Introduction to Collections Framework

1. Write a program to demonstrate adding and printing elements from an ArrayList

```
import java.util.ArrayList;

public class ArrayListExample {
    public static void main(String[] args) {
        // Create an ArrayList of Strings
        ArrayList<String> fruits = new ArrayList<>();

        // Adding elements to the ArrayList
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Mango");
        fruits.add("Orange");

        // Printing the elements of the ArrayList
        System.out.println("Fruits in the list:");
        for (String fruit : fruits) {
            System.out.println(fruit);
        }
    }
}
```

2. Show how to use Collections.max() and Collections.min() on a list of integers

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

public class MaxMinExample {
    public static void main(String[] args) {
        // Create an ArrayList of Integers
        ArrayList<Integer> numbers = new ArrayList<>();

        // Add elements to the list
        numbers.add(45);
        numbers.add(10);
        numbers.add(67);
        numbers.add(32);
        numbers.add(89);

        // Find the maximum and minimum values using Collections
        int max = Collections.max(numbers);
        int min = Collections.min(numbers);

        // Display the result
        System.out.println("List: " + numbers);
        System.out.println("Maximum value: " + max);
        System.out.println("Minimum value: " + min);
    }
}
```

3. Demonstrate the use of Collections.sort() on a list of strings.

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

public class SortStringsExample {
    public static void main(String[] args) {
        // Create an ArrayList of Strings
        ArrayList<String> names = new ArrayList<>();

        // Add names to the list
        names.add("Zara");
        names.add("Aman");
        names.add("John");
        names.add("Bella");

        // Display list before sorting
        System.out.println("Before sorting: " + names);

        // Sort the list in alphabetical order
        Collections.sort(names);

        // Display list after sorting
        System.out.println("After sorting: " + names);
    }
}
```

4. You need to store a dynamic list of student names and display them in alphabetical order.

Implement this using a suitable collection.

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

public class StudentList {
    public static void main(String[] args) {
        // Create a dynamic list to store student names
        ArrayList<String> studentNames = new ArrayList<>();

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of students: ");
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        // Take input from the user
        for (int i = 1; i <= n; i++) {
            System.out.print("Enter name of student " + i + ": ");
            String name = scanner.nextLine();
            studentNames.add(name);
        }

        // Sort the list alphabetically
        Collections.sort(studentNames);
```

```
        // Display the sorted list
        System.out.println("\nStudent names in alphabetical order:");
        for (String name : studentNames) {
            System.out.println(name);
        }

        scanner.close();
    }
}
```

5. A user can input any number of integers. Your program should store them and display the sum of all elements using the Collection Framework

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

public class IntegerSumUsingList {
    public static void main(String[] args) {
        // Create a dynamic list to store integers
        ArrayList<Integer> numbers = new ArrayList<>();

        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter integers (type -1 to finish):");

        while (true) {
            int num = scanner.nextInt();
            if (num == -1) {
                break; // Stop input on -1
            }
            numbers.add(num); // Add to list
        }

        // Calculate sum
        int sum = 0;
        for (int n : numbers) {
            sum += n;
        }

        // Display result
        System.out.println("Numbers entered: " + numbers);
        System.out.println("Sum of all elements: " + sum);

        scanner.close();
    }
}
```

List Interface

1. Write a Java program to add, remove, and access elements in an ArrayList.

```
import java.util.ArrayList;

public class ArrayListOperations {
    public static void main(String[] args) {
        // Step 1: Create an ArrayList of Strings
        ArrayList<String> fruits = new ArrayList<>();

        // Step 2: Add elements
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");
        fruits.add("Date");

        System.out.println("Fruits list after adding elements: " + fruits);

        // Step 3: Access elements using index
        System.out.println("First fruit: " + fruits.get(0));  // Apple
        System.out.println("Third fruit: " + fruits.get(2));  // Cherry

        // Step 4: Remove an element
        fruits.remove("Banana"); // Remove by value
        fruits.remove(2);        // Remove by index (removes "Date")

        System.out.println("Fruits list after removing elements: " + fruits);

        // Step 5: Add more elements
        fruits.add("Elderberry");
        fruits.add("Fig");

        System.out.println("Final fruits list: " + fruits);
    }
}
```

