Course Code: CSE 2104
Course Title: Object-oriented Programming Lab
Lab Work 4

SUBMITTED TO :

Shakib Mahamud Dipto
CSE
lecturer
University of Liberal Arts

SUBMITTED BY :

Piyas sarkar
223014024
Section 4

Question-1:

Calculate the average of array numbers.

Code:

```java
package oopd4;

import java.util.Scanner;

public class AverageOfArrayNumbers {
    public static void main(String[] args) {
        int[] array = new int[5];
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter 5 values:");
        for (int i = 0; i < 5; i++) {
            System.out.print("Enter the value of index " + i + ": ");
            array[i] = scanner.nextInt();
        }

        int sum = 0;
        for (int i = 0; i < array.length; i++) {
            sum += array[i];
        }

        double average = (double) sum / array.length;
        System.out.println("Average of the array numbers: " + average);

        scanner.close();
    }
}

```

Discussion:

The code creates a one-dimensional array of integers with a size of five in the first problem. It reads five integer values from the user via a Scanner object and then places them in the array.

The code then loops through the array to get the total of its members. Following the total computation, the average is obtained by dividing the total by the array's length of five. The average of the entered values is shown on the console as a consequence of the outcome. To liberate the resources, the scanner is closed at the end.

Question 2:

Transpose of a matrix.

Code:

```java

package oopd4;

import java.util.Scanner;

public class TransposeMatrix {
    public static void main(String[] args) {
        int[][] array = new int[3][3];
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter 9 values for the matrix:");
        for (int i = 0; i < array.length; i++) {
            for (int j = 0; j < array[i].length; j++) {
                System.out.print("Enter the value of index [" + i + "][" + j + "]: ");
                array[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Original Matrix:");
        for (int i = 0; i < array.length; i++) {
            for (int j = 0; j < array[i].length; j++) {
                System.out.print(array[i][j] + " ");
            }
            System.out.println();
        }

        int[][] transposedMatrix = new int[array[0].length][array.length];
        for (int i = 0; i < array.length; i++) {
            for (int j = 0; j < array[i].length; j++) {
                transposedMatrix[j][i] = array[i][j];
            }
        }

        System.out.println("Transposed Matrix:");
        for (int i = 0; i < transposedMatrix.length; i++) {
            for (int j = 0; j < transposedMatrix[i].length; j++) {
                System.out.print(transposedMatrix[i][j] + " ");
            }
            System.out.println();
        }

        scanner.close();
    }
}

```

Discussion:

The second challenge is a two-dimensional array (matrix) size 3 by 3. The code asks the user to provide nine integer values to populate the matrix. After that, a structured printout of the original matrix is produced. The procedure starts a new matrix with reversed dimensions and iterates over the previous matrix, switching the rows and columns to determine the transpose of the matrix. Next, the transposed matrix is printed out, illustrating the change from rows to columns and vice versa. Understanding matrix operations, particularly transposition, is aided by this procedure.

Question 3:

Perform various string operations.

Code:

```java

package oopd4;

import java.util.Scanner;

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

        System.out.print("Enter a string: ");
        String inputString = scanner.nextLine();

        System.out.println("Length of the string: " + inputString.length());

        System.out.print("Enter a character to search in the string: ");
        char searchChar = scanner.next().charAt(0);
        int charCount = 0;
        for (int i = 0; i < inputString.length(); i++) {
            if (inputString.charAt(i) == searchChar) {
                charCount++;
            }
        }
        System.out.println("Number of occurrences of the character '" + searchChar + "': " + charCount);

        System.out.print("Enter a substring to search in the string: ");
        String searchSubstring = scanner.next();
        if (inputString.contains(searchSubstring)) {
            System.out.println("Substring found in the string.");
        } else {
            System.out.println("Substring not found in the string.");
        }

        System.out.print("Enter a character to replace in the string: ");
        char replaceChar = scanner.next().charAt(0);
        System.out.print("Enter a character to replace with: ");
        char replacementChar = scanner.next().charAt(0);
        String replacedString = inputString.replace(replaceChar, replacementChar);
        System.out.println("String after replacement: " + replacedString);

        System.out.print("Enter a substring to replace in the string: ");
        String replaceSubstring = scanner.next();
        System.out.print("Enter a substring to replace with: ");
        String replacementSubstring = scanner.next();
        String replacedSubstring = inputString.replace(replaceSubstring, replacementSubstring);
        System.out.println("String after substring replacement: " + replacedSubstring);

        System.out.println("Concatenated string: " + inputString.concat(replacedSubstring));

        System.out.println("Lowercase string: " + inputString.toLowerCase());

        System.out.println("Uppercase string: " + inputString.toUpperCase());

        System.out.print("Enter another string for comparison: ");
        String comparisonString = scanner.next();
        if (inputString.equals(comparisonString)) {
            System.out.println("The strings are equal.");
        } else {
            System.out.println("The strings are not equal.");
        }

        int comparisonResult = inputString.compareTo(comparisonString);
        if (comparisonResult == 0) {
            System.out.println("The strings are equal.");
        } else if (comparisonResult < 0) {
            System.out.println("The first string is lexicographically smaller than the second string.");
        } else {
            System.out.println("The first string is lexicographically greater than the second string.");
        }

        scanner.close();
    }
}


