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 reverse(List<Integer> lst) {
        int len = lst.size();

        for (int i = 0; i <= len / 2; i++) {
            int temp = lst.get(i);
            lst.set(i, lst.get(len - 1 - i));
            lst.set(len - 1 - i, temp);
        }
    }

    public static void printList(List<Integer> lst) {
        for (int i = 0; i < lst.size(); i++) {
            System.out.print(lst.get(i) + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        List<Integer> lst = new ArrayList<>();
        lst.add(1);
        lst.add(2);
        lst.add(3);
        lst.add(4);
        lst.add(5);

        printList(lst);

        reverse(lst);
```

```
        printList(lst);
    }

}
```

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}.

```
import java.util.*;

public class Frequency {

    public static void displayFreq(HashMap<String, Integer> map) {
        for (String str : map.keySet()) {
            System.out.println(str + " -> " + map.get(str));
        }
    }

    public static void main(String[] args) {
        HashMap<String, Integer> map = new HashMap<>();

        List<String> lst = new ArrayList<>();
        lst.add("apple");
        lst.add("banana");
        lst.add("apple");
        lst.add("orange");

        for (int i = 0; i < lst.size(); i++) {
            String str = lst.get(i);
            if (map.containsKey(str)) {
                map.put(str, map.get(str) + 1);
            } else {
                map.put(str, 1);
            }
        }
```

```
        displayFreq(map);
    }

}
```

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].

```
import java.util.*;

public class RotateList {

    public static void rotate(List<Integer> lst, int n) {
        n = n % lst.size();

        for (int i = 0; i < n; i++) {
            lst.add(lst.get(i));
        }

        for (int i = 0; i < n; i++) {
            lst.remove(0);
        }
    }

    public static void main(String[] args) {
        List<Integer> lst = new ArrayList<>();
        lst.add(10);
        lst.add(20);
        lst.add(30);
        lst.add(40);
        lst.add(50);

        System.out.println(lst);

        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        sc.close();
```

```
        rotate(lst, n);

        System.out.println(lst);

    }

}
```

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].

```
import java.util.*;

public class RemoveDuplicates {

    public static void removeDuplicate(List<Integer> lst) {
        HashSet<Integer> set = new HashSet<>();
        for (int i = 0; i < lst.size(); i++) {
            if (!set.contains(lst.get(i))) {
                set.add(lst.get(i));
            } else {
                lst.remove(i);
                i--;
            }
        }
    }

    public static void main(String[] args) {
        List<Integer> lst = new ArrayList<>();
        lst.add(3);
        lst.add(1);
        lst.add(2);
        lst.add(2);
        lst.add(3);
        lst.add(4);
```

```
        System.out.println("Initial List with duplicates");
        System.out.println(lst);

        removeDuplicate(lst);

        System.out.println("List after removing duplicates");
        System.out.println(lst);
    }
}
```

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.

```
public class NthFromEnd extends Object {


    static class Node {
        char data;
        Node next;

        Node(char data) {
            this.data = data;
            this.next = null;
        }
    }

    public static Node nthFromEnd(Node head, int n) {
        Node fast = head;
        for (int i = 0; i < n && fast != null; i++) {
            fast = fast.next;
        }
        Node slow = head;
```

```
        while (fast != null) {
            fast = fast.next;
            slow = slow.next;
        }

        return slow;
    }

    public static void main(String[] args) {
        Node head = new Node('A');
        head.next = new Node('B');
        head.next.next = new Node('C');
        head.next.next.next = new Node('D');
        head.next.next.next.next = new Node('E');

        Node temp = head;
        System.out.print("List: ");
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();

        System.out.println(nthFromEnd(head, 2).data);

    }
}
```

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 boolean isEqual(Set<Integer> set1, Set<Integer> set2) {
        if (set1.size() != set2.size()) {
            return false;
        }
        for (int i : set1) {
            if (!set2.contains(i)) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter Number of elements you want to add in set1: ");
        int n1 = sc.nextInt();
        Set<Integer> set1 = new HashSet<>();

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


        System.out.print("Enter Number of elements you want to add in set2: ");
        int n2 = sc.nextInt();
        Set<Integer> set2 = new HashSet<>();

        System.out.println("Enter elements:");
```

```
        for (int i = 0; i < n2; i++) {
            set2.add(sc.nextInt());
        }


        if (isEqual(set1, set2)) {
            System.out.println("Set are equal!");
        } else {
            System.out.println("Sets are not equal!");
        }

        sc.close();
    }

}
```