2. Implement a LinkedList that stores and prints employee names.

```
import java.util.LinkedList;

public class EmployeeList {
    public static void main(String[] args) {
        // Step 1: Create a LinkedList to store employee names
        LinkedList<String> employees = new LinkedList<>();

        // Step 2: Add employee names to the list
        employees.add("Alice");
        employees.add("Bob");
        employees.add("Charlie");
        employees.add("Diana");

        // Step 3: Print the list of employee names
        System.out.println("Employee Names:");
```

```
            for (String name : employees) {
                System.out.println(name);
            }
        }
    }
```

3. Demonstrate inserting an element at a specific position in a List.

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

public class InsertElement {
    public static void main(String[] args) {
        // Step 1: Create a list of colors
        List<String> colors = new ArrayList<>();

        // Step 2: Add initial elements
        colors.add("Red");
        colors.add("Green");
        colors.add("Blue");

        // Step 3: Display original list
        System.out.println("Original list: " + colors);

        // Step 4: Insert "Yellow" at position 1 (second place)
        colors.add(1, "Yellow");

        // Step 5: Display updated list
        System.out.println("List after inserting 'Yellow' at index 1: " + colors);
    }
}
```

4. You're building a to-do list manager. Use ArrayList to add tasks, remove completed ones, and display pending tasks

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

public class ToDoListManager {
    public static void main(String[] args) {
        ArrayList<String> tasks = new ArrayList<>();
        Scanner sc = new Scanner(System.in);

        while (true) {
            System.out.println("\n--- To-Do List Menu ---");
            System.out.println("1. Add Task");
            System.out.println("2. Remove Task (Mark as Completed)");
            System.out.println("3. View Pending Tasks");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");

            int choice = sc.nextInt();
            sc.nextLine();  // Clear the buffer
```

```
                switch (choice) {
                    case 1:
                        System.out.print("Enter task to add: ");
                        String task = sc.nextLine();
                        tasks.add(task);
                        System.out.println("Task added.");
                        break;

                    case 2:
                        System.out.print("Enter task number to remove: ");
                        int taskNum = sc.nextInt();
                        sc.nextLine();
                        if (taskNum >= 1 && taskNum <= tasks.size()) {
                            tasks.remove(taskNum - 1);
                            System.out.println("Task removed.");
                        } else {
                            System.out.println("Invalid task number.");
                        }
                        break;

                    case 3:
                        System.out.println("\nPending Tasks:");
                        if (tasks.isEmpty()) {
                            System.out.println("No tasks in the list.");
                        } else {
                            for (int i = 0; i < tasks.size(); i++) {
                                System.out.println((i + 1) + ". " + tasks.get(i));
                            }
                        }
                        break;

                    case 4:
                        System.out.println("Exiting To-Do List Manager.");
                        sc.close();
                        return;

                    default:
                        System.out.println("Invalid choice. Try again.");
                }
            }
        }
    }
```

5. Create a simple shopping cart system where users can add/remove products using a List

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

public class ShoppingCart {
    public static void main(String[] args) {
        ArrayList<String> cart = new ArrayList<>();
        Scanner sc = new Scanner(System.in);
        int choice;

        while (true) {
            System.out.println("\n--- Shopping Cart Menu ---");
            System.out.println("1. Add Product");
            System.out.println("2. Remove Product");
            System.out.println("3. View Cart");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();
            sc.nextLine(); // consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter product name to add: ");
                    String product = sc.nextLine();
                    cart.add(product);
                    System.out.println(product + " added to the cart.");
                    break;

                case 2:
                    if (cart.isEmpty()) {
                        System.out.println("Cart is empty!");
                    } else {
                        System.out.print("Enter product number to remove: ");
                        int index = sc.nextInt();
                        sc.nextLine(); // consume newline
                        if (index >= 1 && index <= cart.size()) {
                            String removed = cart.remove(index - 1);
                            System.out.println(removed + " removed from the cart.");
                        } else {
                            System.out.println("Invalid product number.");
                        }
                    }
                    break;

                case 3:
                    System.out.println("Your Shopping Cart:");
                    if (cart.isEmpty()) {
                        System.out.println("Cart is empty.");
                    } else {
```