```

Discussion:

The third difficulty involves the code doing several string operations. It begins by reading a user-inputted string and displaying its length. The next step asks the user to search the input text for a character and a substring; the results display the number of occurrences and if the substring is there. The code also shows the updated string and supports character and substring substitutions. The text may also be converted to uppercase and lowercase, concatenated with another string, and compared lexicographically with another input string, among other operations. These procedures show off basic Java string manipulations.

Question-4:

Create a Book class with properties such as title, author, year, and genre. Implement a parameterized constructor to initialize 3 objects, store them in an array, and display them. Remove a particular object and then display the existing objects.

Code:

```java
package oopd4;

import java.util.Scanner;

public class BookClassManipulation {
    static class Book {
        private String title;
        private String author;
        private int year;
        private String genre;

        public Book(String title, String author, int year, String genre) {
            this.title = title;
            this.author = author;
            this.year = year;
            this.genre = genre;
        }

        public String getTitle() {
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        public String getAuthor() {
            return author;
        }

        public void setAuthor(String author) {
            this.author = author;
        }

        public int getYear() {
            return year;
        }

        public void setYear(int year) {
            this.year = year;
        }

        public String getGenre() {
            return genre;
        }

        public void setGenre(String genre) {
            this.genre = genre;
        }
    }

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

        Book[] books = new Book[3];

        for (int i = 0; i < 3; i++) {
            System.out.println("Enter details for Book " + (i + 1) + ":");
            System.out.print("Title: ");
            String title = scanner.nextLine();
            System.out.print("Author: ");
            String author = scanner.nextLine();
            System.out.print("Year: ");
            int year = scanner.nextInt();
            scanner.nextLine();
            System.out.print("Genre: ");
            String genre = scanner.nextLine();

            books[i] = new Book(title, author, year, genre);
        }

        System.out.println("Books:");
        for (int i = 0; i < 3; i++) {
            System.out.println("Book " + (i + 1) + ":");
            System.out.println("Title: " + books[i].getTitle());
            System.out.println("Author: " + books[i].getAuthor());
            System.out.println("Year: " + books[i].getYear());
            System.out.println("Genre: " + books[i].getGenre());
        }

        System.out.print("Enter the index of the book to remove: ");
        int indexToRemove = scanner.nextInt();
        scanner.nextLine();

        if (indexToRemove >= 0 && indexToRemove < 3) {
            for (int i = indexToRemove; i < 2; i++) {
                books[i] = books[i + 1];
            }
            books[2] = null;

            System.out.println("Books after removing Book " + (indexToRemove + 1) + ":");
            for (int i = 0; i < 3; i++) {
                if (books[i] != null) {
                    System.out.println("Book " + (i + 1) + ":");
                    System.out.println("Title: " + books[i].getTitle());
                    System.out.println("Author: " + books[i].getAuthor());
                    System.out.println("Year: " + books[i].getYear());
                    System.out.println("Genre: " + books[i].getGenre());
                }
            }
        } else {
            System.out.println("Invalid index!");
        }

        scanner.close();
    }
}
```

Discussion:

Creating a Book class with properties like title, author, year, and genre is the fourth challenge. Three books are requested to have their information entered by the user and then saved in an array of Book objects by the code. The user is prompted to enter the index of a book to delete when the book details are displayed. The code eliminates the selected book by moving the remaining books in the array and changing the last element to null. Next, the revised book list is printed, showcasing Java's object manipulation and array handling capabilities. This challenge demonstrates the usage of fundamental array operations and custom classes.

