**1. Introduction to Collections Framework**

**Direct Questions**

**1. Adding and printing elements from an ArrayList**

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

public class ArrayListDemo {
    public static void main(String[] args) {
        // Create an ArrayList to store elements
        List<String> list = new ArrayList<>();

        // Adding elements to the ArrayList
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");

        // Print the elements of the ArrayList
        System.out.println("ArrayList elements: " + list);
    }
}
```

**2. Using Collections.max() and Collections.min() on a list of integers**

```
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class MaxMinDemo {
```

```
    public static void main(String[] args) {
        // Create a list of integers
        List<Integer> numbers = Arrays.asList(3, 1, 4, 1, 5, 9, 2, 6, 5);

        // Find and print the maximum and minimum values in the list
        System.out.println("Maximum: " + Collections.max(numbers));
        System.out.println("Minimum: " + Collections.min(numbers));
    }
}
```

**3. Using Collections.sort() on a list of strings**

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

public class SortDemo {
    public static void main(String[] args) {
        // Create an ArrayList of strings
        List<String> fruits = new ArrayList<>();
        fruits.add("Orange");
        fruits.add("Apple");
        fruits.add("Banana");

        // Sort the list using Collections.sort()
        Collections.sort(fruits);

        // Print the sorted list
        System.out.println("Sorted list: " + fruits);
```

```
    }
}
```

**Scenario-Based Questions**

**4. Storing and displaying student names in alphabetical order**

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

public class StudentNames {
    public static void main(String[] args) {
        // Create a list to store student names
        List<String> studentNames = new ArrayList<>();
        studentNames.add("John");
        studentNames.add("Alice");
        studentNames.add("Bob");

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

        // Print the sorted list of student names
        System.out.println("Student names in alphabetical order: " + studentNames);
    }
}
```

**5. Storing and displaying the sum of user-input integers**

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

```
import java.util.Scanner;

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

        // Prompt the user to enter integers
        System.out.println("Enter integers (enter a non-integer to stop):");
        while (scanner.hasNextInt()) {
            numbers.add(scanner.nextInt());
        }

        // Calculate the sum of the integers
        int sum = numbers.stream().mapToInt(Integer::intValue).sum();

        // Print the sum of the elements
        System.out.println("Sum of all elements: " + sum);
    }
}
```

---

**2. List Interface**

**Direct Questions**

**1. Adding, removing, and accessing elements in an ArrayList**

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

```
public class ArrayListOperations {
    public static void main(String[] args) {
        // Create an ArrayList
        List<String> list = new ArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");

        // Print the initial list
        System.out.println("Initial list: " + list);

        // Remove an element from the list
        list.remove("Banana");
        System.out.println("After removing 'Banana': " + list);

        // Access an element at a specific index
        System.out.println("Element at index 0: " + list.get(0));
    }
}
```

**2. Implementing a LinkedList to store and print employee names**

```
import java.util.LinkedList;
import java.util.List;

public class LinkedListDemo {
    public static void main(String[] args) {
        // Create a LinkedList to store employee names
        List<String> employeeNames = new LinkedList<>();
```

```
        employeeNames.add("John Doe");
        employeeNames.add("Jane Smith");
        employeeNames.add("Jim Brown");

        // Print the employee names
        System.out.println("Employee names: " + employeeNames);
    }
}
```

**3. 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) {
        // Create an ArrayList
        List<String> list = new ArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");

        // Insert an element at a specific position
        list.add(1, "Blueberry");

        // Print the list after insertion
        System.out.println("List after insertion: " + list);
    }
}
```

**Scenario-Based Questions**

**4. Building a to-do list manager**

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

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

        // Menu-driven loop to manage the to-do list
        while (true) {
            System.out.println("1. Add task\n2. Remove task\n3. Display tasks\n4. Exit");
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    System.out.println("Enter task to add:");
                    tasks.add(scanner.nextLine());
                    break;
                case 2:
                    System.out.println("Enter task to remove:");
                    tasks.remove(scanner.nextLine());
                    break;
                case 3:
```

```
                System.out.println("Pending tasks: " + tasks);
                break;
            case 4:
                System.exit(0);
        }
    }
  }
}
```

**5. Creating a simple shopping cart system**

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

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

      // Menu-driven loop to manage the shopping cart
      while (true) {
         System.out.println("1. Add product\n2. Remove product\n3. Display cart\n4. Exit");
         int choice = scanner.nextInt();
         scanner.nextLine(); // Consume newline

         switch (choice) {
            case 1:
               System.out.println("Enter product to add:");
```

```
                cart.add(scanner.nextLine());
                break;
            case 2:
                System.out.println("Enter product to remove:");
                cart.remove(scanner.nextLine());
                break;
            case 3:
                System.out.println("Shopping cart: " + cart);
                break;
            case 4:
                System.exit(0);
        }
    }
  }
}
```