```
                    for (int i = 0; i < cart.size(); i++) {
                        System.out.println((i + 1) + ". " + cart.get(i));
                    }
                }
                break;

            case 4:
                System.out.println("Thank you for shopping. Exiting...");
                sc.close();
                return;

            default:
                System.out.println("Invalid choice. Try again.");
        }
    }
}
```

Set Interface.

1. Write a program using HashSet to store unique student roll numbers.

```
import java.util.HashSet;
import java.util.Scanner;

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

        System.out.print("How many roll numbers do you want to enter? ");
        int count = sc.nextInt();

        for (int i = 1; i <= count; i++) {
            System.out.print("Enter roll number " + i + ": ");
            int roll = sc.nextInt();

            // Add to set, duplicates will be automatically ignored
            if (rollNumbers.add(roll)) {
                System.out.println("Added successfully.");
            } else {
                System.out.println("Duplicate roll number! Not added.");
            }
        }

        // Display all unique roll numbers
        System.out.println("\nUnique Roll Numbers:");
        for (int num : rollNumbers) {
            System.out.println(num);
        }

        sc.close();
```

```
    }
}
```

2. Demonstrate how to use TreeSet to automatically sort elements.

```
import java.util.Scanner;
import java.util.TreeSet;

public class SortedNamesWithTreeSet {
    public static void main(String[] args) {
        TreeSet<String> names = new TreeSet<>();
        Scanner sc = new Scanner(System.in);

        System.out.print("How many names do you want to enter? ");
        int count = sc.nextInt();
        sc.nextLine(); // consume newline

        for (int i = 1; i <= count; i++) {
            System.out.print("Enter name " + i + ": ");
            String name = sc.nextLine();
            names.add(name);
        }

        System.out.println("\nNames in Sorted Order:");
        for (String name : names) {
            System.out.println(name);
        }

        sc.close();
    }
}
```

3. Use LinkedHashSet to maintain insertion order and prevent duplicates.

```
import java.util.LinkedHashSet;
import java.util.Scanner;

public class LinkedHashSetDemo {
    public static void main(String[] args) {
        LinkedHashSet<String> cities = new LinkedHashSet<>();
        Scanner sc = new Scanner(System.in);

        System.out.print("How many cities do you want to enter? ");
        int count = sc.nextInt();
        sc.nextLine(); // consume the leftover newline

        for (int i = 1; i <= count; i++) {
            System.out.print("Enter city " + i + ": ");
            String city = sc.nextLine();
            if (cities.add(city)) {
                System.out.println("Added: " + city);
            } else {
                System.out.println("Duplicate city! Not added.");
            }
```

```
        }

        System.out.println("\nCities in the order entered (no duplicates):");
        for (String city : cities) {
            System.out.println(city);
        }

        sc.close();
    }
}
```

4. Design a program to store registered email IDs of users such that no duplicates are allowed.

```
import java.util.HashSet;
import java.util.Scanner;

public class EmailRegistry {
    public static void main(String[] args) {
        HashSet<String> emailSet = new HashSet<>();
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of email IDs to register: ");
        int count = sc.nextInt();
        sc.nextLine();  // consume leftover newline

        for (int i = 1; i <= count; i++) {
            System.out.print("Enter email ID " + i + ": ");
            String email = sc.nextLine().toLowerCase(); // to treat emails case-
insensitively
            if (emailSet.add(email)) {
                System.out.println("Registered: " + email);
            } else {
                System.out.println("Duplicate email! Already registered.");
            }
        }

        System.out.println("\nRegistered Email IDs:");
        for (String email : emailSet) {
            System.out.println(email);
        }

        sc.close();
    }
}
```

5. Create a program where a Set is used to eliminate duplicate entries from a list of city names entered by users.

