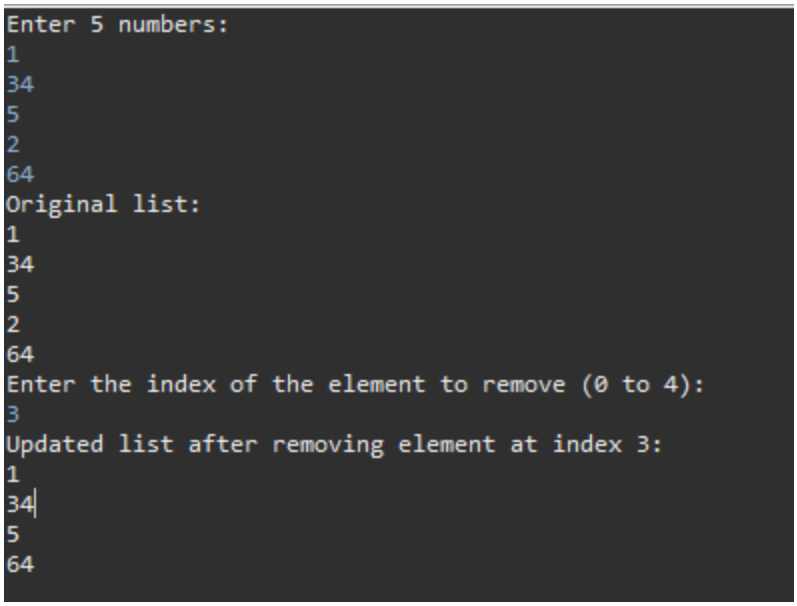
1. ArrayList Questions

1. **Basic List Operations**
    - Create an ArrayList of integers.
    - Add five numbers to it.
    - Print all elements using a loop.
    - Remove an element at index 2 and print the updated list

```
import java.util.ArrayList;
import java.util.Scanner;

public class ArrayListExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ArrayList<Integer> numbers = new ArrayList<>();

        System.out.println("Enter 5 numbers:");
        for (int i = 0; i < 5; i++) {
            numbers.add(scanner.nextInt());
        }

        System.out.println("Original list:");
        for (int number : numbers) {
            System.out.println(number);
        }

        System.out.println("Enter the index of the element to remove (0 to 4):");
        int index = scanner.nextInt();
        numbers.remove(index);

        System.out.println("Updated list after removing element at index " + index + ":");
        for (int number : numbers) {
            System.out.println(number);
        }
    }
}
```

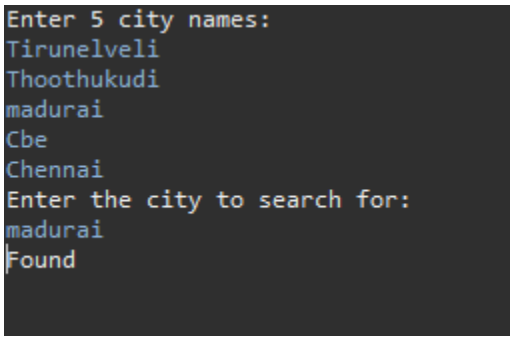


2.Check if an Element Exists in a List

- Create an ArrayList of strings.
- Add five city names.
- Check if "New York" exists in the list.
- Print "Found" if it exists; otherwise, print "Not Found".

```
import java.util.ArrayList;

import java.util.Scanner;

public class CitySearch {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        ArrayList<String> cities = new ArrayList<>();

        System.out.println("Enter 5 city names:");

        for (int i = 0; i < 5; i++) {

            cities.add(scanner.nextLine());

        }
```

```
        System.out.println("Enter the city to search for:");
        String cityToSearch = scanner.nextLine();
        if (cities.contains(cityToSearch)) {
            System.out.println("Found");
        } else {
            System.out.println("Not Found");
        }
    }
}
```



3.Iterating Over a List

- Create an ArrayList of Double values.
- Add five decimal numbers.
- Use a **for-each loop** and an **Iterator** to print all elements.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;
public class ListIteration {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
        ArrayList<Double> numbers = new ArrayList<>();
        System.out.println("Enter 5 decimal numbers:");
        for (int i = 0; i < 5; i++) {
            numbers.add(scanner.nextDouble());
        }
        System.out.println("Using for-each loop:");
        for (Double number : numbers) {
            System.out.println(number);
        }
        System.out.println("\nUsing Iterator:");
        Iterator<Double> iterator = numbers.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
    }
   }
}
```

```
Enter 5 decimal numbers:
5.6
23.8
12.5
64.9
3.2
Using for-each loop:
5.6
23.8
12.5
64.9
3.2

Using Iterator:
5.6
23.8
12.5
64.9
3.2
```

4. **Sorting a List**
    - Create an ArrayList of integers.
    - Add ten numbers to it.
    - Sort the list in **ascending** and **descending** order using Collections.sort().
    - Print both sorted lists.

```
import java.util.ArrayList;

import java.util.Collections;

import java.util.Scanner;

public class ListSorting {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        ArrayList<Integer> numbers = new ArrayList<>();

        System.out.println("Enter 10 numbers:");

        for (int i = 0; i < 10; i++) {

            numbers.add(scanner.nextInt());

        }

        Collections.sort(numbers);

        System.out.println("List sorted in ascending order:");

        for (int number : numbers) {

            System.out.println(number);

        }

        Collections.sort(numbers, Collections.reverseOrder());

        System.out.println("List sorted in descending order:");

        for (int number : numbers) {

            System.out.println(number);
```

```
    }
  }
}
```

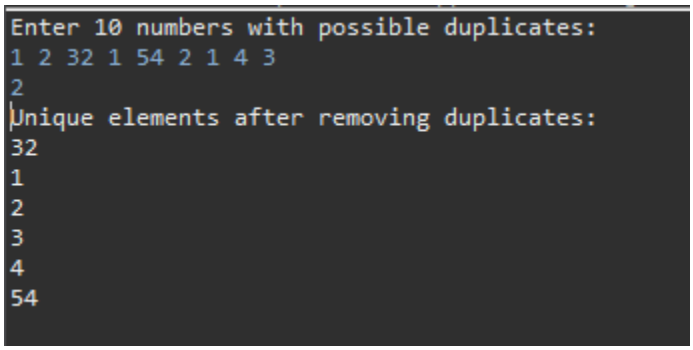
```
Enter 10 numbers:
34
23
1
534
75
26
72
5
90
6
List sorted in ascending order:
1
5
6
23
26
34
72
75
90
534
List sorted in descending order:
534
90
75
72
34
26
23
6
5
1
```

5. Remove Duplicates from a List

- Create an ArrayList with duplicate numbers.
- Remove duplicates by converting the list into a HashSet, then back to a list.
- Print the unique elements.

```
import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Scanner;

public class RemoveDuplicates {

  public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);
        ArrayList<Integer> numbers = new ArrayList<>();
        System.out.println("Enter 10 numbers with possible duplicates:");
        for (int i = 0; i < 10; i++) {
            numbers.add(scanner.nextInt());
        }
        HashSet<Integer> uniqueNumbersSet = new HashSet<>(numbers);
        List<Integer> uniqueNumbersList = new ArrayList<>(uniqueNumbersSet);
        System.out.println("Unique elements after removing duplicates:");
        for (int number : uniqueNumbersList) {
            System.out.println(number);
        }
    }
}
```



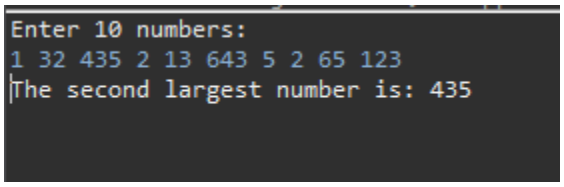
6.Find the Second Largest Number

- Create an ArrayList of integers.
- Find the second-largest number without sorting the list.

```
import java.util.ArrayList;
```

```
import java.util.Scanner;
public class SecondLargestNumber {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ArrayList<Integer> numbers = new ArrayList<>();
        System.out.println("Enter 10 numbers:");
        for (int i = 0; i < 10; i++) {
            numbers.add(scanner.nextInt());
        }
       int largest = Integer.MIN_VALUE;
        int secondLargest = Integer.MIN_VALUE;
        for (int num : numbers) {
            if (num > largest) {
                secondLargest = largest;
                largest = num;
            } else if (num > secondLargest && num < largest) {
                secondLargest = num;
            }
        }
        if (secondLargest != Integer.MIN_VALUE) {
            System.out.println("The second largest number is: " + secondLargest);
        } else {
            System.out.println("There is no second largest number.");
```

```
    }
  }
}
```



7. **Reversing a List Without Using Built-in Methods**
    - Create an ArrayList of strings.
    - Write a function to reverse the list manually **without using** Collections.reverse().

```
import java.util.ArrayList;

import java.util.Scanner;

public class ReverseList {

  public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    ArrayList<String> strings = new ArrayList<>();

    System.out.println("Enter 5 strings:");

    for (int i = 0; i < 5; i++) {

      strings.add(scanner.nextLine());

    }

    reverseList(strings);

    System.out.println("Reversed List:");

    for (String str : strings) {

      System.out.println(str);

    }
```

```
    }

    public static void reverseList(ArrayList<String> list) {
        int start = 0;
        int end = list.size() - 1;
        while (start < end) {
            String temp = list.get(start);
            list.set(start, list.get(end));
            list.set(end, temp);


            start++;
            end--;
        }
    }
}
```

```
Enter 5 strings:
hi
hello
welcome
to
nec
Reversed List:
nec
to
welcome
hello
hi
```

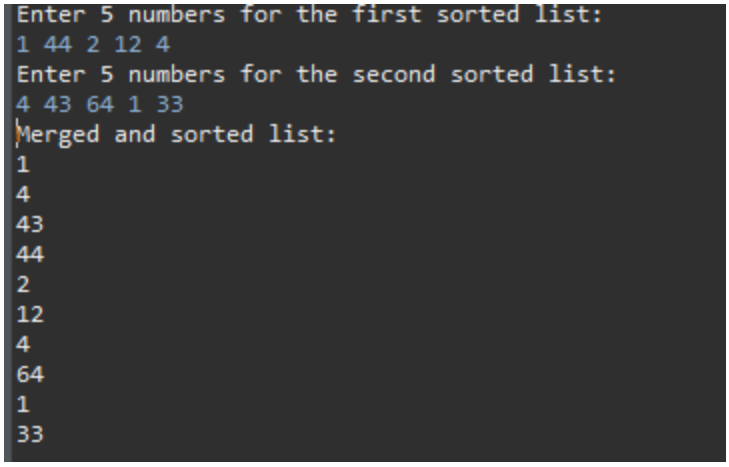
8.Merging Two Sorted Lists

- Given two sorted ArrayList<Integer>, merge them into a single sorted list.
- Ensure the final list remains sorted.