---

**3. Set Interface**

**Direct Questions**

**1. Using HashSet to store unique student roll numbers**

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

public class UniqueRollNumbers {
   public static void main(String[] args) {
      // Create a HashSet to store unique roll numbers
      Set<Integer> rollNumbers = new HashSet<>();
      rollNumbers.add(101);
```

```
        rollNumbers.add(102);
        rollNumbers.add(103);
        rollNumbers.add(101); // Duplicate

        // Print the unique roll numbers
        System.out.println("Unique roll numbers: " + rollNumbers);
    }
}
```

**2. Using TreeSet to automatically sort elements**

```
import java.util.Set;
import java.util.TreeSet;

public class SortedElements {
    public static void main(String[] args) {
        // Create a TreeSet to automatically sort elements
        Set<String> sortedSet = new TreeSet<>();
        sortedSet.add("Banana");
        sortedSet.add("Apple");
        sortedSet.add("Cherry");

        // Print the sorted elements
        System.out.println("Sorted elements: " + sortedSet);
    }
}
```

**3. Using LinkedHashSet to maintain insertion order and prevent duplicates**

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

```
import java.util.Set;

public class InsertionOrderSet {
    public static void main(String[] args) {
        // Create a LinkedHashSet to maintain insertion order and prevent duplicates
        Set<String> set = new LinkedHashSet<>();
        set.add("Apple");
        set.add("Banana");
        set.add("Cherry");
        set.add("Apple"); // Duplicate

        // Print the elements in insertion order
        System.out.println("Elements in insertion order: " + set);
    }
}
```

**Scenario-Based Questions**

**4. Storing registered email IDs without duplicates**

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

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

        // Prompt the user to enter email IDs
```

```
        System.out.println("Enter email IDs (enter 'exit' to stop):");
        while (true) {
            String email = scanner.nextLine();
            if (email.equals("exit")) {
                break;
            }
            emailIds.add(email);
        }

        // Print the registered email IDs
        System.out.println("Registered email IDs: " + emailIds);
    }
}
```

**5. Eliminating duplicate entries from a list of city names**

```
import java.util.ArrayList;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Scanner;
import java.util.Set;

public class UniqueCities {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<String> cities = new ArrayList<>();

        // Prompt the user to enter city names
        System.out.println("Enter city names (enter 'exit' to stop):");
```

```
        while (true) {
            String city = scanner.nextLine();
            if (city.equals("exit")) {
                break;
            }
            cities.add(city);
        }

        // Use LinkedHashSet to eliminate duplicates and maintain insertion order
        Set<String> uniqueCities = new LinkedHashSet<>(cities);

        // Print the unique city names
        System.out.println("Unique city names: " + uniqueCities);
    }
}
```

---

**4. Map Interface**

**Direct Questions**

**1. Using HashMap to store student names and their marks**

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

public class StudentMarks {
    public static void main(String[] args) {
        // Create a HashMap to store student names and their marks
        Map<String, Integer> studentMarks = new HashMap<>();
        studentMarks.put("John", 85);
```

```
        studentMarks.put("Alice", 90);
        studentMarks.put("Bob", 78);

        // Print the student marks
        System.out.println("Student marks: " + studentMarks);
    }
}
```

**2. Iterating over a Map using entrySet()**

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

public class IterateMap {
    public static void main(String[] args) {
        // Create a HashMap
        Map<String, Integer> map = new HashMap<>();
        map.put("Apple", 1);
        map.put("Banana", 2);
        map.put("Cherry", 3);

        // Iterate over the Map using entrySet()
        for (Map.Entry<String, Integer> entry : map.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
    }
}
```

**3. Updating 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
        Map<String, Integer> map = new HashMap<>();
        map.put("Apple", 1);
        map.put("Banana", 2);

        // Update the value associated with a key
        map.put("Apple", 3);

        // Print the updated map
        System.out.println("Updated map: " + map);
    }
}
```

**Scenario-Based Questions**

**4. Building a phone directory**

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

public class PhoneDirectory {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Map<String, String> directory = new HashMap<>();
```

```
        // Menu-driven loop to manage the phone directory
        while (true) {
            System.out.println("1. Add contact\n2. Search contact\n3. Exit");
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    System.out.println("Enter name:");
                    String name = scanner.nextLine();
                    System.out.println("Enter phone number:");
                    String phoneNumber = scanner.nextLine();
                    directory.put(name, phoneNumber);
                    break;
                case 2:
                    System.out.println("Enter name to search:");
                    String searchName = scanner.nextLine();
                    System.out.println("Phone number: " + directory.get(searchName));
                    break;
                case 3:
                    System.exit(0);
            }
        }
    }
}
```