```
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;

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

      System.out.print("Enter the number of city names: ");
      int n = sc.nextInt();
      sc.nextLine(); // consume leftover newline

      for (int i = 1; i <= n; i++) {
         System.out.print("Enter city name " + i + ": ");
         String city = sc.nextLine().trim().toLowerCase(); // normalize input
         if (cities.add(city)) {
            System.out.println("Added: " + city);
         } else {
            System.out.println("Duplicate! City already exists.");
         }
      }

      System.out.println("\nUnique city names:");
      for (String city : cities) {
         System.out.println(city);
      }

      sc.close();
   }
}
```

Map Interface

1. Write a program using HashMap to store student names and their marks.

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

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

      System.out.print("Enter the number of students: ");
      int n = sc.nextInt();
      sc.nextLine(); // consume leftover newline

      for (int i = 1; i <= n; i++) {
         System.out.print("Enter name of student " + i + ": ");
         String name = sc.nextLine();
```

```
            System.out.print("Enter marks of " + name + ": ");
            int marks = sc.nextInt();
            sc.nextLine(); // consume leftover newline

            studentMap.put(name, marks);
        }

        System.out.println("\nStudent Marks:");
        for (Map.Entry<String, Integer> entry : studentMap.entrySet()) {
            System.out.println("Name: " + entry.getKey() + ", Marks: " +
entry.getValue());
        }

        sc.close();
    }
}
```

2. Demonstrate how to iterate over a Map using entrySet().

```
import java.util.HashMap;
import java.util.Map;

public class IterateMap {
    public static void main(String[] args) {
        // Creating a HashMap of country and capital
        Map<String, String> countryCapitalMap = new HashMap<>();

        // Adding entries to the map
        countryCapitalMap.put("India", "New Delhi");
        countryCapitalMap.put("USA", "Washington D.C.");
        countryCapitalMap.put("France", "Paris");
        countryCapitalMap.put("Japan", "Tokyo");

        // Iterating using entrySet()
        System.out.println("Country - Capital List:");
        for (Map.Entry<String, String> entry : countryCapitalMap.entrySet()) {
            String country = entry.getKey();
            String capital = entry.getValue();
            System.out.println(country + " → " + capital);
        }
    }
}
```

3. Show how to update the value associated with a key in a Map.

```
import java.util.HashMap;
import java.util.Map;

public class UpdateMapValue {
    public static void main(String[] args) {
        // Create a HashMap of student names and their marks
        Map<String, Integer> studentMarks = new HashMap<>();
```

```
        // Add initial values
        studentMarks.put("Alice", 75);
        studentMarks.put("Bob", 82);
        studentMarks.put("Charlie", 68);

        System.out.println("Before Update: " + studentMarks);

        // Update Bob's marks
        if (studentMarks.containsKey("Bob")) {
            studentMarks.put("Bob", 90);  // Overwrites old value
            System.out.println("Updated Bob's marks to 90");
        }

        // Attempt to update a key that doesn't exist
        if (studentMarks.containsKey("David")) {
            studentMarks.put("David", 70);
        } else {
            System.out.println("David not found in the map.");
        }

        System.out.println("After Update: " + studentMarks);
    }
}
```

4. **Build a phone directory where names are keys and phone numbers are values**

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class PhoneDirectory {
    public static void main(String[] args) {
        // Create a HashMap to store the phone directory
        Map<String, String> phoneDirectory = new HashMap<>();

        Scanner scanner = new Scanner(System.in);
        int choice;

        do {
            System.out.println("\n Phone Directory Menu:");
            System.out.println("1. Add Contact");
            System.out.println("2. View Contact");
            System.out.println("3. Display All Contacts");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
            scanner.nextLine(); // consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter Name: ");
                    String name = scanner.nextLine();
```

