different types of sets (HashSet, LinkedHashSet, TreeSet), each serving different purposes:
    - HashSet for quick lookups.
    - LinkedHashSet to maintain the order of insertion.
    - TreeSet to maintain policies sorted by expiry date.

2. Retrieve Policies: Implement methods to retrieve and display policies based on certain criteria:
    - All unique policies.
    - Policies expiring soon (within the next 30 days
    - Policies with a specific coverage type.
    - Duplicate policies based on policy numbers.

3. Performance Comparison: Compare the performance of HashSet, LinkedHashSet, and TreeSet in terms of adding, removing, and searching for policies.

Queue Interface

1. **Reverse a Queue**
   Reverse the elements of a queue using only queue operations (e.g., add, remove, isEmpty).
   Example:
   Input: [10, 20, 30] → Output: [30, 20, 10].

```
import java.util.*;

public class ReverseQueue {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Queue<Integer> queue = new LinkedList<>();
        System.out.print("Enter number of elements: ");
        int n = sc.nextInt();
        System.out.println("Enter elements:");
        for (int i = 0; i < n; i++) {
            queue.add(sc.nextInt());
        }

        Stack<Integer> stack = new Stack<>();
        while (!queue.isEmpty()) {
            stack.push(queue.remove());
        }
        while (!stack.isEmpty()) {
            queue.add(stack.pop());
        }
        System.out.println("Reversed Queue: " + queue);
    }
}
```

2. **Generate Binary Numbers Using a Queue**
   Generate the first N binary numbers (as strings) using a queue.
   Example:
   N=5 → Output: ["1", "10", "11", "100", "101"].

```
import java.util.*;

public class BinaryNumbers {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter N: ");
        int n = sc.nextInt();
        Queue<String> queue = new LinkedList<>();
        queue.add("1");

        System.out.println("Binary Numbers:");
        for (int i = 0; i < n; i++) {
            String current = queue.remove();
            System.out.print(current + " ");
            queue.add(current + "0");
            queue.add(current + "1");
        }
    }
}
```

3. **Hospital Triage System**
   Simulate a hospital triage system using a PriorityQueue where patients with higher severity are treated first.
   Example:
   Patients: [("John", 3), ("Alice", 5), ("Bob", 2)] → Order: Alice, John, Bob.

```
import java.util.*;
class Patient {
    String name;
    int severity;

    Patient(String name, int severity) {
        this.name = name;
        this.severity = severity;
    }
}

public class HospitalTriage {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        PriorityQueue<Patient> pq = new PriorityQueue<>(new
Comparator<Patient>() {
            public int compare(Patient p1, Patient p2) {
                return p2.severity - p1.severity;
            }
        });

        System.out.print("Enter number of patients: ");
        int n = sc.nextInt();
        for (int i = 0; i < n; i++) {
            System.out.print("Enter name: ");
            String name = sc.next();
            System.out.print("Enter severity: ");
            int severity = sc.nextInt();
            pq.add(new Patient(name, severity));
        }
        System.out.println("Treatment Order:");
        while (!pq.isEmpty()) {
            Patient p = pq.remove();
            System.out.println(p.name + " (Severity: " +
p.severity + ")");
        }
    }
}
```

4. **Implement a Stack Using Queues**
   Implement a stack data structure using two queues and support push, pop, and top operations.
   Example:
   Push 1, 2, 3 → Pop → Output: 3.

```
import java.util.*;

public class StackUsingQueues {
    static Queue<Integer> q1 = new LinkedList<>();
    static Queue<Integer> q2 = new LinkedList<>();

    public static void push(int x) {
        q2.add(x);
        while (!q1.isEmpty()) {
            q2.add(q1.remove());
        }
        Queue<Integer> temp = q1;
        q1 = q2;
        q2 = temp;
    }

    public static int pop() {
        if (q1.isEmpty()) return -1;
        return q1.remove();
    }

    public static int top() {
        if (q1.isEmpty()) return -1;
        return q1.peek();
    }

    public static void main(String[] args) {
        push(1);
        push(2);
        push(3);
        System.out.println("Top: " + top());
        System.out.println("Pop: " + pop());
        System.out.println("Top after pop: " + top());
    }
}
```