Question-5:

Create an ArrayList and a LinkedList of integers. Apply various methods such as add, size, get, set, remove, and sort.

Code:

```java
package oopd4;

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

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

        // ArrayList operations
        List<Integer> arrayList = new ArrayList<>();

        System.out.print("Enter the number of elements to add to the ArrayList: ");
        int numElements = scanner.nextInt();

        System.out.println("Enter " + numElements + " elements:");

        for (int i = 0; i < numElements; i++) {
            int element = scanner.nextInt();
            arrayList.add(element);
        }

        System.out.println("ArrayList size: " + arrayList.size());

        System.out.print("Enter the index of the element to remove: ");
        int indexToRemove = scanner.nextInt();
        if (indexToRemove >= 0 && indexToRemove < arrayList.size()) {
            arrayList.remove(indexToRemove);
            System.out.println("Element removed successfully.");
        } else {
            System.out.println("Invalid index. Element not removed.");
        }

        System.out.println("ArrayList elements:");
        for (int element : arrayList) {
            System.out.println(element);
        }

        scanner.close();
    }
}

```

Discussion:

The creation and management of ArrayList and LinkedList collections are the main topics of the fifth problem. The code asks the user to enter a certain amount of numbers after initially initializing an ArrayList. It shows how to add elements, obtain the size, get, set, and remove elements based on index, and sort the list, among other activities. We then apply the same procedures to a LinkedList. The code illustrates the versatility of both data structures and the range of

available Java list manipulation techniques. The dynamic nature of lists and their useful applications in managing data collections are highlighted by this problem.

Question-6(practice-1):

Write a program to sort an array using bubble sort, selection sort, and merge sort.

Code:

```java
package oopd4;



public class MultiplyMatrix {
    public static void main(String[] args) {

        // Scanner sc = new Scanner(System.in);
        // System.out.println("Enter the number of rows and columns of the matrix:");
        // int rows = sc.nextInt();
        // int cols = sc.nextInt();

        int[][] matrix1 = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        int[][] matrix2 = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};

        int[][] result = multiplyMatrices(matrix1, matrix2);

        System.out.println("Result:");
        printMatrix(result);

    }

    public static int[][] multiplyMatrices(int[][] matrix1, int[][] matrix2) {
        int rows1 = matrix1.length;
        int cols1 = matrix1[0].length;
        int cols2 = matrix2[0].length;

        int[][] result = new int[rows1][cols2];

        for (int i = 0; i < rows1; i++) {
            for (int j = 0; j < cols2; j++) {
                for (int k = 0; k < cols1; k++) {
                    result[i][j] += matrix1[i][k] * matrix2[k][j];
                }
            }
        }

        return result;
    }

    public static void printMatrix(int[][] matrix) {
        for (int[] row : matrix) {
            for (int num : row) {
                System.out.print(num + " ");
            }
            System.out.println();
        }
    }
}

```

Discussion:

The `MultiplyMatrix` Java program exemplifies matrix multiplication by computing the product of two 3x3 matrices, `matrix1` and `matrix2`. Using nested loops, the `multiplyMatrices` method calculates each element of the resulting matrix by iterating over rows and columns, leveraging the dot product approach. The `printMatrix` method then formats and displays the resulting matrix in a readable format. While currently hardcoded, the program could easily be adapted to accept user input for matrix dimensions and values, enhancing its versatility. Overall, this code showcases fundamental matrix operations in Java, emphasizing clarity and efficiency in computational tasks involving matrices.

Question-7(practice-2):

Write a program to multiply two matrices.