```
                System.out.print("Enter Phone Number: ");
                String phone = scanner.nextLine();
                phoneDirectory.put(name, phone);
                System.out.println("Contact added.");
                break;

            case 2:
                System.out.print("Enter Name to Search: ");
                String searchName = scanner.nextLine();
                if (phoneDirectory.containsKey(searchName)) {
                    System.out.println(searchName + "'s Phone Number: " +
phoneDirectory.get(searchName));
                } else {
                    System.out.println("Contact not found.");
                }
                break;

            case 3:
                System.out.println("\nAll Contacts:");
                for (Map.Entry<String, String> entry : phoneDirectory.entrySet()) {
                    System.out.println(entry.getKey() + " → " + entry.getValue());
                }
                break;

            case 4:
                System.out.println("Exiting Phone Directory.");
                break;

            default:
                System.out.println("Invalid choice. Try again.");
        }

    } while (choice != 4);

    scanner.close();
  }
}
```

5. Create a frequency counter for words in a sentence using a Map.

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

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

    // Step 1: Get input from user
    System.out.print("Enter a sentence: ");
    String sentence = scanner.nextLine();
```

```
        // Step 2: Split the sentence into words
        String[] words = sentence.toLowerCase().split("\\s+"); // Split by whitespace

        // Step 3: Create a HashMap to store word frequencies
        Map<String, Integer> wordCountMap = new HashMap<>();

        // Step 4: Loop through each word and count frequency
        for (String word : words) {
            if (wordCountMap.containsKey(word)) {
                // Word already exists, increment count
                wordCountMap.put(word, wordCountMap.get(word) + 1);
            } else {
                // Word is new, add with count 1
                wordCountMap.put(word, 1);
            }
        }

        // Step 5: Display word frequencies
        System.out.println("\nWord Frequencies:");
        for (Map.Entry<String, Integer> entry : wordCountMap.entrySet()) {
            System.out.println(entry.getKey() + " → " + entry.getValue());
        }

        scanner.close();
    }
}
```

Queue Interface

1. Implement a simple task queue using LinkedList as a Queue.

```
import java.util.LinkedList;
import java.util.Queue;

public class TaskQueue {
    public static void main(String[] args) {
        // Step 1: Create a LinkedList to use as a Queue
        Queue<String> taskQueue = new LinkedList<>();

        // Step 2: Add tasks to the queue (enqueue)
        taskQueue.add("Task 1: Email the client");
        taskQueue.add("Task 2: Review code");
        taskQueue.add("Task 3: Deploy application");

        // Step 3: Process tasks (dequeue)
        System.out.println("Processing Tasks in Queue Order:\n");
        while (!taskQueue.isEmpty()) {
            String task = taskQueue.poll(); // Retrieves and removes the head of the
queue
            System.out.println("Processing: " + task);
        }

        // Step 4: After processing all tasks
```

```
            System.out.println("\nAll tasks processed.");
        }
    }
```

2. Demonstrate how to add and remove elements using offer() and poll().

```
import java.util.LinkedList;
import java.util.Queue;

public class QueueExample {
    public static void main(String[] args) {
        // Create a queue using LinkedList
        Queue<String> queue = new LinkedList<>();

        // Adding elements using offer()
        queue.offer("Task A");
        queue.offer("Task B");
        queue.offer("Task C");

        System.out.println("Queue after offers: " + queue);

        // Removing elements using poll()
        String firstTask = queue.poll();  // removes "Task A"
        System.out.println("Polled: " + firstTask);
        System.out.println("Queue after first poll: " + queue);

        String secondTask = queue.poll(); // removes "Task B"
        System.out.println("Polled: " + secondTask);
        System.out.println("Queue after second poll: " + queue);
    }
}
```