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}.

```
import java.util.*;

public class UnionIntersection {

    public static Set<Integer> getUnion(Set<Integer> set1,
Set<Integer> set2) {
        Set<Integer> union = new HashSet<>();
        for (int i : set1) {
            union.add(i);
        }
        for (int i : set2) {
            union.add(i);
        }

        return union;
```

```
    }

    public static Set<Integer> getIntersection(Set<Integer>
set1, Set<Integer> set2) {
        Set<Integer> intersection = new HashSet<>();

        for (int i : set2) {
            if (set1.contains(i)) {
                intersection.add(i);
            }
        }

        return intersection;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter Number of elements you want to
add in set1: ");
        int n1 = sc.nextInt();
        Set<Integer> set1 = new HashSet<>();

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

        System.out.print("Enter Number of elements you want to
add in set2: ");
        int n2 = sc.nextInt();
        Set<Integer> set2 = new HashSet<>();

        System.out.println("Enter elements:");
        for (int i = 0; i < n2; i++) {
```

```
            set2.add(sc.nextInt());
        }

        Set<Integer> union = getUnion(set1, set2);
        System.out.println("Union: " + union);

        Set<Integer> intersection = getIntersection(set1, set2);
        System.out.println("Intersection: " + intersection);

        sc.close();
    }

}
```

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 Set<Integer> getSymmetricDifference(Set<Integer> set1,
Set<Integer> set2) {
        Set<Integer> diff = new HashSet<>();
        for (int i : set1) {
            if (!set2.contains(i)) {
                diff.add(i);
            }
        }

        for (int i : set2) {
            if (!set1.contains(i)) {
                diff.add(i);
```

```
            }
        }

        return diff;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter Number of elements you want to add in set1: ");
        int n1 = sc.nextInt();
        Set<Integer> set1 = new HashSet<>();

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

        System.out.print("Enter Number of elements you want to add in set2: ");
        int n2 = sc.nextInt();
        Set<Integer> set2 = new HashSet<>();

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

        System.out.println("Symmetric Difference: " +
getSymmetricDifference(set1, set2));

        sc.close();
    }
}
```

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 SetToList {

    public static List<Integer> setToList(Set<Integer> set) {
        List<Integer> lst = new ArrayList<>();
        for (int i : set) {
            lst.add(i);
        }

        Collections.sort(lst);

        return lst;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter Number of elements you want to add in set: ");
        int n1 = sc.nextInt();
        Set<Integer> set = new HashSet<>();

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

        System.out.print("Set to Sorted List: ");
        List<Integer> lst = setToList(set);
        System.out.println(lst);

        sc.close();
    }
}
```

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 Subset {

    public static boolean isSubset(Set<Integer> set1, Set<Integer> set2) {
        boolean isSubset = true;
        for (int i : set1) {
            if (!set2.contains(i)) {
                isSubset = false;
                break;
            }
        }
        if (!isSubset) {
            for (int i : set2) {
                if (!set1.contains(i)) {
                    isSubset = false;
                    break;
                }
            }
        }
        return isSubset;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter Number of elements you want to add in set1: ");
        int n1 = sc.nextInt();
        Set<Integer> set1 = new HashSet<>();

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

        System.out.print("Enter Number of elements you want to add in set2: ");
        int n2 = sc.nextInt();
        Set<Integer> set2 = new HashSet<>();
```

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

        System.out.println("Is Subset: " + isSubset(set1, set2));

        sc.close();
    }
}
```