```
import java.util.ArrayList;
```

```
import java.util.Scanner;

public class MergeSortedLists {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        ArrayList<Integer> list1 = new ArrayList<>();

        ArrayList<Integer> list2 = new ArrayList<>();

        System.out.println("Enter 5 numbers for the first sorted list:");

     for (int i = 0; i < 5; i++) {

            list1.add(scanner.nextInt());

        }

        System.out.println("Enter 5 numbers for the second sorted list:");

        for (int i = 0; i < 5; i++) {

            list2.add(scanner.nextInt());

        }

        ArrayList<Integer> mergedList = mergeSortedLists(list1, list2);

        System.out.println("Merged and sorted list:");

        for (int num : mergedList) {

            System.out.println(num);

        }

    }

    public static ArrayList<Integer> mergeSortedLists(ArrayList<Integer> list1,
ArrayList<Integer> list2) {

        ArrayList<Integer> mergedList = new ArrayList<>();

        int i = 0, j = 0;
```

```
        while (i < list1.size() && j < list2.size()) {
            if (list1.get(i) <= list2.get(j)) {
                mergedList.add(list1.get(i));
                i++;
            } else {
                mergedList.add(list2.get(j));
                j++;
            }
        }
        while (i < list1.size()) {
            mergedList.add(list1.get(i));
            i++;
        }
        while (j < list2.size()) {
            mergedList.add(list2.get(j));
            j++;
        }
        return mergedList;
    }
}
```



9.Find the Most Frequent Element

- Create an ArrayList<Integer> with random numbers.
- Find the most frequently occurring number in the list.

```
import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class MostFrequentElement {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        ArrayList<Integer> numbers = new ArrayList<>();

        System.out.println("Enter the number of elements in the list:");

        int n = scanner.nextInt();

        System.out.println("Enter the numbers:");

        for (int i = 0; i < n; i++) {

            numbers.add(scanner.nextInt());

        }
```

```
        int mostFrequent = findMostFrequent(numbers);
        System.out.println("Most frequent element: " + mostFrequent);
    }
    public static int findMostFrequent(ArrayList<Integer> list) {
        Map<Integer, Integer> frequencyMap = new HashMap<>();
        for (int num : list) {
            frequencyMap.put(num, frequencyMap.getOrDefault(num, 0) + 1);
        }
        int mostFrequent = list.get(0);
        for (int num : frequencyMap.keySet()) {
            if (frequencyMap.get(num) > frequencyMap.get(mostFrequent)) {
                mostFrequent = num;
            }
        }
        return mostFrequent;
    }
}
```

```
Enter the number of elements in the list:
5
Enter the numbers:
23 2 2 54 1
Most frequent element: 2
```

2. LinkedList-Based Questions

1.Basic LinkedList Operations

- Create a LinkedList<String> and add five names.
- Print all elements using a loop.

- Remove the first and last element and print the updated list.

```
import java.util.LinkedList;
import java.util.Scanner;
public class LinkedListExample {
    private LinkedList<String> namesList;
    public LinkedListExample() {
        namesList = new LinkedList<>();
    }

    public LinkedList<String> getNamesList() {
        return namesList;
    }
    public void setNamesList(LinkedList<String> namesList) {
        this.namesList = namesList;
    }
    public void addNames(Scanner scanner) {
        for (int i = 0; i < 5; i++) {
            System.out.println("Enter name #" + (i + 1) + ": ");
            String name = scanner.nextLine();
            namesList.add(name);
        }
    }
    public void printNames() {
        System.out.println("\nNames in the LinkedList:");
        for (String name : namesList) {
            System.out.println(name);
        }
    }
    public void removeFirstAndLast() {
        if (!namesList.isEmpty()) {
            System.out.println("\nRemoving first and last names...");
            namesList.removeFirst();
            namesList.removeLast();
        } else {
            System.out.println("\nThe list is empty!");
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        LinkedListExample linkedListExample = new LinkedListExample();
        linkedListExample.addNames(scanner);
        linkedListExample.printNames();
        linkedListExample.removeFirstAndLast();
```

```
        linkedListExample.printNames();
        scanner.close();
    }
}
```

```
Enter name #1:
John
Enter name #2:
Joseph
Enter name #3:
Mary
Enter name #4:
Leo
Enter name #5:
Krish

Names in the LinkedList:
John
Joseph
Mary
Leo
Krish

Removing first and last names...

Names in the LinkedList:
Joseph
Mary
Leo
```

2.Accessing Elements in LinkedList

- Create a LinkedList<Integer>.
- Add five numbers and retrieve the **first** and **last** element using built-in methods.

```
import java.util.LinkedList;
import java.util.Scanner;
public class LinkedListAccess {
    private LinkedList<Integer> numbersList;
    public LinkedListAccess() {
        numbersList = new LinkedList<>();
    }
    public LinkedList<Integer> getNumbersList() {
        return numbersList;
    }
    public void setNumbersList(LinkedList<Integer> numbersList) {
        this.numbersList = numbersList;
    }
    public void addNumbers(Scanner scanner) {
        for (int i = 0; i < 5; i++) {
            System.out.println("Enter number #" + (i + 1) + ": ");
            int number = scanner.nextInt();
            numbersList.add(number);
```

```
        }
    }
    public void printFirstAndLast() {
        if (!numbersList.isEmpty()) {
            System.out.println("\nFirst Element: " + numbersList.getFirst());
            System.out.println("Last Element: " + numbersList.getLast());
        } else {
            System.out.println("\nThe list is empty!");
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        LinkedListAccess linkedListAccess = new LinkedListAccess();
        linkedListAccess.addNumbers(scanner);
        linkedListAccess.printFirstAndLast();
        scanner.close();
    }
}
```

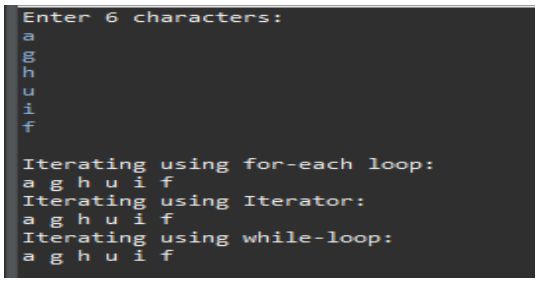
```
Enter number #1:
10
Enter number #2:
20
Enter number #3:
30
Enter number #4:
40
Enter number #5:
50

First Element: 10
Last Element: 50
```

3.Iterate Over a LinkedList

- Create a LinkedList<Character> and add six characters.
- Iterate over it using **for-each**, **iterator**, and **while-loop**.

```
package linkedlist;
import java.util.LinkedList;
import java.util.Iterator;
import java.util.Scanner;
public class LinkedListDemo {
    private LinkedList<Character> charList;
    public LinkedListDemo() {
        charList = new LinkedList<>();
    }
    public LinkedList<Character> getCharList() {
```

```
        return charList;
    }
    public void setCharList(LinkedList<Character> charList) {
        this.charList = charList;
    }
    public void addCharacters(Scanner scanner) {
        System.out.println("Enter 6 characters:");
        for (int i = 0; i < 6; i++) {
            char ch = scanner.next().charAt(0);
            charList.add(ch);
        }
    }
    public void iterateForEach() {
        System.out.println("\nIterating using for-each loop:");
        for (char ch : charList) {
            System.out.print(ch + " ");
        }
    }
    public void iterateIterator() {
        System.out.println("\nIterating using Iterator:");
        Iterator<Character> iterator = charList.iterator();
        while (iterator.hasNext()) {
            System.out.print(iterator.next() + " ");
        }
    }
    public void iterateWhileLoop() {
        System.out.println("\nIterating using while-loop:");
        int i = 0;
        while (i < charList.size()) {
            System.out.print(charList.get(i) + " ");
            i++;
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        LinkedListDemo listExample = new LinkedListDemo();
        listExample.addCharacters(scanner);
        listExample.iterateForEach();
        listExample.iterateIterator();
        listExample.iterateWhileLoop();
        scanner.close();
    }
}
```

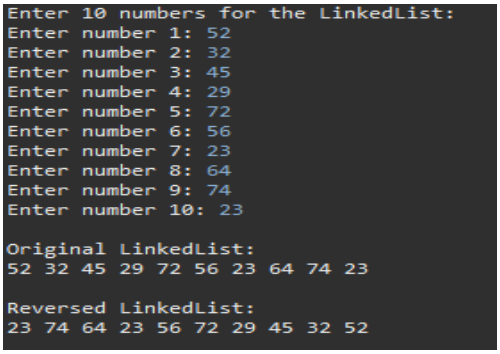


4.**Reverse a LinkedList**

- Create a LinkedList<Integer> with ten numbers.
- Reverse the linked list **without using** Collections.reverse().