3. Use a PriorityQueue to order tasks by priority (integers).

```
import java.util.PriorityQueue;

public class TaskPriorityQueue {
    public static void main(String[] args) {
        // Creating a PriorityQueue to store tasks with priorities (lower = higher
priority)
        PriorityQueue<Task> taskQueue = new PriorityQueue<>();

        // Adding tasks with different priorities
        taskQueue.add(new Task("Complete Java Assignment", 3));
        taskQueue.add(new Task("Pay Electricity Bill", 1));
        taskQueue.add(new Task("Grocery Shopping", 4));
        taskQueue.add(new Task("Call Mom", 2));

        // Poll tasks from the queue in order of priority
        System.out.println("Tasks in priority order:");
        while (!taskQueue.isEmpty()) {
            Task task = taskQueue.poll();
            System.out.println(task.name + " (Priority: " + task.priority + ")");
        }
```

```
    }
}

// Custom class representing a task
class Task implements Comparable<Task> {
    String name;
    int priority;

    // Constructor
    public Task(String name, int priority) {
        this.name = name;
        this.priority = priority;
    }

    // Compare tasks by priority (lower number = higher priority)
    public int compareTo(Task other) {
        return Integer.compare(this.priority, other.priority);
    }
}
```

4. Simulate a print queue system where print jobs are processed in order.

```
import java.util.LinkedList;
import java.util.Queue;

class PrintJob {
    private String documentName;

    public PrintJob(String documentName) {
        this.documentName = documentName;
    }

    public String getDocumentName() {
        return documentName;
    }
}

public class PrintQueueSimulation {
    public static void main(String[] args) {
        // Create a queue to hold print jobs
        Queue<PrintJob> printQueue = new LinkedList<>();

        // Add print jobs to the queue
        printQueue.offer(new PrintJob("Document1.pdf"));
        printQueue.offer(new PrintJob("Invoice_March.docx"));
        printQueue.offer(new PrintJob("Resume.pdf"));
        printQueue.offer(new PrintJob("Poster_Design.ppt"));

        System.out.println(" Print Queue Simulation Started...\n");

        // Process each job in the queue (FIFO order)
        while (!printQueue.isEmpty()) {
```

```
            PrintJob currentJob = printQueue.poll(); // remove and get head of queue
            System.out.println("Printing: " + currentJob.getDocumentName());
        }

        System.out.println("\nAll documents printed.");
    }
}
```

5. Create a ticket booking system where customer names are added to a queue and served in order.

```
import java.util.LinkedList;
import java.util.Queue;

public class TicketBookingSystem {
    public static void main(String[] args) {
        // Create a queue to store customer names
        Queue<String> customerQueue = new LinkedList<>();

        // Simulate customers booking tickets
        customerQueue.offer("Alice");
        customerQueue.offer("Bob");
        customerQueue.offer("Charlie");
        customerQueue.offer("Diana");

        System.out.println(" Ticket Booking System Started...\n");

        // Serve customers in the order they booked
        while (!customerQueue.isEmpty()) {
            String customer = customerQueue.poll();  // Retrieves and removes the
head of the queue
            System.out.println("Ticket issued to: " + customer);
        }

        System.out.println("\n All customers have been served.");
    }
}
```

Iterator Interface

1. Write a program to iterate through a list using Iterator.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class IteratorExample {
    public static void main(String[] args) {
        // Create a List and add some elements
        List<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Mango");
        fruits.add("Orange");
```

```
        // Create an Iterator for the list
        Iterator<String> iterator = fruits.iterator();

        // Use the Iterator to iterate through the list
        System.out.println("Fruits in the list:");
        while (iterator.hasNext()) {
            String fruit = iterator.next();
            System.out.println(fruit);
        }
    }
}
```

2. Demonstrate removing an element from a list while iterating using Iterator.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class RemoveWhileIterating {
    public static void main(String[] args) {
        // Create a List of names
        List<String> names = new ArrayList<>();
        names.add("Alice");
        names.add("Bob");
        names.add("Charlie");
        names.add("David");

        // Create an Iterator
        Iterator<String> iterator = names.iterator();

        // Use iterator to remove "Charlie" while looping
        while (iterator.hasNext()) {
            String name = iterator.next();
            if (name.equals("Charlie")) {
                iterator.remove();  // Safe removal
            }
        }

        // Step 4: Print updated list
        System.out.println("Updated list after removal: " + names);
    }
}
```

