Map Interface

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

```
import java.util.HashMap;
public class StudentMarks {
    public static void main(String[] args) {
        HashMap<String, Integer> marks = new HashMap<>();
        // Adding student names and marks
        marks.put("Alice", 85);
        marks.put("Bob", 78);
        marks.put("Charlie", 92);
        // Printing the map
        System.out.println("Student Marks:");
        for (String name : marks.keySet()) {
            System.out.println(name + ": " + marks.get(name));
        }
    }
}
```

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) {
        HashMap<String, Integer> scores = new HashMap<>();
        scores.put("Math", 90);
        scores.put("English", 85);
        scores.put("Science", 95);
        System.out.println("Subjects and Scores:");
        for (Map.Entry<String, Integer> entry : scores.entrySet()) {
            System.out.println(entry.getKey() + " => " + entry.getValue());
        } } }
```

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

```
import java.util.HashMap;
public class UpdateMapValue {
    public static void main(String[] args) {
        HashMap<String, Integer> stock = new HashMap<>();
        stock.put("Apples", 50);
        stock.put("Bananas", 30);

        // Update stock for Apples
        stock.put("Apples", 70); // Overwrites 50

        System.out.println("Updated Stock:");
        for (String item : stock.keySet()) {
            System.out.println(item + ": " + stock.get(item));
        }
    }
}
```

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

```
import java.util.HashMap;
import java.util.Scanner;
public class PhoneDirectory {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        HashMap<String, String> phoneBook = new HashMap<>();
        System.out.println("Enter 3 contacts:");
        for (int i = 0; i < 3; i++) {
            System.out.print("Enter name: ");
            String name = sc.nextLine();
            System.out.print("Enter phone number: ");
            String phone = sc.nextLine();
            phoneBook.put(name, phone);
        }
        System.out.println("\nPhone Directory:");
        for (String key : phoneBook.keySet()) {
            System.out.println(key + " => " + phoneBook.get(key));
        }
    }
}
```

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

```
import java.util.HashMap;
import java.util.Scanner;
public class WordFrequency {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a sentence: ");
        String sentence = sc.nextLine();
        String[] words = sentence.split(" ");
        HashMap<String, Integer> freqMap = new HashMap<>();

        for (String word : words) {
            word = word.toLowerCase(); // Case-insensitive
            freqMap.put(word, freqMap.getOrDefault(word, 0) + 1);
        }

        System.out.println("Word Frequencies:");
        for (String w : freqMap.keySet()) {
            System.out.println(w + " = " + freqMap.get(w));
        }
    }
}
```

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) {

        Queue<String> tasks = new LinkedList<>();

        tasks.offer("Task 1");         // Add tasks to the queue

        tasks.offer("Task 2");

        tasks.offer("Task 3");

        System.out.println("Processing Tasks:");

        while (!tasks.isEmpty()) {

            String task = tasks.poll();  // Removes the head of the queue

            System.out.println("Processing: " + task);

        }

    }

}
```

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

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

public class OfferPollDemo {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();

        queue.offer("Element A");
        queue.offer("Element B");

        System.out.println("Removed: " + queue.poll());  // Element A
        System.out.println("Removed: " + queue.poll());  // Element B
        System.out.println("Removed: " + queue.poll());  // null (queue is empty)
    }
}
```

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

```
import java.util.PriorityQueue;

public class PriorityTaskQueue {
    public static void main(String[] args) {
        PriorityQueue<Integer> taskQueue = new PriorityQueue<>();

        taskQueue.offer(5);  // Low priority
        taskQueue.offer(1);  // High priority
        taskQueue.offer(3);

        System.out.println("Processing tasks based on priority:");
        while (!taskQueue.isEmpty()) {
            System.out.println("Processing task with priority: " + taskQueue.poll());
        }
    }
}
```

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

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

public class PrintQueue {
    public static void main(String[] args) {
        Queue<String> printJobs = new LinkedList<>();

        printJobs.offer("Document1.pdf");
        printJobs.offer("Image.png");
        printJobs.offer("Resume.docx");

        System.out.println("Printing documents:");
        while (!printJobs.isEmpty()) {
            System.out.println("Printing: " + printJobs.poll());
        }
    }
}
```

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;
import java.util.Scanner;

public class TicketBooking {
    public static void main(String[] args) {
        Queue<String> customers = new LinkedList<>();
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter 3 customer names for ticket booking:");
        for (int i = 0; i < 3; i++) {
            System.out.print("Customer Name: ");
            String name = sc.nextLine();
            customers.offer(name);
        }

        System.out.println("\nServing Customers:");
        while (!customers.isEmpty()) {
            System.out.println("Serving: " + customers.poll());
        }
    }
}
```

Iterator Interface

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

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

public class IteratorExample {
    public static void main(String[] args) {
        ArrayList<String> names = new ArrayList<>();
        names.add("Ram");
        names.add("Sita");
        names.add("Laxman");

        Iterator<String> it = names.iterator();

        System.out.println("Names in list:");
        while (it.hasNext()) {
            System.out.println(it.next());
        }
    }
}
```