```
package linkedlist;
import java.util.LinkedList;
import java.util.Scanner;

class LinkedListReversal {
   static class Node {
      int data;
      Node next;
      public Node(int data) {
         this.data = data;
         this.next = null;
      }
   }
   static class MyLinkedList {
      private Node head;
      public MyLinkedList() {
         this.head = null;
      }
      public Node getHead() {
         return head;
      }
      public void setHead(Node head) {
         this.head = head;
      }
      public void add(int data) {
         Node newNode = new Node(data);
         if (head == null) {
            head = newNode;
         } else {
            Node temp = head;
            while (temp.next != null) {
               temp = temp.next;
            }
            temp.next = newNode;
         }
```

```
        }
        public void reverse() {
            Node prev = null;
            Node current = head;
            Node next = null;

            while (current != null) {
                next = current.next;
                current.next = prev;
                prev = current;
                current = next;
            }
            head = prev;
        }
        public void printList() {
            Node temp = head;
            while (temp != null) {
                System.out.print(temp.data + " ");
                temp = temp.next;
            }
            System.out.println();
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        MyLinkedList list = new MyLinkedList();
        System.out.println("Enter 10 numbers for the LinkedList:");
        for (int i = 0; i < 10; i++) {
            System.out.print("Enter number " + (i + 1) + ": ");
            int num = scanner.nextInt();
            list.add(num);
        }
        System.out.println("\nOriginal LinkedList:");
        list.printList();
        list.reverse();
        System.out.println("\nReversed LinkedList:");
        list.printList();
        scanner.close();
    }
}
```



5.Find Middle Element in a LinkedList

- Given a LinkedList<Integer>, find the middle element **without using size()**.
- Use **two-pointer technique** (slow and fast pointer).

```
package jeri;
import java.util.Scanner;
class Node {
  private int data;
  private Node next;
  public Node(int data) {
     this.data = data;
     this.next = null;
  }
  public int getData() {
     return data;
  }
  public void setData(int data) {
     this.data = data;
  }
  public Node getNext() {
     return next;
  }
  public void setNext(Node next) {
     this.next = next;
  }
}
class LinkedList {
  private Node head;
  public void add(int data) {
     Node newNode = new Node(data);
     if (head == null) {
        head = newNode;
     } else {
        Node current = head;
        while (current.getNext() != null) {
           current = current.getNext();
        }
        current.setNext(newNode);
     }
  }
  public int findMiddle() {
     if (head == null) {
        throw new IllegalStateException("List is empty");
```

```
        }
        Node slow = head;
        Node fast = head;
        while (fast != null && fast.getNext() != null) {
            slow = slow.getNext();
            fast = fast.getNext().getNext();
        }
        return slow.getData();
    }
    public void printList() {
        Node current = head;
        while (current != null) {
            System.out.print(current.getData() + " -> ");
            current = current.getNext();
        }
        System.out.println("null");
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        LinkedList list = new LinkedList();
        System.out.println("Enter numbers to add to the LinkedList (type 'done' to finish):");
        while (scanner.hasNext()) {
            if (scanner.hasNextInt()) {
                int num = scanner.nextInt();
                list.add(num);
            } else {
                String input = scanner.next();
                if (input.equalsIgnoreCase("done")) {
                    break;
                } else {
                    System.out.println("Invalid input, please enter integers or 'done'");
                }
            }
        }
        System.out.print("LinkedList: ");
        list.printList();
        try {
            int middle = list.findMiddle();
            System.out.println("Middle element is: " + middle);
        } catch (IllegalStateException e) {
            System.out.println(e.getMessage());
        }
        scanner.close();
    }
}
```

```
Enter numbers to add to the LinkedList (type 'done' to finish):
10
20
30
40
50
done
LinkedList: 10 -> 20 -> 30 -> 40 -> 50 -> null
Middle element is: 30
```

6. **Merge Two Sorted LinkedLists**

   - Given two **sorted** LinkedList<Integer>, merge them into a single sorted list.

```
package jeri;
import java.util.Scanner;
class MyNode {
private int data;
private MyNode next;
public MyNode(int data) {
   this.data = data;
   this.next = null;
}
public int getData() {
   return data;
}
public void setData(int data) {
   this.data = data;
}
public MyNode getNext() {
   return next;
}
public void setNext(MyNode next) {
   this.next = next;
}
}
class MyLinkedList {
private MyNode head;
public void add(int data) {
   MyNode newNode = new MyNode(data);
   if (head == null) {
      head = newNode;
   } else {
      MyNode current = head;
      while (current.getNext() != null) {
         current = current.getNext();
      }
      current.setNext(newNode);
   }
}
public static MyLinkedList mergeSorted(MyLinkedList list1, MyLinkedList list2) {
   MyNode head1 = list1.head;
   MyNode head2 = list2.head;
   MyNode dummy = new MyNode(0);
   MyNode current = dummy;
   while (head1 != null && head2 != null) {
      if (head1.getData() <= head2.getData()) {
         current.setNext(head1);
         head1 = head1.getNext();
```

```
        } else {
            current.setNext(head2);
            head2 = head2.getNext();
        }
        current = current.getNext();
    }
    if (head1 != null) {
        current.setNext(head1);
    } else {
        current.setNext(head2);
    }
    MyLinkedList mergedList = new MyLinkedList();
    mergedList.head = dummy.getNext();
    return mergedList;
}
public void printList() {
    MyNode current = head;
    while (current != null) {
        System.out.print(current.getData() + " -> ");
        current = current.getNext();
    }
    System.out.println("null");
}
}
public class Main2 {
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    MyLinkedList list1 = new MyLinkedList();
    MyLinkedList list2 = new MyLinkedList();
    System.out.println("Enter sorted elements for List 1 (type 'done' to stop):");
    while (scanner.hasNext()) {
        if (scanner.hasNextInt()) {
            list1.add(scanner.nextInt());
        } else {
            String input = scanner.next();
            if (input.equalsIgnoreCase("done")) break;
            else System.out.println("Invalid input. Enter integers or 'done'.");
        }
    }
    System.out.println("Enter sorted elements for List 2 (type 'done' to stop):");
    while (scanner.hasNext()) {
        if (scanner.hasNextInt()) {
            list2.add(scanner.nextInt());
        } else {
            String input = scanner.next();
            if (input.equalsIgnoreCase("done")) break;
            else System.out.println("Invalid input. Enter integers or 'done'.");
        }
    }
```

```
    System.out.print("List 1: ");
    list1.printList();
    System.out.print("List 2: ");
    list2.printList();
    MyLinkedList merged = MyLinkedList.mergeSorted(list1, list2);
    System.out.print("Merged Sorted List: ");
    merged.printList();
    scanner.close();
}
}
```

```
Enter sorted elements for List 1 (type 'done' to stop):
1 5 7 done
Enter sorted elements for List 2 (type 'done' to stop):
 2 4 6 done
List 1: 1 -> 5 -> 7 -> null
List 2: 2 -> 4 -> 6 -> null
Merged Sorted List: 1 -> 2 -> 4 -> 5 -> 6 -> 7 -> null
```

7. **Detect a Cycle in a LinkedList**

    - Implement Floyd's Cycle Detection Algorithm to detect if a LinkedList has a loop.

```
package jeri;
import java.util.Scanner;
class MyNode {
private int data;
private MyNode next;
public MyNode(int data) {
    this.data = data;
    this.next = null;
}
public int getData() {
    return data;
}
public void setData(int data) {
    this.data = data;
}
public MyNode getNext() {
    return next;
}
public void setNext(MyNode next) {
    this.next = next;
}
}
class MyLinkedList {
private MyNode head;
```

```
public void add(int data) {
    MyNode newNode = new MyNode(data);
    if (head == null) {
        head = newNode;
    } else {
        MyNode current = head;
        while (current.getNext() != null) {
            current = current.getNext();
        }
        current.setNext(newNode);
    }
}
public boolean hasCycle() {
    MyNode slow = head;
    MyNode fast = head;
    while (fast != null && fast.getNext() != null) {
        slow = slow.getNext();
        fast = fast.getNext().getNext();
        if (slow == fast) {
            return true;
        }
    }
    return false;
}
public void createCycle(int pos) {
    if (pos < 0) return;
    MyNode cycleNode = null;
    MyNode current = head;
    int index = 0;
    while (current.getNext() != null) {
        if (index == pos) {
            cycleNode = current;
        }
        current = current.getNext();
        index++;
    }
    if (cycleNode != null) {
        current.setNext(cycleNode);
    }
}
public void printList(int limit) {
    MyNode current = head;
    int count = 0;
    while (current != null && count < limit) {
        System.out.print(current.getData() + " -> ");
        current = current.getNext();
        count++;
    }
    System.out.println(current == null ? "null" : "...(cycle detected)");
```

```
}
}
public class CycleDetectionApp {
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    MyLinkedList list = new MyLinkedList();
    System.out.println("Enter elements for the LinkedList (type 'done' to stop):");
    while (scanner.hasNext()) {
        if (scanner.hasNextInt()) {
            list.add(scanner.nextInt());
        } else {
            String input = scanner.next();
            if (input.equalsIgnoreCase("done")) break;
            else System.out.println("Invalid input. Enter integers or 'done'.");
        }
    }
    System.out.print("Do you want to create a cycle? Enter index to link last node to (-1 for no cycle): ");
    int cyclePos = scanner.nextInt();
    if (cyclePos >= 0) {
        list.createCycle(cyclePos);
    }
    System.out.println("Checking for cycle...");
    boolean hasCycle = list.hasCycle();
    System.out.print("List preview: ");
    list.printList(15);
    System.out.println(hasCycle ? "Cycle detected in the linked list!" : "No cycle in the linked list.");
    scanner.close();
}
}
```

```
Enter elements for the LinkedList (type 'done' to stop):
1 2 3 4 5 done
Do you want to create a cycle? Enter index to link last node to (-1 for no cycle): -1
Checking for cycle...
List preview: 1 -> 2 -> 3 -> 4 -> 5 -> null
No cycle in the linked list.
```

8. **Remove Nth Node from End**

- Given a LinkedList, remove the **Nth node from the end** in a single pass.