3. Show how to use ListIterator to iterate in both directions.

```
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;

public class ListIteratorExample {
    public static void main(String[] args) {
        // Step 1: Create and populate a list
        List<String> fruits = new ArrayList<>();
        fruits.add("Apple");
```

```
        fruits.add("Banana");
        fruits.add("Mango");
        fruits.add("Orange");

        // Step 2: Create a ListIterator
        ListIterator<String> listIterator = fruits.listIterator();

        System.out.println("Forward Direction:");
        // Step 3: Forward iteration
        while (listIterator.hasNext()) {
            String fruit = listIterator.next();
            System.out.println(fruit);
        }

        System.out.println("\nBackward Direction:");
        // Step 4: Backward iteration
        while (listIterator.hasPrevious()) {
            String fruit = listIterator.previous();
            System.out.println(fruit);
        }
    }
}
```

4. Design a program that reads a list of book titles and removes those starting with a specific letter using an iterator.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class RemoveBooksByLetter {
    public static void main(String[] args) {
        // Step 1: Create and populate a list of book titles
        List<String> books = new ArrayList<>();
        books.add("The Hobbit");
        books.add("To Kill a Mockingbird");
        books.add("Moby Dick");
        books.add("Twilight");
        books.add("Harry Potter");

        // Define the letter to remove books by
        char targetLetter = 'T';

        // Step 2: Use Iterator to remove titles starting with the target letter
        Iterator<String> iterator = books.iterator();
        while (iterator.hasNext()) {
            String book = iterator.next();
            if (book.startsWith(String.valueOf(targetLetter))) {
                iterator.remove();  // Safe removal during iteration
            }
        }
```

```
        // Step 3: Print remaining books
        System.out.println("Books after removing those starting with '" +
    targetLetter + "':");
        for (String title : books) {
            System.out.println(title);
        }
    }
}
```

5. Create a program that reverses the elements in a list using ListIterator.

```
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;

public class ReverseListWithListIterator {
    public static void main(String[] args) {
        // Step 1: Create and populate the list
        List<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");
        fruits.add("Date");

        // Step 2: Create a ListIterator starting at the end of the list
        ListIterator<String> listIterator = fruits.listIterator(fruits.size());

        // Step 3: Traverse the list in reverse using hasPrevious() and previous()
        System.out.println("Fruits in reverse order:");
        while (listIterator.hasPrevious()) {
            System.out.println(listIterator.previous());
        }
    }
}
```

Sorting and Searching Collections

1. Sort an ArrayList of integers in ascending and descending order.

```
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;

public class ReverseListWithListIterator {
    public static void main(String[] args) {
        // Step 1: Create and populate the list
        List<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");
        fruits.add("Date");

        // Step 2: Create a ListIterator starting at the end of the list
        ListIterator<String> listIterator = fruits.listIterator(fruits.size());
```

```
        // Step 3: Traverse the list in reverse using hasPrevious() and
    previous()
        System.out.println("Fruits in reverse order:");
        while (listIterator.hasPrevious()) {
            System.out.println(listIterator.previous());
        }
    }
}
```

2. Use Collections.binarySearch() to find an element in a sorted list.

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

public class SortArrayListExample {
    public static void main(String[] args) {
        // Step 1: Create and populate the ArrayList
        ArrayList<Integer> numbers = new ArrayList<>();
        numbers.add(15);
        numbers.add(3);
        numbers.add(27);
        numbers.add(10);
        numbers.add(6);

        // Step 2: Sort in Ascending Order
        Collections.sort(numbers);
        System.out.println("Ascending Order: " + numbers);

        // Step 3: Sort in Descending Order
        Collections.sort(numbers, Collections.reverseOrder());
        System.out.println("Descending Order: " + numbers);
    }
}
```

3. Sort a list of custom objects like Employees by name using Comparator.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

// Employee class
class Employee {
    int id;
    String name;
    double salary;