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.

```
import java.util.*;

class PolicyManagement {

    HashSet<Policy> hs;
    LinkedHashSet<Policy> ls;
    TreeSet<Policy> ts;

    PolicyManagement() {

        this.hs = new HashSet<>();
        this.ls = new LinkedHashSet<>();
        this.ts = new TreeSet<>(new Comparator<Policy>() {
            // overiding the compare method of Treeset
            @Override
            public int compare(Policy p1, Policy p2) {
                return p1.expiryDate - p2.expiryDate;
            }
        });

    }

    public void add(Policy p) {

        for (Policy po : hs) {
            if (po.policyNumber == p.policyNumber &&
!po.policyHolderName.equals(p.policyHolderName)) {
                System.out.println("Cannot add another user with used policy
number");
                return;
            }
```

```
        }

        hs.add(p);
        ls.add(p);

        ts.add(p);

    }

    public void retrieveAllUnique() {

        Iterator<Policy> it = ls.iterator();

        while (it.hasNext()) {
            Policy p = it.next();
            p.display();
        }
    }

    public void retrieveExpiring() {

        Iterator<Policy> it = ts.iterator();
        while (it.hasNext()) {
            Policy p = it.next();
            if (p.expiryDate <= 30) {
                p.display();
            } else
                break;
        }

    }

    public void retrieveCoverage(String coverageType) {

        Iterator<Policy> it = ls.iterator();
        while (it.hasNext()) {
            Policy p = it.next();
            if (p.coverageType.equals(coverageType)) {
                p.display();
            }
        }

    }
```

```
    public void retrieveDuplicates(int policyNumber) {

        Iterator<Policy> it = ls.iterator();
        while (it.hasNext()) {
            Policy p = it.next();
            if (p.policyNumber == policyNumber) {
                p.display();
            }
        }

    }

    public void remove(Policy p) {

        hs.remove(p);
        ts.remove(p);
        ls.remove(p);

    }

}

class Policy {

    int policyNumber;
    String policyHolderName;
    int expiryDate;
    String coverageType;
    int premiumAmount;

    Policy(int policyNumber, String policyHolderName, int expiryDate, String
coverageType, int premiumAmount) {

        this.policyNumber = policyNumber;
        this.policyHolderName = policyHolderName;
        this.expiryDate = expiryDate;
        this.coverageType = coverageType;
        this.premiumAmount = premiumAmount;

    }

    public void display() {
```

```
        System.out.println("Policy number is " + this.policyNumber);
        System.out.println("Policy Holder name is " + this.policyHolderName);
        System.out.println("Policy expiry date is " + this.expiryDate);
        System.out.println("Policy coverage type is " + this.coverageType);
        System.out.println("Policy premium amount is " + this.premiumAmount);
        System.out.println();

    }

}

public class  InsurancePolicyManagementSystem{

    public static void main(String[] args) {

        Policy p1 = new Policy(1, "Aman", 25, "Health", 5);
        Policy p2 = new Policy(2, "Kushagra", 40, "Auto", 3);
        Policy p3 = new Policy(1, "Aman", 25, "Auto", 5);
        Policy p4 = new Policy(1, "Kushagra", 25, "Health", 5);
        Policy p5 = new Policy(3, "Aman", 25, "Health", 5);

        PolicyManagement i1 = new PolicyManagement();

        i1.add(p1);
        i1.add(p2);
        i1.add(p3);
        i1.add(p4);
        i1.add(p5);

        // i1.retrieveAllUnique();

        // i1.retrieveExpiring();

        // i1.retrieveCoverage("Health");

        i1.retrieveDuplicates(1);
    }

}
```

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 reverse(Queue<Integer> q) {
        if (q.isEmpty()) {
            return;
        }
        int ele = q.remove();
        reverse(q);
        q.add(ele);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of elements you want to add in queue:
");
        int n = sc.nextInt();
        Queue<Integer> q = new LinkedList<>();

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

        System.out.println("Initial queue: " + q);

        reverse(q);

        System.out.println("Reversed queue: " + q);

        sc.close();}}
```

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 GenerateBinaryNumbers {


    public static Queue<String> generateNumbers(int n) {
        Queue<String> q = new LinkedList<>();
        Queue<String> ans = new LinkedList<>();
        q.add("1");
        if (n == 1) {
            return q;
        }

        for(int i=0;i<n;i++) {
            String num = q.poll();

            ans.add(num);

            q.add(num + "0");
            q.add(num + "1");

        }

        return ans;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of terms you want in binary format
starting from 1: ");
        int n = sc.nextInt();

        Queue<String> q = generateNumbers(n);

        System.out.println("Generated Queue: " + q);
        sc.close();}}
```

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 HospitalTriageSystem {

    static class Patient {

        String name;
        int severity;

        Patient(String name, int severity) {

            this.name = name;
            this.severity = severity;

        }

        public void display() {
            System.out.println("Patient name is : " + this.name);
            System.out.println("Patient severity is : " + this.severity);

        }

    }

    PriorityQueue<Patient> pq;

    HospitalTriageSystem() {

        pq = new PriorityQueue<>(new Comparator<Patient>() {
            public int compare(Patient p1, Patient p2) {
                return p1.severity - p2.severity;
            }
        });

    }

    public void add(Patient p) {
```

```
        pq.add(p);

    }

    public void simulate() {

        System.out.println("Order in which the patients will be treated is :
");

        Stack<Patient> st = new Stack<>();

        while (!pq.isEmpty()) {

            Patient p = pq.poll();
            System.out.println(p.name);
            st.push(p);
        }
        while (!st.isEmpty()) {