```
package jeri;
import java.util.Scanner;
class MyNode {
private int data;
private MyNode next;
```

```
public MyNode(int data) {
   this.data = data;
   this.next = null;
}
public int getData() {
   return data;
}
public MyNode getNext() {
   return next;
}
public void setNext(MyNode next) {
   this.next = next;
}
}
class MyLinkedList {
private MyNode head;
public void add(int data) {
   MyNode newNode = new MyNode(data);
   if (head == null) {
      head = newNode;
   } else {
      MyNode current = head;
      while (current.getNext() != null) {
         current = current.getNext();
      }
      current.setNext(newNode);
   }
}
public void removeNthFromEnd(int n) {
   MyNode dummy = new MyNode(0);
   dummy.setNext(head);
   MyNode fast = dummy;
   MyNode slow = dummy;
   for (int i = 0; i <= n; i++) {
      if (fast == null) return;
      fast = fast.getNext();
   }
   while (fast != null) {
      fast = fast.getNext();
      slow = slow.getNext();
   }
   if (slow.getNext() != null) {
      slow.setNext(slow.getNext().getNext());
   }
   head = dummy.getNext();
}
public void printList() {
   MyNode current = head;
   while (current != null) {
```

```
        System.out.print(current.getData() + " -> ");
        current = current.getNext();
    }
    System.out.println("null");
}
}
public class RemoveNthFromEndApp {
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    MyLinkedList list = new MyLinkedList();
    System.out.println("Enter elements for the LinkedList (type 'done' to stop):");
    while (scanner.hasNext()) {
        if (scanner.hasNextInt()) {
            list.add(scanner.nextInt());
        } else {
            String input = scanner.next();
            if (input.equalsIgnoreCase("done")) break;
            else System.out.println("Invalid input. Enter integers or 'done'.");
        }
    }
    System.out.print("Enter value of N (Nth node from end to remove): ");
    int n = scanner.nextInt();
    System.out.println("Original List:");
    list.printList();
    list.removeNthFromEnd(n);
    System.out.println("List after removing " + n + "th node from the end:");
    list.printList();
    scanner.close();
}
}
```

```
Enter elements for the LinkedList (type 'done' to stop):
10 20 30 40 50 done
Enter value of N (Nth node from end to remove): 2
Original List:
10 -> 20 -> 30 -> 40 -> 50 -> null
List after removing 2th node from the end:
10 -> 20 -> 30 -> 50 -> null
```

9. **Implement a Doubly LinkedList**

   - Implement a **custom doubly linked list** with add(), remove(), reverse(), and print() methods.

```
package mav.example;
import java.util.Scanner;
class Node {
private int data;
private Node next;
private Node prev;
public Node(int data) {
   this.data = data;
   this.next = null;
   this.prev = null;
}
public int getData() {
   return data;
}
public void setData(int data) {
   this.data = data;
}
public Node getNext() {
   return next;
}
public void setNext(Node next) {
   this.next = next;
}
public Node getPrev() {
   return prev;
}
public void setPrev(Node prev) {
   this.prev = prev;
}
}
class DoublyLinkedList {
private Node head;
public void add(int data) {
   Node newNode = new Node(data);
   if (head == null) {
      head = newNode;
      return;
   }
   Node temp = head;
   while (temp.getNext() != null) {
      temp = temp.getNext();
   }
```

```
    temp.setNext(newNode);
    newNode.setPrev(temp);
}
public void remove(int data) {
    if (head == null) return;
    Node temp = head;
    if (temp.getData() == data) {
        head = temp.getNext();
        if (head != null) {
            head.setPrev(null);
        }
        return;
    }
    while (temp != null && temp.getData() != data) {
        temp = temp.getNext();
    }
    if (temp == null) return; // Not found
    if (temp.getNext() != null) {
        temp.getNext().setPrev(temp.getPrev());
    }
    if (temp.getPrev() != null) {
        temp.getPrev().setNext(temp.getNext());
    }
}
public void reverse() {
    Node current = head;
    Node temp = null;
    while (current != null) {
        temp = current.getPrev();
        current.setPrev(current.getNext());
        current.setNext(temp);
        current = current.getPrev();
    }
    if (temp != null) {
        head = temp.getPrev();
    }
}
public void print() {
    Node temp = head;
    while (temp != null) {
        System.out.print(temp.getData() + " ");
        temp = temp.getNext();
    }
    System.out.println();
}
}
public class DoublyLinkedListDemo {
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
```

```
    DoublyLinkedList list = new DoublyLinkedList();
    System.out.println("Enter elements to add to the list (type -1 to stop):");
    while (true) {
        int input = scanner.nextInt();
        if (input == -1) break;
        list.add(input);
    }
    System.out.print("Current list: ");
    list.print();
    System.out.print("Enter a value to remove: ");
    int valToRemove = scanner.nextInt();
    list.remove(valToRemove);
    System.out.print("After removal: ");
    list.print();
    System.out.println("Reversing the list...");
    list.reverse();
    System.out.print("Reversed list: ");
    list.print();
    scanner.close();
}
}
```

```
Enter elements to add to the list (type -1 to stop):

5
7
8
9
0
4
3
15
77
-1
Current list: 5 7 8 9 0 4 3 15 77
Enter a value to remove: 4
After removal: 5 7 8 9 0 3 15 77
Reversing the list...
Reversed list: 77 15 3 0 9 8 7 5
```

3. Stack-Based Questions

1. **Basic Stack Operations**
    - Create a Stack<Integer>.
    - Push five numbers onto the stack.
    - Pop the top element and print the remaining stack.

```
package collection;
	import java.util.Scanner;
	import java.util.Stack;
	class IntegerStack {
	   private Stack<Integer> stack = new Stack<>();
	   public void setElement(int element) {
	      stack.push(element);
	   }
	   public int getTopElement() {
	      return stack.peek();
	   }
	   public int popElement() {
	      return stack.pop();
	   }
	   public Stack<Integer> getStack() {
	      return stack;
	   }
	}
	public class BasicStackOperations {
	   public static void main(String[] args) {
	      Scanner scanner = new Scanner(System.in);
	      IntegerStack integerStack = new IntegerStack();
	      System.out.println("Enter 5 integers to push onto the stack:");
	      for (int i = 0; i < 5; i++) {
	         int num = scanner.nextInt();
	         integerStack.setElement(num);
	      }
	      System.out.println("Initial Stack: " + integerStack.getStack());
	      int popped = integerStack.popElement();
	      System.out.println("Popped Element: " + popped);
	      System.out.println("Remaining Stack: " + integerStack.getStack());
	   }
	}
```

```
Enter 5 integers to push onto the stack:
45
64
23
12
34
Initial Stack: [45, 64, 23, 12, 34]
Popped Element: 34
Remaining Stack: [45, 64, 23, 12]
```

2. **Check if Stack is Empty**

Implement a method that checks if a given stack is empty and returns true/false.

```
package collection;
import java.util.Stack;
public class StackCheck {
  public static boolean isStackEmpty(Stack<Integer> stack) {
     return stack.isEmpty();
  }
  public static void main(String[] args) {
     Stack<Integer> stack = new Stack<>();
     System.out.println("Is stack empty? " + isStackEmpty(stack));
     stack.push(100);
     System.out.println("Is stack empty after push? " + isStackEmpty(stack));
  }
}
```

```
Is stack empty? true
Is stack empty after push? false
```

3. **Reverse a String using Stack**

- Given a string, use a **Stack** to reverse it.

```
package collection;
import java.util.Scanner;
import java.util.Stack;
public class StringReverser {
  public static String reverseString(String input) {
     Stack<Character> stack = new Stack<>();
     for (char ch : input.toCharArray()) {
        stack.push(ch);
     }
     StringBuilder reversed = new StringBuilder();
     while (!stack.isEmpty()) {
        reversed.append(stack.pop());
     }
     return reversed.toString();
```

```
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string to reverse: ");
        String input = scanner.nextLine();
        String reversed = reverseString(input);
        System.out.println("Reversed String: " + reversed);
    }
}
```

```
Enter a string to reverse: helloworld
Reversed String: dlrowolleh
```

4. **Next Greater Element**

    - Given an array [4, 5, 2, 10, 8], find the **next greater element** for each element using a Stack.

```
package collection;
import java.util.Scanner;
import java.util.Stack;
public class NextGreaterElement {
    public static void findNextGreaterElements(int[] arr) {
        int n = arr.length;
        int[] result = new int[n];
        Stack<Integer> stack = new Stack<>();
        for (int i = n - 1; i >= 0; i--) {
            while (!stack.isEmpty() && stack.peek() <= arr[i]) {
                stack.pop();
            }
            result[i] = stack.isEmpty() ? -1 : stack.peek();
            stack.push(arr[i]);
        }
        System.out.println("Element\tNext Greater");
        for (int i = 0; i < n; i++) {
            System.out.println(arr[i] + "\t" + result[i]);
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter number of elements: ");
```

```
        int n = scanner.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        findNextGreaterElements(arr);
    }
}
```

```
Enter number of elements: 5
Enter the elements:
2
3
4
5
6
Element Next Greater
2       3
3       4
4       5
5       6
6       -1
```

5. **Balanced Parentheses Checker**

    - Given a string of brackets ("({[]})"), check if it is **balanced** using a Stack.

```
package collection;
import java.util.Scanner;
import java.util.Stack;
public class BalancedParenthesesChecker {
    public static boolean isBalanced(String str) {
        Stack<Character> stack = new Stack<>();
        for (char ch : str.toCharArray()) {
            if (ch == '(' || ch == '{' || ch == '[') {
                stack.push(ch);
            } else if (ch == ')' || ch == '}' || ch == ']') {
                if (stack.isEmpty()) return false;
                char top = stack.pop();
                if ((ch == ')' && top != '(') ||
                    (ch == '}' && top != '{') ||
                    (ch == ']' && top != '[')) {
                    return false;
```

```
            }
        }
    }
    return stack.isEmpty();
  }
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter a string of brackets: ");
    String input = scanner.nextLine();
    if (isBalanced(input)) {
      System.out.println("The string is balanced.");
    } else {
      System.out.println("The string is not balanced.");
    }
  }
}
```

```
Enter a string of brackets: {[}()}
The string is not balanced.
```

6. **Implement Two Stacks in One Array**

    - Implement two Stacks in a **single array** with push/pop operations.