    // Constructor
    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
```

```
    // Display method
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}

// Comparator to sort by name
class NameComparator implements Comparator<Employee> {
    public int compare(Employee e1, Employee e2) {
        return e1.name.compareToIgnoreCase(e2.name);
    }
}

// Main class
public class SortEmployeesByName {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();

        // Add employee objects
        employees.add(new Employee(102, "Ravi", 45000));
        employees.add(new Employee(101, "Anjali", 50000));
        employees.add(new Employee(103, "Meena", 42000));

        System.out.println("Before Sorting:");
        for (Employee e : employees) {
            System.out.println(e);
        }

        // Sort using Comparator
        Collections.sort(employees, new NameComparator());

        System.out.println("\nAfter Sorting by Name:");
        for (Employee e : employees) {
            System.out.println(e);
        }
    }
}
```

4. You have a list of products with prices. Sort them by price and then search for a product within a specific price range.

```
import java.util.*;

// Product class
class Product {
    String name;
    double price;

    // Constructor
    public Product(String name, double price) {
        this.name = name;
        this.price = price;
```

```
    }

    // Display format
    public String toString() {
        return name + " - ₹" + price;
    }
}

// Comparator to sort by price
class PriceComparator implements Comparator<Product> {
    public int compare(Product p1, Product p2) {
        return Double.compare(p1.price, p2.price);
    }
}

public class ProductSearchByPrice {
    public static void main(String[] args) {
        List<Product> products = new ArrayList<>();

        // Add products
        products.add(new Product("Mouse", 299.99));
        products.add(new Product("Keyboard", 499.50));
        products.add(new Product("Monitor", 8999.00));
        products.add(new Product("USB Cable", 149.75));
        products.add(new Product("Charger", 349.00));

        // Sort products by price
        Collections.sort(products, new PriceComparator());

        System.out.println("Sorted Products by Price:");
        for (Product p : products) {
            System.out.println(p);
        }

        // Define price range
        double minPrice = 200;
        double maxPrice = 500;

        System.out.println("\nProducts in price range ₹" + minPrice + " - ₹" +
maxPrice + ":");
        for (Product p : products) {
            if (p.price >= minPrice && p.price <= maxPrice) {
                System.out.println(p);
            }
        }
    }
}
```

5. **Build a leaderboard system that keeps players sorted by scores (highest first). Allow searching for a specific player's rank**

```
import java.util.*;

// Player class
class Player {
    String name;
    int score;

    public Player(String name, int score) {
        this.name = name;
        this.score = score;
    }

    public String toString() {
        return name + " - " + score;
    }
}

// Comparator to sort by score (highest first)
class ScoreComparator implements Comparator<Player> {
    public int compare(Player p1, Player p2) {
        return Integer.compare(p2.score, p1.score); // descending
    }
}

public class LeaderboardSystem {
    public static void main(String[] args) {
        List<Player> leaderboard = new ArrayList<>();

        // Add players
        leaderboard.add(new Player("Alice", 1200));
        leaderboard.add(new Player("Bob", 1500));
        leaderboard.add(new Player("Charlie", 1100));
        leaderboard.add(new Player("Diana", 1800));
        leaderboard.add(new Player("Ethan", 1500));

        // Sort leaderboard by score (descending)
        Collections.sort(leaderboard, new ScoreComparator());

        // Display leaderboard
        System.out.println(" Leaderboard:");
        int rank = 1;
        for (Player p : leaderboard) {
            System.out.println(rank + ". " + p);
            rank++;
        }

        // Search for a specific player's rank
        String searchName = "Bob";
        System.out.println("\n Searching rank for player: " + searchName);
        boolean found = false;
```

```
        for (int i = 0; i < leaderboard.size(); i++) {
            if (leaderboard.get(i).name.equalsIgnoreCase(searchName)) {
                System.out.println(searchName + "'s Rank: " + (i + 1));
                found = true;
                break;
            }
        }

        if (!found) {
            System.out.println("Player '" + searchName + "' not found in the
leaderboard.");
        }
    }
}
```