5. **Circular Buffer Simulation**
   Implement a circular buffer (fixed-size queue) using an array-based queue. When full, overwrite the oldest element.
   Example:
   Buffer size=3: Insert 1, 2, 3 → Insert 4 → Buffer: [2, 3, 4].

```
import java.util.*;

public class CircularBuffer {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Buffer size: ");
        int size = sc.nextInt();
        int[] buffer = new int[size];
        int index = 0;

        System.out.print("Enter number of elements to insert:
");
        int n = sc.nextInt();
        for (int i = 0; i < n; i++) {
            int val = sc.nextInt();
            buffer[index] = val;
            index = (index + 1) % size;
        }

        System.out.print("Buffer: ");
        for (int i = 0; i < size; i++) {
            System.out.print(buffer[(index + i) % size] + " ");
        }
    }
}
```

Map Interface

1. **Word Frequency Counter**
   Read a text file and count the frequency of each word using a HashMap. Ignore case and punctuation.
   Example:
   Input: "Hello world, hello Java!" → Output: {hello=2, world=1, java=1}

```
import java.util.*;
import java.io.*;

public class WordFrequency {
    public static void main(String[] args) throws IOException {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter file path: ");
        String path = sc.nextLine();

        BufferedReader br = new BufferedReader(new
FileReader(path));
        HashMap<String, Integer> map = new HashMap<>();
        String line;

        while ((line = br.readLine()) != null) {
            line = line.toLowerCase().replaceAll("[^a-z0-9\\s]",
"");
            String[] words = line.split("\\s+");
            for (int i = 0; i < words.length; i++) {
                if (!words[i].isEmpty()) {
                    map.put(words[i], map.getOrDefault(words[i],
0) + 1);
                }
            }
        }
        br.close();
        System.out.println("Word Frequency: " + map);
    }
}
```

2. **Invert a Map**
   Invert a Map<K, V> to produce a Map<V, K>. Handle duplicate values by storing them in a list.

Example:
Input: {A=1, B=2, C=1} → Output: {1=[A, C], 2=[B]}.

```
import java.util.*;

public class InvertMap {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        HashMap<String, Integer> map = new HashMap<>();
        System.out.print("Enter number of entries: ");
        int n = sc.nextInt();

        for (int i = 0; i < n; i++) {
            System.out.print("Enter key: ");
            String key = sc.next();
            System.out.print("Enter value: ");
            int value = sc.nextInt();
            map.put(key, value);
        }

        HashMap<Integer, List<String>> inverted = new
HashMap<>();
        for (Map.Entry<String, Integer> entry : map.entrySet())
{
            int val = entry.getValue();
            if (!inverted.containsKey(val)) {
                inverted.put(val, new ArrayList<>());
            }
            inverted.get(val).add(entry.getKey());
        }

        System.out.println("Inverted Map: " + inverted);
    }
}
```

3. **Find the Key with the Highest Value**
   Given a Map<String, Integer>, find the key with the maximum value.
   Example:
   Input: {A=10, B=20, C=15} → Output: B.

```
import java.util.*;

public class MaxValueKey {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        HashMap<String, Integer> map = new HashMap<>();
        System.out.print("Enter number of entries: ");
        int n = sc.nextInt();

        for (int i = 0; i < n; i++) {
            System.out.print("Enter key: ");
            String key = sc.next();
            System.out.print("Enter value: ");
            int value = sc.nextInt();
            map.put(key, value);
        }

        String maxKey = null;
        int maxVal = Integer.MIN_VALUE;

        for (Map.Entry<String, Integer> entry : map.entrySet())
{
            if (entry.getValue() > maxVal) {
                maxVal = entry.getValue();
                maxKey = entry.getKey();
            }
        }
```

```java
        System.out.println("Key with highest value: " + maxKey);
    }
}
```