```
package collection;
import java.util.Scanner;
public class TwoStacks {
  int[] arr;
  int top1, top2, size;
  public TwoStacks(int size) {
    this.size = size;
    arr = new int[size];
    top1 = -1;
    top2 = size;
  }
  public void push1(int value) {
    if (top1 + 1 < top2) {
      arr[++top1] = value;
    } else {
```

```
            System.out.println("Stack Overflow in Stack 1");
        }
    }
    public void push2(int value) {
        if (top1 + 1 < top2) {
            arr[--top2] = value;
        } else {
            System.out.println("Stack Overflow in Stack 2");
        }
    }
    public int pop1() {
        if (top1 >= 0) {
            return arr[top1--];
        } else {
            System.out.println("Stack Underflow in Stack 1");
            return -1;
        }
    }
    public int pop2() {
        if (top2 < size) {
            return arr[top2++];
        } else {
            System.out.println("Stack Underflow in Stack 2");
            return -1;
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int size = scanner.nextInt();
        TwoStacks ts = new TwoStacks(size);
        while (true) {
            System.out.println("\nChoose an operation:");
            System.out.println("1. Push to Stack 1");
            System.out.println("2. Push to Stack 2");
            System.out.println("3. Pop from Stack 1");
            System.out.println("4. Pop from Stack 2");
            System.out.println("5. Exit");
            System.out.print("Your choice: ");
            int choice = scanner.nextInt();
```

```
        switch (choice) {
            case 1:
                System.out.print("Enter value to push to Stack 1: ");
                int val1 = scanner.nextInt();
                ts.push1(val1);
                break;
            case 2:
                System.out.print("Enter value to push to Stack 2: ");
                int val2 = scanner.nextInt();
                ts.push2(val2);
                break;
            case 3:
                int popped1 = ts.pop1();
                if (popped1 != -1)
                    System.out.println("Popped from Stack 1: " + popped1);
                break;
            case 4:
                int popped2 = ts.pop2();
                if (popped2 != -1)
                    System.out.println("Popped from Stack 2: " + popped2);
                break;
            case 5:
                System.out.println("Exiting...");
                return;
            default:
                System.out.println("Invalid choice.");
        }
    }
}
}
```

```
Enter value to push to Stack 2: 3

Choose an operation:
1. Push to Stack 1
2. Push to Stack 2
3. Pop from Stack 1
4. Pop from Stack 2
5. Exit
Your choice: 1
Enter value to push to Stack 1: 3

Choose an operation:
1. Push to Stack 1
2. Push to Stack 2
3. Pop from Stack 1
4. Pop from Stack 2
5. Exit
Your choice: 3
Popped from Stack 1: 3

Choose an operation:
1. Push to Stack 1
2. Push to Stack 2
3. Pop from Stack 1
4. Pop from Stack 2
5. Exit
Your choice:
4
Popped from Stack 2: 3

Choose an operation:
1. Push to Stack 1
2. Push to Stack 2
3. Pop from Stack 1
4. Pop from Stack 2
5. Exit
```

7. **Sort a Stack Without Extra Space**

- Given a Stack<Integer>, sort it **without using extra space or another stack**.

```
package collection;
import java.util.Scanner;
import java.util.Stack;
public class SortStack {
  public static void sortStack(Stack<Integer> stack) {
    if (!stack.isEmpty()) {
      int temp = stack.pop();
      sortStack(stack);
      insertSorted(stack, temp);
    }
  }
  private static void insertSorted(Stack<Integer> stack, int element) {
    if (stack.isEmpty() || stack.peek() <= element) {
      stack.push(element);
    } else {
      int temp = stack.pop();
      insertSorted(stack, element);
      stack.push(temp);
    }
  }
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    Stack<Integer> stack = new Stack<>();
    System.out.print("Enter number of elements: ");
    int n = scanner.nextInt();
    System.out.println("Enter the elements:");
    for (int i = 0; i < n; i++) {
```

```
            stack.push(scanner.nextInt());
        }
        System.out.println("Original Stack: " + stack);
        sortStack(stack);
        System.out.println("Sorted Stack: " + stack);
    }
}
```

```
Enter number of elements: 5
Enter the elements:
3
754
2
13
5
Original Stack: [3, 754, 2, 13, 5]
Sorted Stack: [2, 3, 5, 13, 754]
```

8. **Implement a Stack with min() in O(1)**
    - Design a stack that supports push(), pop(), and min() in **O(1) time complexity**.

```
package hello;
import java.util.Scanner;
import java.util.Stack;
public class MinStack {
  private Stack<Integer> mainStack;
  private Stack<Integer> minStack;
  public MinStack() {
     mainStack = new Stack<>();
     minStack = new Stack<>();
  }
  public void push(int x) {
     mainStack.push(x);
     if (minStack.isEmpty() || x <= minStack.peek()) {
        minStack.push(x);
     }
  }
  public void pop() {
     if (mainStack.isEmpty()) return;
     int popped = mainStack.pop();
     if (popped == minStack.peek()) {
        minStack.pop();
     }
  }
```

```
    public int top() {
        if (mainStack.isEmpty()) throw new RuntimeException("Stack is empty");
        return mainStack.peek();
    }
    public int min() {
        if (minStack.isEmpty()) throw new RuntimeException("Stack is empty");
        return minStack.peek();
    }
public static void main(String[] args) {
    MinStack stack = new MinStack();
    Scanner scanner = new Scanner(System.in);
    while (true) {
        System.out.println("1. Push\n2. Pop\n3. Top\n4. Min\n5. Exit");
        int choice = scanner.nextInt();
        switch (choice) {
            case 1:
                System.out.print("Enter value to push: ");
                int val = scanner.nextInt();
                stack.push(val);
                break;
            case 2:
                stack.pop();
                System.out.println("Popped");
                break;
            case 3:
                System.out.println("Top: " + stack.top());
                break;
            case 4:
                System.out.println("Min: " + stack.min());
                break;
            case 5:
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice");
        }
    }
}
}
```

```
1
Enter value to push: 23
1. Push
2. Pop
3. Top
4. Min
5. Exit
2
Popped
1. Push
2. Pop
3. Top
4. Min
5. Exit
1
Enter value to push: 34
1. Push
2. Pop
3. Top
4. Min
5. Exit
3
Top: 34
1. Push
2. Pop
3. Top
4. Min
5. Exit
1
Enter value to push: 23
1. Push
2. Pop
3. Top
4. Min
5. Exit
4
Min: 23
```

9.. **Evaluate Postfix Expression**

- Implement an algorithm to evaluate a **postfix expression** using a stack.

```
package hello;
import java.util.Scanner;
import java.util.Stack;
public class PostfixEvaluator {
  public static int evaluate(String expression) {
    Stack<Integer> stack = new Stack<>();
    String[] tokens = expression.split(" ");
    for (String token : tokens) {
      if (token.matches("-?\\d+")) {
        stack.push(Integer.parseInt(token));
      } else {
        int b = stack.pop();
        int a = stack.pop();
        switch (token) {
          case "+": stack.push(a + b); break;
          case "-": stack.push(a - b); break;
          case "*": stack.push(a * b); break;
          case "/": stack.push(a / b); break;
```

```
            }
        }
    }
    return stack.pop();
  }
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter postfix expression (tokens space-separated): ");
    String input = scanner.nextLine();
    int result = evaluate(input);
    System.out.println("Result: " + result);
    scanner.close();
  }
}
```

```
Enter postfix expression (tokens space-separated): 5 6 2 + * 12 4 / -
Result: 37
```

4. Vector based Questions

1. **Basic Vector Operations**
    - Create a Vector<Integer>.
    - Add five numbers to it.
    - Print all elements using a loop.

```
package Vector;
import java.util.Scanner;
import java.util.Vector;
public class VectorInputExample {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    Vector<Integer> numbers = new Vector<>();
    System.out.println("Enter 5 integers:");
    for (int i = 0; i < 5; i++) {
      System.out.print("Enter number " + (i + 1) + ": ");
      int num = scanner.nextInt();
      numbers.add(num);
    }
    System.out.println("\nElements in the Vector:");
    for (int i = 0; i < numbers.size(); i++) {
      System.out.println(numbers.get(i));
    }
    scanner.close();
  }
}
```

```
Enter 5 integers:
Enter number 1: 20
Enter number 2: 30
Enter number 3: 40
Enter number 4: 50
Enter number 5: 60

Elements in the Vector:
20
30
40
50
60
```

2.**Accessing Elements in a Vector**

- Create a Vector<String> with five city names.
- Retrieve the first and last elements using built-in methods.

```
package Vector;
import java.util.Scanner;
import java.util.Vector;
public class VectorAccessInputExample {
  public static void main(String[] args) {
     Scanner scanner = new Scanner(System.in);
     Vector<String> cities = new Vector<>();
     System.out.println("Enter 5 city names:");
     for (int i = 0; i < 5; i++) {
        System.out.print("Enter city " + (i + 1) + ": ");
        String city = scanner.nextLine();
        cities.add(city);
     }
     String firstCity = cities.firstElement();
     String lastCity = cities.lastElement();
     System.out.println("\nFirst city: " + firstCity);
     System.out.println("Last city: " + lastCity);
     scanner.close();}}
```

```
Enter 5 city names:
Enter city 1: arumuganeri
Enter city 2: thoothukudi
Enter city 3: thirunelveli
Enter city 4: tenkasi
Enter city 5: madurai

First city: arumuganeri
Last city: madurai
```

3.**Removing Elements from a Vector**

- Create a Vector<Double> and add five decimal values.
- Remove an element at index 2 and print the updated vector.

```
package Vector;
import java.util.Scanner;
import java.util.Vector;
public class vectorRemoveExample {
  public static void main(String[] args) {
     Scanner scanner = new Scanner(System.in);
     Vector<Double> decimals = new Vector<>();
     System.out.println("Enter 5 decimal values:");
     for (int i = 0; i < 5; i++) {
        System.out.print("Enter value " + (i + 1) + ": ");
        double value = scanner.nextDouble();
        decimals.add(value);
     }
     decimals.remove(2);
     System.out.println("\nUpdated Vector after removing element at index 2:");
     for (int i = 0; i < decimals.size(); i++) {
        System.out.println(decimals.get(i));
     }
     scanner.close();
  }
}
```

```
Enter 5 decimal values:
Enter value 1: 2.334
Enter value 2: 3.45
Enter value 3: 33.55
Enter value 4: 6.745
Enter value 5: 22.45

Updated Vector after removing element at index 2:
2.334
3.45
6.745
22.45
```

4.**Sort a Vector**

- Create a Vector<Integer> with ten numbers.
- Sort the vector in **ascending** and **descending** order using Collections.sort().