Code:

```java
package oopd4;

import java.util.Scanner;

public class Sortings {

// Bubble Sort =>
// Start with an unsorted array of elements.
// Iterate through the array from the first element to the second-to-last element using the outer loop.
// Within the outer loop, iterate through the array from the first element to the (n - i - 1) element using the inner loop. Here, 'n' represents the length of the array and 'i' represents the current iteration of the outer loop.
// Compare each pair of adjacent elements in the inner loop.
// If the elements are in the wrong order (the current element is greater than the next element), swap them.
// Repeat steps 3-5 until the inner loop completes.
// Repeat steps 2-6 until the outer loop completes.
// The array is now sorted in ascending order.


    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

//Selection Sort =>
// The selectionSort method takes an array of integers (arr) as input and performs the selection sort algorithm on it.

// It starts by initializing a variable n with the length of the array, which represents the number of elements in the array.

// The outer loop runs from i = 0 to n - 1. This loop selects the current minimum element and places it in the correct position.

// Inside the outer loop, we initialize a variable minIndex with the value of i. This variable keeps track of the index of the minimum element found so far.

// The inner loop runs from j = i + 1 to n. It compares each element in the unsorted part of the array with the current minimum element (arr[minIndex]). If a smaller element is found, we update minIndex to the index of that element.

// After the inner loop finishes, we have found the minimum element in the unsorted part of the array. We then swap this minimum element with the element at index i, effectively placing it in the correct position in the sorted part of the array.

// This swapping is done using a temporary variable temp. We store the value of arr[i] in temp, assign the value of arr[minIndex] to arr[i], and finally assign the value of temp to arr[minIndex].

// The outer loop continues until all elements are sorted, and the array is sorted in ascending order.
    public static void selectionSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            int temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
        }
    }

```

```java
package oopd4;

import java.util.Scanner;

public class Sortings {

// Bubble Sort =>
// Start with an unsorted array of elements.
// Iterate through the array from the first element to the second-to-last element using the outer loop.
// Within the outer loop, iterate through the array from the first element to the (n - i - 1) element using the inner loop. Here, 'n' represents the length of the array and 'i' represents the current iteration of the outer loop.
// Compare each pair of adjacent elements in the inner loop.
// If the elements are in the wrong order (the current element is greater than the next element), swap them.
// Repeat steps 3-5 until the inner loop completes.
// Repeat steps 2-6 until the outer loop completes.
// The array is now sorted in ascending order.


    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

//Selection Sort =>
// The selectionSort method takes an array of integers (arr) as input and performs the selection sort algorithm on it.

// It starts by initializing a variable n with the length of the array, which represents the number of elements in the array.

// The outer loop runs from i = 0 to n - 1. This loop selects the current minimum element and places it in the correct position.

// Inside the outer loop, we initialize a variable minIndex with the value of i. This variable keeps track of the index of the minimum element found so far.

// The inner loop runs from j = i + 1 to n. It compares each element in the unsorted part of the array with the current minimum element (arr[minIndex]). If a smaller element is found, we update minIndex to the index of that element.

// After the inner loop finishes, we have found the minimum element in the unsorted part of the array. We then swap this minimum element with the element at index i, effectively placing it in the correct position in the sorted part of the array.

// This swapping is done using a temporary variable temp. We store the value of arr[i] in temp, assign the value of arr[minIndex] to arr[i], and finally assign the value of temp to arr[minIndex].

// The outer loop continues until all elements are sorted, and the array is sorted in ascending order.
    public static void selectionSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            int temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
        }
    }

    // Merge Sort =>
    // The mergeSort method takes an array of integers (arr), the left index (left), and the right index (right) as input and performs the merge sort algorithm on the specified range of the array.

    public static void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            int mid = (left + right) / 2;
            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);
            merge(arr, left, mid, right);
        }
    }

    private static void merge(int[] arr, int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;

        int[] leftArr = new int[n1];
        int[] rightArr = new int[n2];

        for (int i = 0; i < n1; i++) {
            leftArr[i] = arr[left + i];
        }
        for (int j = 0; j < n2; j++) {
            rightArr[j] = arr[mid + 1 + j];
        }

        int i = 0, j = 0;
        int k = left;

        while (i < n1 && j < n2) {
            if (leftArr[i] <= rightArr[j]) {
                arr[k] = leftArr[i];
                i++;
            } else {
                arr[k] = rightArr[j];
                j++;
            }
            k++;
        }

        while (i < n1) {
            arr[k] = leftArr[i];
            i++;
            k++;
        }

        while (j < n2) {
            arr[k] = rightArr[j];
            j++;
            k++;
        }
    }

    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();

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

        scanner.close();

        bubbleSort(arr);
        System.out.println("Bubble Sort:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();

        selectionSort(arr);
        System.out.println("Selection Sort:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();

        mergeSort(arr, 0, arr.length - 1);
        System.out.println("Merge Sort:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}