**5. Creating a frequency counter for words in a sentence**

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

public class WordFrequencyCounter {
    public static void main(String[] args) {
        String sentence = "This is a test. This is only a test.";
        String[] words = sentence.split(" ");

        // Create a HashMap to store word frequencies
        Map<String, Integer> frequencyMap = new HashMap<>();
        for (String word : words) {
            frequencyMap.put(word, frequencyMap.getOrDefault(word, 0) + 1);
        }

        // Print the word frequencies
        System.out.println("Word frequencies: " + frequencyMap);
    }
}
```

---

**5. Queue Interface**

**Direct Questions**

**1. Implementing 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) {
```

```
        // Create a Queue using LinkedList
        Queue<String> taskQueue = new LinkedList<>();
        taskQueue.offer("Task 1");
        taskQueue.offer("Task 2");
        taskQueue.offer("Task 3");

        // Print the task queue
        System.out.println("Task queue: " + taskQueue);
    }
}
```

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

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

public class QueueOperations {
    public static void main(String[] args) {
        // Create a Queue using LinkedList
        Queue<String> queue = new LinkedList<>();
        queue.offer("Element 1");
        queue.offer("Element 2");

        // Print the queue after adding elements
        System.out.println("Queue after adding elements: " + queue);

        // Remove an element from the queue
        System.out.println("Removed element: " + queue.poll());
```

```
        // Print the queue after removal
        System.out.println("Queue after removal: " + queue);
    }
}
```

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

```
import java.util.PriorityQueue;
import java.util.Queue;

public class PriorityTaskQueue {
    public static void main(String[] args) {
        // Create a PriorityQueue to order tasks by priority
        Queue<Integer> priorityQueue = new PriorityQueue<>();
        priorityQueue.offer(3);
        priorityQueue.offer(1);
        priorityQueue.offer(2);

        // Print the priority queue
        System.out.println("Priority queue: " + priorityQueue);
    }
}
```

**Scenario-Based Questions**

**4. Simulating a print queue system**

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

public class PrintQueueSystem {
```

```
    public static void main(String[] args) {
        // Create a Queue to simulate a print queue
        Queue<String> printQueue = new LinkedList<>();
        printQueue.offer("Print Job 1");
        printQueue.offer("Print Job 2");
        printQueue.offer("Print Job 3");

        // Process the print jobs in order
        while (!printQueue.isEmpty()) {
            System.out.println("Processing: " + printQueue.poll());
        }
    }
}
```

**5. Creating a ticket booking system**

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

public class TicketBookingSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Queue<String> ticketQueue = new LinkedList<>();

        // Menu-driven loop to manage the ticket booking system
        while (true) {
            System.out.println("1. Book ticket\n2. Serve next customer\n3. Exit");
            int choice = scanner.nextInt();
```

```
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    System.out.println("Enter customer name:");
                    ticketQueue.offer(scanner.nextLine());
                    break;
                case 2:
                    System.out.println("Serving: " + ticketQueue.poll());
                    break;
                case 3:
                    System.exit(0);
            }
        }
    }
}
```

---

**6. Iterator Interface**

**Direct Questions**

**1. Iterating through a list using Iterator**

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

public class IteratorDemo {
    public static void main(String[] args) {
        // Create an ArrayList
```

```
        List<String> list = new ArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");

        // Create an Iterator to iterate through the list
        Iterator<String> iterator = list.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

**2. 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 an ArrayList
        List<String> list = new ArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");

        // Create an Iterator and remove an element while iterating
        Iterator<String> iterator = list.iterator();
```

```
        while (iterator.hasNext()) {
            String element = iterator.next();
            if (element.equals("Banana")) {
                iterator.remove();
            }
        }

        // Print the list after removal
        System.out.println("List after removal: " + list);
    }
}
```

**3. Using ListIterator to iterate in both directions**

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

public class ListIteratorDemo {
    public static void main(String[] args) {
        // Create an ArrayList
        List<String> list = new ArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");

        // Create a ListIterator to iterate in both directions
        ListIterator<String> listIterator = list.listIterator();
        while (listIterator.hasNext()) {
```