```
package Vector;
import java.util.Scanner;
import java.util.Vector;
import java.util.Collections;
```

```
public class VectorSortExample {
  public static void main(String[] args) {
     Scanner scanner = new Scanner(System.in);
     Vector<Integer> numbers = new Vector<>();
     System.out.println("Enter 10 integers:");
     for (int i = 0; i < 10; i++) {
        System.out.print("Enter number " + (i + 1) + ": ");
        int num = scanner.nextInt();
        numbers.add(num);
     }
     Collections.sort(numbers);
     System.out.println("\nVector in Ascending Order:");
     for (int number : numbers) {
        System.out.println(number);
     }
     Collections.sort(numbers, Collections.reverseOrder());
     System.out.println("\nVector in Descending Order:");
     for (int number : numbers) {
        System.out.println(number);
     }
     scanner.close();
  }
}
```

```
Enter 10 integers:
Enter number 1: 45
Enter number 2: 2
Enter number 3: 36
Enter number 4: 16
Enter number 5: 52
Enter number 6: 77
Enter number 7: 66
Enter number 8: 100
Enter number 9: 1
Enter number 10: 110

Vector in Ascending Order:
1
2
16
36
45
52
66
77
100
110

Vector in Descending Order:
110
100
77
66
52
45
36
16
2
1
```

5.**Find Maximum and Minimum in a Vector**

- Given a Vector<Integer>, find the **maximum** and **minimum** element **without sorting**.

```
package Vector;
import java.util.Scanner;
import java.util.Vector;
public class VectorMinMaxExample {
  public static void main(String[] args) {
     Scanner scanner = new Scanner(System.in);
     Vector<Integer> numbers = new Vector<>();
     System.out.print("How many numbers do you want to enter? ");
     int count = scanner.nextInt();
     System.out.println("Enter " + count + " integers:");
     for (int i = 0; i < count; i++) {
        System.out.print("Enter number " + (i + 1) + ": ");
        int num = scanner.nextInt();
        numbers.add(num);
     }
     int max = numbers.get(0);
     int min = numbers.get(0);
     for (int i = 1; i < numbers.size(); i++) {
        if (numbers.get(i) > max) {
           max = numbers.get(i);
        }
        if (numbers.get(i) < min) {
           min = numbers.get(i);
        }
     }
     System.out.println("\nMaximum: " + max);
     System.out.println("Minimum: " + min);
     scanner.close();
  }
}
```

```
How many numbers do you want to enter? 5
Enter 5 integers:
Enter number 1: 3
Enter number 2: 5
Enter number 3: 2
Enter number 4: 7
Enter number 5: 10

Maximum: 10
Minimum: 2
```

6.**Convert a Vector to an ArrayList**

- Convert a Vector<String> into an ArrayList<String> and print both collections.

```
package Vector;
import java.util.Scanner;
import java.util.Vector;
import java.util.ArrayList;
public class VectorToArrayListExample {
  public static void main(String[] args) {
     Scanner scanner = new Scanner(System.in);
     Vector<String> vector = new Vector<>();
     System.out.print("How many strings do you want to enter? ");
     int count = scanner.nextInt();
     scanner.nextLine(); // Consume newline
     System.out.println("Enter " + count + " strings:");
     for (int i = 0; i < count; i++) {
        System.out.print("Enter string " + (i + 1) + ": ");
        String input = scanner.nextLine();
        vector.add(input);
     }
     ArrayList<String> arrayList = new ArrayList<>(vector);
     System.out.println("\nVector contents:");
     for (String s : vector) {
        System.out.println(s);
     }
     System.out.println("\nArrayList contents:");
     for (String s : arrayList) {
        System.out.println(s);
     }
     scanner.close();
  }
}
```

```
How many strings do you want to enter? 5
Enter 5 strings:
Enter string 1: she
Enter string 2: is
Enter string 3: an
Enter string 4: engineering
Enter string 5: student

Vector contents:
she
is
an
engineering
student

ArrayList contents:
she
is
an
engineering
student
```

7.**Reverse a Vector Without Using Collections.reverse()**

- Given a Vector<Integer>, write a function to **reverse it manually**.

```
package Vector;
import java.util.Scanner;
import java.util.Vector;
public class VectorManualReverse {
  public static void main(String[] args) {
     Scanner scanner = new Scanner(System.in);
     Vector<Integer> numbers = new Vector<>();
     System.out.print("How many numbers do you want to enter? ");
     int count = scanner.nextInt();
     System.out.println("Enter " + count + " integers:");
     for (int i = 0; i < count; i++) {
        System.out.print("Enter number " + (i + 1) + ": ");
        int num = scanner.nextInt();
        numbers.add(num);
     }
     reverseVector(numbers);
     System.out.println("\nReversed Vector:");
     for (int num : numbers) {
        System.out.println(num);
     }
     scanner.close();
  }
  public static void reverseVector(Vector<Integer> vec) {
     int left = 0;
     int right = vec.size() - 1;
     while (left < right) {
        int temp = vec.get(left);
        vec.set(left, vec.get(right));
        vec.set(right, temp);
        left++;
        right--;}}}
```

```
How many numbers do you want to enter? 5
Enter 5 integers:
Enter number 1: 33
Enter number 2: 56
Enter number 3: 78
Enter number 4: 92
Enter number 5: 27

Reversed Vector:
27
92
78
56
33
```

8.Find the Most Frequent Element in a Vector

- Given a Vector<Integer>, find the **most frequently occurring** element.

```
package hello;
import java.util.*;
public class MostFrequentElement {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    Vector<Integer> vector = new Vector<>();
    System.out.println("Enter integers (type 'done' to finish):");
    while (true) {
      String input = scanner.nextLine().trim();
      if (input.equalsIgnoreCase("done")) break;
      try {
        int value = Integer.parseInt(input);
        vector.add(value);
      } catch (NumberFormatException e) {
        System.out.println("Invalid input. Please enter an integer or 'done' to finish.");
      }
    }
    if (vector.isEmpty()) {
      System.out.println("No elements entered.");
      return;
    }
    Map<Integer, Integer> frequencyMap = new HashMap<>();
    for (int num : vector) {
      frequencyMap.put(num, frequencyMap.getOrDefault(num, 0) + 1);
    }
    int mostFrequent = vector.get(0);
    int maxCount = 1;
    for (Map.Entry<Integer, Integer> entry : frequencyMap.entrySet()) {
      if (entry.getValue() > maxCount) {
        mostFrequent = entry.getKey();
        maxCount = entry.getValue();
      }
    }
    System.out.println("Most Frequent Element: " + mostFrequent + " (Frequency: " + maxCount + ")");
    scanner.close();
  }
}
```

```
Enter integers (type 'done' to finish):
10
20
10
30
10
40
20
done
Most Frequent Element: 10 (Frequency: 3)
```

9.Implement a Stack Using Vector

- Implement a **custom stack** using Vector<Integer>.
  - Include methods for push(), pop(), peek(), and isEmpty().

```
package hello;
import java.util.*;
class CustomStack {
  private Vector<Integer> stack;
  public CustomStack() {
     stack = new Vector<>();
  }
  public void push(int value) {
     stack.add(value);
  }
  public int pop() {
     if (isEmpty()) throw new RuntimeException("Stack is empty.");
     return stack.remove(stack.size() - 1);
  }
  public int peek() {
     if (isEmpty()) throw new RuntimeException("Stack is empty.");
     return stack.get(stack.size() - 1);
  }
  public boolean isEmpty() {
     return stack.isEmpty();
  }
  public void display() {
     System.out.println("Current Stack: " + stack);
  }
}
public class CustomStackUserInput {
  public static void main(String[] args) {
     Scanner scanner = new Scanner(System.in);
     CustomStack stack = new CustomStack();
     System.out.println("Custom Stack Using Vector");
```

```
        System.out.println("Available commands: push <value>, pop, peek, isEmpty, display, exit");
        while (true) {
            System.out.print("Enter command: ");
            String[] input = scanner.nextLine().trim().split("\\s+");
            try {
                switch (input[0].toLowerCase()) {
                    case "push":
                        if (input.length < 2) {
                            System.out.println("Usage: push <value>");
                        } else {
                            int value = Integer.parseInt(input[1]);
                            stack.push(value);
                        }
                        break;
                    case "pop":
                        System.out.println("Popped: " + stack.pop());
                        break;
                    case "peek":
                        System.out.println("Top: " + stack.peek());
                        break;
                    case "isempty":
                        System.out.println("Is stack empty? " + stack.isEmpty());
                        break;
                    case "display":
                        stack.display();
                        break;
                    case "exit":
                        System.out.println("Exiting...");
                        scanner.close();
                        return;
                    default:
                        System.out.println("Invalid command.");
                }
            } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
            }
        }
    }
}
```

```
Custom Stack Using Vector
Available commands: push <value>, pop, peek, isEmpty, display, exit
Enter command: push 30
Enter command: push 40
Enter command: peek
Top: 40
Enter command: pop
Popped: 40
Enter command: isEmpty
Is stack empty? false
Enter command: display
Current Stack: [30]
Enter command: exit
Exiting...
```

Complex Questions

1. **Student Name List Management**
    - You are creating an application to manage student names.
    - Implement an `ArrayList<String>` to:
        - Add new student names.
        - Remove a student by name.
        - Check if a specific student exists.
        - Display all students in sorted order.