            Patient p = st.pop();
            pq.add(p);
        }

    }

    public static void main(String[] args) {

        Patient p1 = new Patient("ABC", 1);
        Patient p2 = new Patient("DEF", 2);
        Patient p3 = new Patient("GHI", 3);

        HospitalTriageSystem hs = new HospitalTriageSystem();

        hs.add(p1);
        hs.add(p2);
        hs.add(p3);

        hs.simulate();

    }
}
```

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 StackUsingQueue {

    static class Stack {

        Queue<Integer> q1;
        Queue<Integer> q2;

        Stack() {

            this.q1 = new LinkedList<>();
            this.q2 = new LinkedList<>();

        }

        public void push(int ele) {

            while (!q1.isEmpty()) {
                q2.add(q1.poll());
            }

            q1.add(ele);

            while (!q2.isEmpty()) {
                q1.add(q2.poll());
            }

        }

        public int pop() {

            return q1.poll();

        }
```

```
        public int top() {
            return q1.peek();

        }

        public void display() {

            while (!q1.isEmpty()) {

                int fr = q1.poll();
                q2.add(fr);
                System.out.print(fr + " ");

            }
            System.out.println();

            while (!q2.isEmpty()) {
                q1.add(q2.poll());
            }
        }

        public static void main(String[] args) {

            Stack st = new Stack();

            st.push(1);
            st.push(2);
            st.push(3);
            st.push(4);
            st.push(5);

            System.out.println(st.top());

            st.display();

            System.out.println(st.pop());

            st.display();

        }
    }}
```

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].

```
class CircularBufferSimulation {

    int[] arr;
    int in;
    int size;

    CircularBufferSimulation(int size) {

        arr = new int[size];
        in = 0;
        size = 0;

    }

    public void add(int ele) {

        arr[in] = ele;
        in = (in + 1) % arr.length;

        if (size < arr.length)
            size++;

    }

    public int peek() {

        return arr[0];

    }

    public int poll() {

        if (size == 0) {
            System.out.println("Cannot get element as queue is empty");
            return -1;
        }
```

```
        int ele = arr[0];

        for (int i = 1; i < size; i++)
            arr[i - 1] = arr[i];

        size--;
        return ele;

    }

    public void display() {

        for (int i = 0; i < size; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {

        CircularBufferSimulation cb = new CircularBufferSimulation(5);

        cb.add(1);
        cb.add(2);
        cb.add(3);
        cb.add(4);
        cb.add(5);
        cb.add(6);

        cb.display();

        System.out.println(cb.peek());

        System.out.println(cb.poll());

        cb.display();

    }

}
```

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

public class WordFrequencyCounter {
    public static void main(String[] args) throws IOException {
        BufferedReader reader = new BufferedReader(new InputStreamReader(new
FileInputStream("input.txt")));
        HashMap<String, Integer> hashMap = new HashMap<>();
        String line;
        while ((line = reader.readLine()) != null) {
            line = line.toLowerCase();
            StringBuilder cleanWord = new StringBuilder();
            for (int i = 0; i < line.length(); i++) {
                char c = line.charAt(i);
                if ((c >= 'a' && c <= 'z') || c == ' ')
                    cleanWord.append(c);
            }
            String[] words = cleanWord.toString().split(" ");
            for (String word : words) {
                if (word.length() > 0) {
                    if (hashMap.containsKey(word))
                        hashMap.put(word, hashMap.get(word) + 1);
                    else
                        hashMap.put(word, 1);
                }
            }
        }
        reader.close();
        System.out.println(hashMap);
    }
}
```

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) {
        HashMap<String, Integer> hashMap = new HashMap<>();
        hashMap.put("A", 1);
        hashMap.put("B", 2);
        hashMap.put("C", 1);
        HashMap<Integer, ArrayList<String>> inverted = new HashMap<>();
        for (String key : hashMap.keySet()) {
            int value = hashMap.get(key);
            if (!inverted.containsKey(value))
                inverted.put(value, new ArrayList<>());
            inverted.get(value).add(key);
        }
        System.out.println(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 FindMaxKey {
    public static void main(String[] args) {
        HashMap<String, Integer> hashMap = new HashMap<>();
        hashMap.put("A", 10);
        hashMap.put("B", 20);
        hashMap.put("C", 15);
        String maxKey = "";
        int maxValue = Integer.MIN_VALUE;
        for (String key : hashMap.keySet()) {
            int value = hashMap.get(key);
            if (value > maxValue) {
                maxValue = value;
                maxKey = key;
            }
        }
        System.out.println(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}.