4. **Merge Two Maps**
   Merge two maps such that if a key exists in both, sum their values.
   Example:
   Map1: {A=1, B=2}, Map2: {B=3, C=4} → Output: {A=1, B=5, C=4}.

```java
import java.util.*;

public class MergeMaps {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        HashMap<String, Integer> map1 = new HashMap<>();
        HashMap<String, Integer> map2 = new HashMap<>();

        System.out.print("Enter entries for Map1: ");
        int n1 = sc.nextInt();
        for (int i = 0; i < n1; i++) {
            System.out.print("Enter key: ");
            String key = sc.next();
            System.out.print("Enter value: ");
            int value = sc.nextInt();
            map1.put(key, value);
        }

        System.out.print("Enter entries for Map2: ");
        int n2 = sc.nextInt();
        for (int i = 0; i < n2; i++) {
            System.out.print("Enter key: ");
            String key = sc.next();
            System.out.print("Enter value: ");
            int value = sc.nextInt();
            map2.put(key, value);
        }

        for (Map.Entry<String, Integer> entry : map2.entrySet())
{
            String key = entry.getKey();
            int value = entry.getValue();
            map1.put(key, map1.getOrDefault(key, 0) + value);
        }

        System.out.println("Merged Map: " + map1);
    }
}
```

5. **Group Objects by Property**
   Given a list of Employee objects, group them by their department using a Map<Department, List<Employee>>.
   Example:
   Employees: [Alice (HR), Bob (IT), Carol (HR)] → Output: HR: [Alice, Carol], IT: [Bob].

```
import java.util.*;

class Employee {
    String name;
    String department;

    Employee(String name, String department) {
        this.name = name;
        this.department = department;
    }
}
```

```
public class GroupByDepartment {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        HashMap<String, List<String>> map = new HashMap<>();

        System.out.print("Enter number of employees: ");
        int n = sc.nextInt();

        for (int i = 0; i < n; i++) {
            System.out.print("Enter employee name: ");
            String name = sc.next();
            System.out.print("Enter department: ");
            String dept = sc.next();

            if (!map.containsKey(dept)) {
                map.put(dept, new ArrayList<>());
            }
            map.get(dept).add(name);
        }

        System.out.println("Grouped by Department:");
        for (Map.Entry<String, List<String>> entry :
map.entrySet()) {
            System.out.print(entry.getKey() + ": ");
            List<String> list = entry.getValue();
            for (int j = 0; j < list.size(); j++) {
                System.out.print(list.get(j));
                if (j != list.size() - 1) System.out.print(", 
");
            }
            System.out.println();
        }
    }
}
```

Insurance Policy Management System

Build a system for managing insurance policies where you have to:

- Store and manage policies with unique identifiers.
- Retrieve and manipulate policies based on different criteria.
- Track policies by various attributes such as policyholder name and expiry date.

Requirements:

1. Store Policies in a Map:
    - Use HashMap to store policies with policy numbers as keys and policy
    - details as values.
    - Use LinkedHashMap to maintain the insertion order of policies.
    - Use TreeMap to store policies sorted by expiry date.

2. Retrieve and Manipulate Policies:
    1) Implement methods to:
        - Retrieve a policy by its number.
        - List all policies expiring within the next 30 days.
        - List all policies for a specific policyholder.
        - Remove policies that are expired.

Design a Voting System

Description: Design a system where:

- Votes are stored in a HashMap (Candidate -> Votes).
- TreeMap is used to display the results in sorted order.
- LinkedHashMap is used to maintain the order of votes.

Implement a Shopping Cart

Description:

- Use HashMap to store product prices.
- Use LinkedHashMap to maintain the order of items added.
- Use TreeMap to display items sorted by price.

Implement a Banking System

Description:

- HashMap stores customer accounts (AccountNumber -> Balance).
- TreeMap sorts customers by balance.
- Queue processes withdrawal requests.