```
package mav.example;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;
class Student {
  private String name;
  public Student(String name) {
     this.name = name;
  }
  public String getName() {
     return name;
  }
  public void setName(String name) {
     this.name = name;
  }
}
class StudentManager {
  private ArrayList<Student> studentList = new ArrayList<>();
  public void addStudent(String name) {
     studentList.add(new Student(name));
  }
```

```
    public boolean removeStudent(String name) {
        for (Student student : studentList) {
            if (student.getName().equalsIgnoreCase(name)) {
                studentList.remove(student);
                return true;
            }
        }
        return false;
    }
    public boolean studentExists(String name) {
        for (Student student : studentList) {
            if (student.getName().equalsIgnoreCase(name)) {
                return true;
            }
        }
        return false;
    }
    public void displayStudentsSorted() {
        ArrayList<String> sortedNames = new ArrayList<>();
        for (Student student : studentList) {
            sortedNames.add(student.getName());
        }
        Collections.sort(sortedNames);
        System.out.println("Sorted Student List:");
        for (String name : sortedNames) {
            System.out.println(name);
        }
    }
}
public class StudentNameListApp {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        StudentManager manager = new StudentManager();
        while (true) {
            System.out.println("\n===== Student Management Menu =====");
            System.out.println("1. Add Student");
            System.out.println("2. Remove Student");
            System.out.println("3. Check if Student Exists");
            System.out.println("4. Display All Students (Sorted)");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // consume newline
            switch (choice) {
                case 1:
                    System.out.print("Enter student name to add: ");
                    String addName = scanner.nextLine();
                    manager.addStudent(addName);
                    System.out.println("Student added.");
```

```
                break;
            case 2:
                System.out.print("Enter student name to remove: ");
                String removeName = scanner.nextLine();
                boolean removed = manager.removeStudent(removeName);
                if (removed) {
                    System.out.println("Student removed.");
                } else {
                    System.out.println("Student not found.");
                }
                break;
            case 3:
                System.out.print("Enter student name to check: ");
                String searchName = scanner.nextLine();
                if (manager.studentExists(searchName)) {
                    System.out.println("Student exists.");
                } else {
                    System.out.println("Student does not exist.");
                }
                break;
            case 4:
                manager.displayStudentsSorted();
                break;
            case 5:
                System.out.println("Exiting program. Goodbye!");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
}
```

```
===== Student Management Menu =====
1. Add Student
2. Remove Student
3. Check if Student Exists
4. Display All Students (Sorted)
5. Exit
Enter your choice: 1
Enter student name to add: Dhoni
Student added.

===== Student Management Menu =====
1. Add Student
2. Remove Student
3. Check if Student Exists
4. Display All Students (Sorted)
5. Exit
Enter your choice: 1
Enter student name to add: Kohli
Student added.

===== Student Management Menu =====
1. Add Student
2. Remove Student
3. Check if Student Exists
4. Display All Students (Sorted)
5. Exit
Enter your choice: 1
Enter student name to add: Rohith
Student added.

===== Student Management Menu =====
1. Add Student
2. Remove Student
3. Check if Student Exists
4. Display All Students (Sorted)
5. Exit
```

```
2. Remove Student
3. Check if Student Exists
4. Display All Students (Sorted)
5. Exit
Enter your choice: 2
Enter student name to remove: Rohith
Student removed.

===== Student Management Menu =====
1. Add Student
2. Remove Student
3. Check if Student Exists
4. Display All Students (Sorted)
5. Exit
Enter your choice: 3
Enter student name to check: Rohith
Student does not exist.

===== Student Management Menu =====
1. Add Student
2. Remove Student
3. Check if Student Exists
4. Display All Students (Sorted)
5. Exit
Enter your choice: 4
Sorted Student List:
Dhoni
Kohli

===== Student Management Menu =====
1. Add Student
2. Remove Student
3. Check if Student Exists
4. Display All Students (Sorted)
5. Exit
Enter your choice: 5
Exiting program. Goodbye!
```

2.Recent Searches in a Mobile App

- Implement an `ArrayList<String>` to store recent search queries.
- If the list exceeds **five elements**, remove the oldest search.
- Display the latest searches when the app opens.package mav.example;

```
package mav.example;
import java.util.ArrayList;
import java.util.Scanner;
class SearchManager {
  private ArrayList<String> recentSearches = new ArrayList<>();
  public void setSearch(String query) {
     if (recentSearches.size() >= 5) {
        recentSearches.remove(0);
     }
     recentSearches.add(query);
  }
```

```
    public ArrayList<String> getRecentSearches() {
        return recentSearches;
    }
    public void displayRecentSearches() {
        System.out.println("Recent Searches:");
        if (recentSearches.isEmpty()) {
            System.out.println("No recent searches found.");
        } else {
            for (int i = recentSearches.size() - 1; i >= 0; i--) {
                System.out.println(recentSearches.get(i));
            }
        }
    }
}
public class RecentSearchApp {
    public static void main(String[] args) {
        SearchManager manager = new SearchManager();
        Scanner scanner = new Scanner(System.in);
        manager.displayRecentSearches(); // Display when app starts
        System.out.println("\nEnter your search queries (type 'exit' to stop):");
        while (true) {
            System.out.print("Search: ");
            String input = scanner.nextLine();
            if (input.equalsIgnoreCase("exit")) break;
            manager.setSearch(input);
        }
        System.out.println("\nUpdated Recent Searches:");
        manager.displayRecentSearches();
    }
}
```

```
Recent Searches:
No recent searches found.

Enter your search queries (type 'exit' to stop):
Search: mobile phone
Search: laptop
Search: earbude
Search: smart watch
Search: exit

Updated Recent Searches:
Recent Searches:
smart watch
earbude
laptop
mobile phone
```

3.Tracking Employee IDs

- You need to maintain a list of employee IDs using `LinkedList<Integer>`.
- Add new employee IDs to the **end** of the list.
- Remove an employee ID when they leave the company.
- Print all active employee IDs.

```
package mav.example;
import java.util.LinkedList;
import java.util.Scanner;
class EmployeeManager {
  private LinkedList<Integer> employeeIDs = new LinkedList<>();
  public void addEmployeeID(int id) {
     employeeIDs.add(id);
  }
  public boolean removeEmployeeID(int id) {
     return employeeIDs.remove(Integer.valueOf(id));
  }
  public LinkedList<Integer> getEmployeeIDs() {
     return employeeIDs;
  }
  public void displayActiveEmployees() {
     if (employeeIDs.isEmpty()) {
        System.out.println("No active employee IDs.");
     } else {
        System.out.println("Active Employee IDs:");
        for (int id : employeeIDs) {
           System.out.println(id);
        }
     }
  }
}
public class EmployeeTrackerApp {
  public static void main(String[] args) {
     EmployeeManager manager = new EmployeeManager();
     Scanner scanner = new Scanner(System.in);
     String choice;
     do {
        System.out.println("\nEmployee Tracker Menu:");
        System.out.println("1. Add Employee ID");
        System.out.println("2. Remove Employee ID");
        System.out.println("3. Display Active Employee IDs");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: ");
        choice = scanner.nextLine();
        switch (choice) {
           case "1":
              System.out.print("Enter Employee ID to add: ");
```

```
                int newID = Integer.parseInt(scanner.nextLine());
                manager.addEmployeeID(newID);
                System.out.println("Employee ID added.");
                break;
            case "2":
                System.out.print("Enter Employee ID to remove: ");
                int removeID = Integer.parseInt(scanner.nextLine());
                if (manager.removeEmployeeID(removeID)) {
                    System.out.println("Employee ID removed.");
                } else {
                    System.out.println("Employee ID not found.");
                }
                break;
            case "3":
                manager.displayActiveEmployees();
                break;
            case "4":
                System.out.println("Exiting application.");
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    } while (!choice.equals("4"));
    scanner.close();
  }
}
```

```
Employee Tracker Menu:
1. Add Employee ID
2. Remove Employee ID
3. Display Active Employee IDs
4. Exit
Enter your choice: 1
Enter Employee ID to add: 101
Employee ID added.

Employee Tracker Menu:
1. Add Employee ID
2. Remove Employee ID
3. Display Active Employee IDs
4. Exit
Enter your choice: 1
Enter Employee ID to add: 102
Employee ID added.

Employee Tracker Menu:
1. Add Employee ID
2. Remove Employee ID
3. Display Active Employee IDs
4. Exit
Enter your choice: 1
Enter Employee ID to add: 103
Employee ID added.
```

```
Employee Tracker Menu:
1. Add Employee ID
2. Remove Employee ID
3. Display Active Employee IDs
4. Exit
Enter your choice: 2
Enter Employee ID to remove: 102
Employee ID removed.

Employee Tracker Menu:
1. Add Employee ID
2. Remove Employee ID
3. Display Active Employee IDs
4. Exit
Enter your choice: 3
Active Employee IDs:
101
103

Employee Tracker Menu:
1. Add Employee ID
2. Remove Employee ID
3. Display Active Employee IDs
4. Exit
Enter your choice: 4
Exiting application.
```

4.Task Manager for a To-Do App

- Implement a **task manager** using `ArrayList<String>`.
- Features:
  - Add tasks in the order they are created.
  - Remove a task once completed.
  - Sort tasks alphabetically.

Display pending tasks.

```
package mav.example;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;
class TaskManager {
  private ArrayList<String> tasks = new ArrayList<>();
  public void addTask(String task) {
     tasks.add(task);
  }
  public boolean removeTask(String task) {
     return tasks.remove(task);
  }
  public ArrayList<String> getTasks() {
     return tasks;
  }
```

```
    public void sortTasks() {
        Collections.sort(tasks);
    }
    public void displayTasks() {
        if (tasks.isEmpty()) {
            System.out.println("No pending tasks.");
        } else {
            System.out.println("Pending Tasks:");
            for (String task : tasks) {
                System.out.println("- " + task);
            }
        }
    }
}
public class ToDoTaskManagerApp {
    public static void main(String[] args) {
        TaskManager manager = new TaskManager();
        Scanner scanner = new Scanner(System.in);
        String choice;
        do {
            System.out.println("\nTo-Do Task Manager Menu:");
            System.out.println("1. Add Task");
            System.out.println("2. Remove Completed Task");
            System.out.println("3. Sort Tasks Alphabetically");
            System.out.println("4. Display Pending Tasks");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextLine();
            switch (choice) {
                case "1":
                    System.out.print("Enter task to add: ");
                    String task = scanner.nextLine();
                    manager.addTask(task);
                    System.out.println("Task added.");
                    break;
                case "2":
                    System.out.print("Enter task to remove: ");
                    String completed = scanner.nextLine();
                    if (manager.removeTask(completed)) {
                        System.out.println("Task removed.");
                    } else {
                        System.out.println("Task not found.");
                    }
                    break;
                case "3":
                    manager.sortTasks();
                    System.out.println("Tasks sorted alphabetically.");
                    break;
                case "4":
```

```
                manager.displayTasks();
                break;
            case "5":
                System.out.println("Exiting Task Manager.");
                break;
            default:
                System.out.println("Invalid choice. Try again.");
        }
    } while (!choice.equals("5"));
    scanner.close();
  }
}
```

```
To-Do Task Manager Menu:
1. Add Task
2. Remove Completed Task
3. Sort Tasks Alphabetically
4. Display Pending Tasks
5. Exit
Enter your choice: 1
Enter task to add: wash dishes
Task added.