```
import java.util.*;

public class MergeTwoMaps {
    public static void main(String[] args) {
        HashMap<String, Integer> map1 = new HashMap<>();
        map1.put("A", 1);
        map1.put("B", 2);
        HashMap<String, Integer> map2 = new HashMap<>();
        map2.put("B", 3);
        map2.put("C", 4);
        HashMap<String, Integer> merged = new HashMap<>();
        for (String key : map1.keySet())
            merged.put(key, map1.get(key));
        for (String key : map2.keySet()) {
            if (merged.containsKey(key))
                merged.put(key, merged.get(key) +
map2.get(key));
            else
                merged.put(key, map2.get(key));
        }
        System.out.println(merged);
    }
}
```

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, department;

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

    public String toString() {
        return name;
    }
}

public class GroupObjects {
    public static void main(String[] args) {
        ArrayList<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Alice", "HR"));
        employees.add(new Employee("Bob", "IT"));
        employees.add(new Employee("Carol", "HR"));
        HashMap<String, ArrayList<Employee>> grouped = new HashMap<>();
        for (Employee e : employees) {
            if (!grouped.containsKey(e.department))
                grouped.put(e.department, new ArrayList<>());
            grouped.get(e.department).add(e);
        }
        System.out.println(grouped);
    }
}
```

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.

```
import java.util.*;

class Policy {
    String policyholder;
    int expiryDays;

    Policy(String policyholder, int expiryDays) {
        this.policyholder = policyholder;
        this.expiryDays = expiryDays;
    }

    public String toString() {
        return policyholder + " " + expiryDays;
    }
}

public class InsurancePolicySystemHashMap {
    HashMap<String, Policy> hashMap = new HashMap<>();
```

```
    public void addPolicy(String policyNumber, Policy policy) {
        hashMap.put(policyNumber, policy);
    }

    public Policy getPolicy(String policyNumber) {
        return hashMap.get(policyNumber);
    }

    public ArrayList<Policy> getExpiringSoon() {
        ArrayList<Policy> result = new ArrayList<>();
        for (String key : hashMap.keySet()) {
            Policy policy = hashMap.get(key);
            if (policy.expiryDays <= 30)
                result.add(policy);
        }
        return result;
    }

    public ArrayList<Policy> getPoliciesByHolder(String name) {
        ArrayList<Policy> result = new ArrayList<>();
        for (String key : hashMap.keySet()) {
            if (hashMap.get(key).policyholder.equals(name))
                result.add(hashMap.get(key));
        }
        return result;
    }

    public void removeExpired() {
        ArrayList<String> toRemove = new ArrayList<>();
        for (String key : hashMap.keySet()) {
            if (hashMap.get(key).expiryDays <= 0)
                toRemove.add(key);
        }
        for (String key : toRemove)
            hashMap.remove(key);
    }
}
```

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.

```
import java.util.*;

public class VotingSystem {
    HashMap<String, Integer> votes = new HashMap<>();

    public void vote(String candidate) {
        if (votes.containsKey(candidate))
            votes.put(candidate, votes.get(candidate) + 1);
        else
            votes.put(candidate, 1);
    }

    public void displayResults() {
        ArrayList<String> sortedCandidates = new ArrayList<>(votes.keySet());
        Collections.sort(sortedCandidates);
        for (String candidate : sortedCandidates) {
            System.out.println(candidate + ": " + votes.get(candidate));
        }
    }
}
```

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.

```
import java.util.*;

public class ShoppingCart {
    HashMap<String, Integer> prices = new HashMap<>();
    ArrayList<String> items = new ArrayList<>();

    public void addItem(String item, int price) {
        prices.put(item, price);
        items.add(item);
    }

    public void displaySortedByPrice() {
        ArrayList<String> sortedItems = new ArrayList<>(items);
        Collections.sort(sortedItems, (a, b) -> prices.get(a) - prices.get(b));
        for (String item : sortedItems) {
            System.out.println(item + ": " + prices.get(item));
        }
    }
}
```

Implement a Banking System

Description:

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

```
import java.util.*;

public class BankingSystem {
    HashMap<String, Integer> accounts = new HashMap<>();
    Queue<String> withdrawals = new LinkedList<>();

    public void createAccount(String acc, int balance) {
        accounts.put(acc, balance);
    }

    public void withdraw(String acc) {
        if (accounts.containsKey(acc))
            withdrawals.add(acc);
    }
}
```