```
            System.out.println(listIterator.next());
        }

        while (listIterator.hasPrevious()) {
            System.out.println(listIterator.previous());
        }
    }
}
```

**Scenario-Based Questions**

**4. Removing book titles 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) {
        // Create an ArrayList of book titles
        List<String> bookTitles = new ArrayList<>();
        bookTitles.add("Apple Book");
        bookTitles.add("Banana Book");
        bookTitles.add("Cherry Book");

        // Create an Iterator and remove book titles starting with a specific letter
        Iterator<String> iterator = bookTitles.iterator();
        while (iterator.hasNext()) {
            if (iterator.next().startsWith("A")) {
                iterator.remove();
```

```
            }
        }

        // Print the book titles after removal
        System.out.println("Book titles after removal: " + bookTitles);
    }
}
```

**5. Reversing the elements in a list using ListIterator**

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

public class ReverseList {
    public static void main(String[] args) {
        // Create an ArrayList
        List<String> list = new ArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");

        // Create a ListIterator to reverse the elements in the list
        ListIterator<String> listIterator = list.listIterator(list.size());
        while (listIterator.hasPrevious()) {
            System.out.println(listIterator.previous());
        }
    }
}
```

---

**7. Sorting and Searching Collections**

**Direct Questions**

**1. Sorting an ArrayList of integers in ascending and descending order**

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

public class SortIntegers {
    public static void main(String[] args) {
        // Create an ArrayList of integers
        List<Integer> numbers = new ArrayList<>();
        numbers.add(3);
        numbers.add(1);
        numbers.add(4);
        numbers.add(2);

        // Sort the list in ascending order
        Collections.sort(numbers);
        System.out.println("Ascending order: " + numbers);

        // Sort the list in descending order
        Collections.sort(numbers, Collections.reverseOrder());
        System.out.println("Descending order: " + numbers);
    }
}
```

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

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

public class BinarySearchDemo {
    public static void main(String[] args) {
        // Create a sorted list of integers
        List<Integer> numbers = new ArrayList<>();
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        numbers.add(4);

        // Use binary search to find an element in the list
        int index = Collections.binarySearch(numbers, 3);
        System.out.println("Index of element 3: " + index);
    }
}
```

**3. Sorting 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;

class Employee {
    String name;
```

```
    int id;

    public Employee(String name, int id) {
        this.name = name;
        this.id = id;
    }

    @Override
    public String toString() {
        return name + " (ID: " + id + ")";
    }
}

public class SortEmployees {
    public static void main(String[] args) {
        // Create a list of Employee objects
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee("John", 102));
        employees.add(new Employee("Alice", 101));
        employees.add(new Employee("Bob", 103));

        // Sort the list of employees by name using Comparator
        Collections.sort(employees, Comparator.comparing(employee -> employee.name));

        // Print the sorted list of employees
        System.out.println("Sorted employees: " + employees);
    }
}
```

**Scenario-Based Questions**

**4. Sorting products by price and searching for a product within a specific price range**

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

class Product {
    String name;
    double price;

    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    @Override
    public String toString() {
        return name + " ($" + price + ")";
    }
}

public class ProductManager {
    public static void main(String[] args) {
        // Create a list of Product objects
        List<Product> products = new ArrayList<>();
        products.add(new Product("Laptop", 999.99));
```

```
        products.add(new Product("Phone", 499.99));
        products.add(new Product("Tablet", 299.99));

        // Sort the list of products by price
        Collections.sort(products, Comparator.comparing(product -> product.price));

        // Print the sorted list of products
        System.out.println("Products sorted by price: " + products);

        // Define a price range
        double minPrice = 300.00;
        double maxPrice = 700.00;

        // Search for products within the specified price range
        System.out.println("Products within price range $" + minPrice + " to $" + maxPrice +
":");
        for (Product product : products) {
            if (product.price >= minPrice && product.price <= maxPrice) {
                System.out.println(product);
            }
        }
    }
}
```

**5. Building a leaderboard system that keeps players sorted by scores**

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

```
import java.util.List;

class Player {
    String name;
    int score;

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

    @Override
    public String toString() {
        return name + " (Score: " + score + ")";
    }
}

public class Leaderboard {
    public static void main(String[] args) {
        // Create a list of Player objects
        List<Player> players = new ArrayList<>();
        players.add(new Player("Alice", 150));
        players.add(new Player("Bob", 200));
        players.add(new Player("Charlie", 100));

        // Sort the list of players by score in descending order
        Collections.sort(players, Comparator.comparing(player -> -player.score));
```

```
        // Print the leaderboard
        System.out.println("Leaderboard: " + players);

        // Search for a specific player's rank
        String playerName = "Bob";
        int rank = 1;
        for (Player player : players) {
            if (player.name.equals(playerName)) {
                System.out.println(playerName + "'s rank: " + rank);
                break;
            }
            rank++;
        }
    }
}
```