To-Do Task Manager Menu:
1. Add Task
2. Remove Completed Task
3. Sort Tasks Alphabetically
4. Display Pending Tasks
5. Exit
Enter your choice: 1
Enter task to add: Buy groceries
Task added.

To-Do Task Manager Menu:
1. Add Task
2. Remove Completed Task
3. Sort Tasks Alphabetically
4. Display Pending Tasks
5. Exit
Enter your choice: 1
Enter task to add: cook the dish
Task added.
```

```
2. Remove Completed Task
3. Sort Tasks Alphabetically
4. Display Pending Tasks
5. Exit
Enter your choice: 2
Enter task to remove: cook the dish
Task removed.

To-Do Task Manager Menu:
1. Add Task
2. Remove Completed Task
3. Sort Tasks Alphabetically
4. Display Pending Tasks
5. Exit
Enter your choice: 3
Tasks sorted alphabetically.

To-Do Task Manager Menu:
1. Add Task
2. Remove Completed Task
3. Sort Tasks Alphabetically
4. Display Pending Tasks
5. Exit
Enter your choice: 4
Pending Tasks:
- Buy groceries
- wash dishes

To-Do Task Manager Menu:
1. Add Task
2. Remove Completed Task
3. Sort Tasks Alphabetically
4. Display Pending Tasks
5. Exit
Enter your choice: 5
Exiting Task Manager.
```

5.Merging Two Ordered Product Lists

- Given two sorted `ArrayList<String>` representing product names from two suppliers:
  - Merge them into a single sorted list without duplicates.

```
package hello;
import java.util.*;
public class MergeProductLists {
  public static void main(String[] args) {
     Scanner scanner = new Scanner(System.in);
     ArrayList<String> supplier1 = new ArrayList<>();
     ArrayList<String> supplier2 = new ArrayList<>();
     System.out.println("Enter sorted product names from Supplier 1 (type 'done' to finish):");
     while (true) {
        String input = scanner.nextLine().trim();
        if (input.equalsIgnoreCase("done")) break;
        if (!input.isEmpty()) supplier1.add(input);
     }
```

```
    System.out.println("Enter sorted product names from Supplier 2 (type 'done' to finish):");
    while (true) {
        String input = scanner.nextLine().trim();
        if (input.equalsIgnoreCase("done")) break;
        if (!input.isEmpty()) supplier2.add(input);
    }
    TreeSet<String> mergedSet = new TreeSet<>();
    mergedSet.addAll(supplier1);
    mergedSet.addAll(supplier2);
    ArrayList<String> mergedList = new ArrayList<>(mergedSet);
    System.out.println("\nMerged Sorted Product List (No Duplicates):");
    for (String product : mergedList) {
        System.out.println(product);
    }
    scanner.close();
  }
}
```

```
Enter sorted product names from Supplier 1 (type 'done' to finish):
laptop
monitor
mouse
done
Enter sorted product names from Supplier 2 (type 'done' to finish):
keyboard
airpods
headphones
mouse
done

Merged Sorted Product List (No Duplicates):
airpods
headphones
keyboard
laptop
monitor
mouse
```

6.Removing Duplicate Entries in a Contact List

- You have an `ArrayList<String>` storing **contact names**.
- Some names are **duplicated**.
- Remove duplicate names while keeping the order intact.

```
package hello;
import java.util.*;
public class RemoveDuplicateContacts {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    ArrayList<String> contacts = new ArrayList<>();
    System.out.println("Enter contact names (type 'done' to finish):");
    while (true) {
        System.out.print("Contact: ");
```

```
        String name = scanner.nextLine().trim();
        if (name.equalsIgnoreCase("done")) break;
        if (!name.isEmpty()) contacts.add(name);
    }
    LinkedHashSet<String> uniqueContacts = new LinkedHashSet<>(contacts);
    ArrayList<String> cleanedList = new ArrayList<>(uniqueContacts);
    System.out.println("\nContact list after removing duplicates:");
    for (String contact : cleanedList) {
        System.out.println(contact);
    }
    scanner.close();
  }
}
```

```
Enter contact names (type 'done' to finish):
Contact: hari
Contact: ria
Contact: ram
Contact: sham
Contact: ria
Contact: jai
Contact: done

Contact list after removing duplicates:
hari
ria
ram
sham
jai
```

7. **Find the Most Frequent Product in Purchase History**
    - You have an `ArrayList<String>` of product purchases (with duplicates).
    - Find the **most frequently purchased product**.

```
package hello;
import java.util.*;
public class MostFrequentProduct {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    ArrayList<String> purchases = new ArrayList<>();
    System.out.println("Enter purchased products one by one (type 'done' to finish):");
    while (true) {
        System.out.print("Product: ");
        String input = scanner.nextLine().trim();
        if (input.equalsIgnoreCase("done")) break;
        if (!input.isEmpty()) purchases.add(input);
    }
    if (purchases.isEmpty()) {
        System.out.println("No purchases entered.");
        return;
```

```
    }
    Map<String, Integer> frequencyMap = new HashMap<>();
    for (String product : purchases) {
      frequencyMap.put(product, frequencyMap.getOrDefault(product, 0) + 1);
    }
    String mostFrequentProduct = null;
    int maxCount = 0;
    for (Map.Entry<String, Integer> entry : frequencyMap.entrySet()) {
      if (entry.getValue() > maxCount) {
        mostFrequentProduct = entry.getKey();
        maxCount = entry.getValue();
      }
    }
    System.out.println("Most Frequent Product: " + mostFrequentProduct + " (Purchased " + maxCount + "
times)");
    scanner.close();
  }
}
```

```
Enter purchased products one by one (type 'done' to finish):
Product: mouse
Product: laptop
Product: monito
Product: mouse
Product: keyboard
Product: mouse
Product: airpods
Product: done
Most Frequent Product: mouse (Purchased 3 times)
```

8.Implementing an Undo Feature in a Text Editor

- You need to implement an **undo feature** using a `LinkedList<String>`.
- Every user action (typing, deleting) is added to the list.
- When the user clicks **Undo**, remove the last action and restore the previous state.

```
package hello;
import java.util.*;
public class TextEditorUndo {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    LinkedList<String> history = new LinkedList<>();
    String currentText = "";
    history.add(currentText);
    System.out.println("Simple Text Editor (type 'undo' to undo, 'exit' to quit):");
    while (true) {
      System.out.print("Enter action (type/delete/undo): ");
```

```
        String action = scanner.nextLine().toLowerCase();
        if (action.equals("exit")) break;
        switch (action) {
            case "type":
                System.out.print("Enter text to append: ");
                String toType = scanner.nextLine();
                currentText += toType;
                history.add(currentText);
                break;
            case "delete":
                System.out.print("Enter number of characters to delete: ");
                int count = Integer.parseInt(scanner.nextLine());
                if (count <= currentText.length()) {
                    currentText = currentText.substring(0, currentText.length() - count);
                    history.add(currentText);
                } else {
                    System.out.println("Cannot delete more characters than current length.");
                }
                break;
            case "undo":
                if (history.size() > 1) {
                    history.removeLast();
                    currentText = history.getLast();
                } else {
                    System.out.println("Nothing to undo.");
                }
                break;
            default:
                System.out.println("Invalid action.");
                break;
        }
        System.out.println("Current Text: \"" + currentText + "\"");
    }
    scanner.close();
  }
}
```

```
Simple Text Editor (type 'undo' to undo, 'exit' to quit):
Enter action (type/delete/undo): type
Enter text to append: hello
Current Text: "hello"
Enter action (type/delete/undo): type
Enter text to append: world
Current Text: "helloworld"
Enter action (type/delete/undo): delete
Enter number of characters to delete: 5
Current Text: "hello"
Enter action (type/delete/undo): undo
Current Text: "helloworld"
Enter action (type/delete/undo): exit
```

9.Auto-Suggestions for a Search Bar

- You have an `ArrayList<String>` of **100,000 words**.
- When a user types a few characters, show the **top 5 matching words**.
- Optimize search performance.

```
package hello;
import java.util.*;
class TrieNode {
  Map<Character, TrieNode> children = new HashMap<>();
  boolean isEndOfWord = false;
}
class Trie {
  TrieNode root = new TrieNode();
  public void insert(String word) {
    TrieNode node = root;
    for (char ch : word.toCharArray()) {
      node = node.children.computeIfAbsent(ch, c -> new TrieNode());
    }
    node.isEndOfWord = true;
  }
  public List<String> getSuggestions(String prefix) {
    TrieNode node = root;
    for (char ch : prefix.toCharArray()) {
      node = node.children.get(ch);
      if (node == null) return new ArrayList<>();
    }
    List<String> results = new ArrayList<>();
    dfs(node, new StringBuilder(prefix), results);
    return results;
  }
  private void dfs(TrieNode node, StringBuilder current, List<String> results) {
    if (results.size() >= 5) return;
    if (node.isEndOfWord) {
      results.add(current.toString());
    }
    for (char ch : node.children.keySet()) {
      current.append(ch);
      dfs(node.children.get(ch), current, results);
      current.deleteCharAt(current.length() - 1);
    }
  }
}
public class AutoSuggestion {
  public static void main(String[] args) {
    List<String> words = Arrays.asList(
        "apple", "app", "apricot", "banana", "band", "bandana", "cat", "cater", "cattle", "dog", "dove", "duck"
    );
```

```
        Trie trie = new Trie();
        for (String word : words) {
            trie.insert(word.toLowerCase());
        }
        Scanner scanner = new Scanner(System.in);
        System.out.println("Type a prefix to get suggestions (type 'exit' to quit):");
        while (true) {
            System.out.print("\nSearch: ");
            String input = scanner.nextLine().toLowerCase();
            if (input.equals("exit")) break;
            List<String> suggestions = trie.getSuggestions(input);
            if (suggestions.isEmpty()) {
                System.out.println("No suggestions found.");
            } else {
                System.out.println("Suggestions: " + suggestions);
            }
        }
        scanner.close();
    }
}
```

```
Type a prefix to get suggestions (type 'exit' to quit):

Search: ap
Suggestions: [app, apple, apricot]

Search: cn
No suggestions found.
```