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

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

public class RemoveUsingIterator {
    public static void main(String[] args) {
        ArrayList<String> items = new ArrayList<>();
        items.add("Apple");
        items.add("Banana");
        items.add("Mango");

        Iterator<String> it = items.iterator();

        while (it.hasNext()) {
            String fruit = it.next();
            if (fruit.equals("Banana")) {
                it.remove();  // Safely removes "Banana"
            }
        }

        System.out.println("After removal: " + items);
    }
}
```

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

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

public class ListIteratorDemo {
    public static void main(String[] args) {
        ArrayList<String> colors = new ArrayList<>();
        colors.add("Red");
        colors.add("Green");
        colors.add("Blue");

        ListIterator<String> it = colors.listIterator();

        System.out.println("Forward:");
        while (it.hasNext()) {
            System.out.println(it.next());
        }

        System.out.println("Backward:");
        while (it.hasPrevious()) {
            System.out.println(it.previous());
        }
    }
}
```

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.Scanner;

public class BookFilter {
    public static void main(String[] args) {
        ArrayList<String> books = new ArrayList<>();
        books.add("Ramayana");
        books.add("Mahabharata");
        books.add("Romeo and Juliet");
        books.add("Cinderella");

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter starting letter to remove: ");
        char ch = sc.next().charAt(0);
        Iterator<String> it = books.iterator();
        while (it.hasNext()) {
            String title = it.next();
            if (title.toLowerCase().startsWith(Character.toString(ch).toLowerCase())) {
                it.remove();
            }
        }
        System.out.println("Remaining Books: " + books);
    }
}
```

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

```
import java.util.ArrayList;

import java.util.ListIterator;

public class ReverseList {

   public static void main(String[] args) {

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

      cities.add("Delhi");

      cities.add("Mumbai");

      cities.add("Chennai");

      ListIterator<String> it = cities.listIterator(cities.size());


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

      while (it.hasPrevious()) {

         System.out.println(it.previous());

      }

   }

}
```

Sorting and Searching Collections

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

```
import java.util.*;

public class SortIntegers {
   public static void main(String[] args) {
      ArrayList<Integer> numbers = new ArrayList<>(Arrays.asList(45, 12, 78, 23, 56));

      System.out.println("Original: " + numbers);

      // Ascending
      Collections.sort(numbers);
      System.out.println("Ascending: " + numbers);

      // Descending
      Collections.sort(numbers, Collections.reverseOrder());
      System.out.println("Descending: " + numbers);
   }
}
```

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

```
import java.util.*;

public class BinarySearchExample {
   public static void main(String[] args) {
      List<Integer> nums = new ArrayList<>(Arrays.asList(10, 20, 30, 40, 50));
      Collections.sort(nums); // Required before binarySearch

      int index = Collections.binarySearch(nums, 30);
      System.out.println("Element 30 found at index: " + index);
   }
}
```

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

```
import java.util.*;
class Employee {
   String name;
   int id;
   Employee(String name, int id) {
      this.name = name;
      this.id = id;
   }

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

public class SortEmployees {
   public static void main(String[] args) {
      List<Employee> employees = new ArrayList<>();
      employees.add(new Employee("John", 3));
      employees.add(new Employee("Alice", 1));
      employees.add(new Employee("Bob", 2));

      // Sort by name
      employees.sort(Comparator.comparing(e -> e.name));

      System.out.println("Sorted 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.*;

class Product {
    String name;
    double price;

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

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

public class ProductSorting {
    public static void main(String[] args) {
        List<Product> products = new ArrayList<>();
        products.add(new Product("Mouse", 599.0));
        products.add(new Product("Keyboard", 999.0));
        products.add(new Product("Monitor", 7999.0));
        products.add(new Product("Webcam", 1999.0));

        // Sort by price
        products.sort(Comparator.comparingDouble(p -> p.price));
        System.out.println("Sorted Products by Price:");
        for (Product p : products) {
            System.out.println(p);
        }

        // Search for products between ₹1000 and ₹3000
        System.out.println("\nProducts between ₹1000 and ₹3000:");
        for (Product p : products) {
            if (p.price >= 1000 && p.price <= 3000) {
                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.*;

class Player {
    String name;
    int score;

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

    public String toString() {
        return name + ": " + score;
    }
}
public class Leaderboard {
    public static void main(String[] args) {
        List<Player> players = new ArrayList<>();
        players.add(new Player("Alice", 90));
        players.add(new Player("Bob", 75));
        players.add(new Player("John", 88));
        players.add(new Player("Eve", 92));

        players.sort((a, b) -> b.score - a.score);        // Sort by score descending

        System.out.println("Leaderboard:");
        int rank = 1;
        for (Player p : players) {
            System.out.println("Rank " + rank + " - " + p);
            rank++;
        }

        // Search for a specific player’s rank
        String searchName = "John";
        rank = 1;
        for (Player p : players) {
            if (p.name.equalsIgnoreCase(searchName)) {
                System.out.println("\n" + searchName + "'s rank is: " + rank);
                break;
            }
            rank++;
        }
    }
}
```