```

Discussion:

The Sortings program demonstrates three classic sorting algorithms: Bubble Sort, Selection Sort, and Merge Sort. Bubble Sort iteratively compares adjacent elements and swaps them if they are in the wrong order until the array is sorted. Selection Sort finds the smallest element in the unsorted portion of the array and swaps it with the first unsorted element, incrementally sorting the array. Merge Sort recursively divides the array into halves, sorts each half, and then merges them back together in sorted order. User input defines the array size and elements, which are then sorted using each algorithm sequentially. The program showcases fundamental sorting techniques and provides insight into their implementation and efficiency in different scenarios.

Question-8(practice-3):

Solve problem 4 by using ArrayList, LinkedList, and their various methods

Code:

```java
package oopd4;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class Pacticeproblem8 {
    public static void main(String[] args) {
        Book book1 = new Book("Title1", "Author1", 2001, "Genre1");
        Book book2 = new Book("Title2", "Author2", 2002, "Genre2");
        Book book3 = new Book("Title3", "Author3", 2003, "Genre3");

        List<Book> bookArrayList = new ArrayList<>();
        bookArrayList.add(book1);
        bookArrayList.add(book2);
        bookArrayList.add(book3);

        System.out.println("Books in ArrayList:");
        for (Book book : bookArrayList) {
            System.out.println(book);
        }

        bookArrayList.remove(book2);

        System.out.println("\nBooks in ArrayList after removing book2:");
        for (Book book : bookArrayList) {
            System.out.println(book);
        }

        List<Book> bookLinkedList = new LinkedList<>();
        bookLinkedList.add(book1);
        bookLinkedList.add(book2);
        bookLinkedList.add(book3);

        System.out.println("\nBooks in LinkedList:");
        for (Book book : bookLinkedList) {
            System.out.println(book);
        }

        bookLinkedList.remove(book2);

        System.out.println("\nBooks in LinkedList after removing book2:");
        for (Book book : bookLinkedList) {
            System.out.println(book);
        }
    }
}

class Book {
    private String title;
    private String author;
    private int year;
```

Discussion:

we are required to create a Book class that holds information about a book, such as the title, author, year, and genre. The class should have a parameterized constructor to initialize these properties. After creating three book objects, we store them in both an ArrayList and a LinkedList, display the contents, remove one of the objects, and display the updated lists.

Using both ArrayList and LinkedList provides an opportunity to compare their behavior and performance characteristics. ArrayList offers faster access times for getting elements due to its underlying array structure, making it a better choice when random access is a frequent operation. On the other hand, LinkedList excels in scenarios where frequent insertions and deletions are required, thanks to its doubly linked structure, which avoids shifting elements.

In the solution provided, we demonstrated how to add and remove elements from both types of lists and displayed their contents. This practical example helps illustrate the strengths and weaknesses of each list type in Java, giving insight into their appropriate usage contexts.

Question-9(practice-4):

Solve practice problems from Lab Sheet #3 using array, ArrayList, and LinkedList.

Code:

```java
package oopd4;

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

public class Practiceproblem9 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter details for 3 books (title, author, year, genre):");
        List<Book> booksArrayList = new ArrayList<>();
        for (int i = 0; i < 3; i++) {
            System.out.print("Enter title: ");
            String title = scanner.nextLine();
            System.out.print("Enter author: ");
            String author = scanner.nextLine();
            System.out.print("Enter year: ");
            int year = scanner.nextInt();
            scanner.nextLine();
            System.out.print("Enter genre: ");
            String genre = scanner.nextLine();
            booksArrayList.add(new Book(title, author, year, genre));
        }

        System.out.println("\nBooks in ArrayList:");
        for (Book book : booksArrayList) {
            book.displayDetails();
        }
        Book.displayTotalBooks();

        List<Book> booksLinkedList = new LinkedList<>(booksArrayList);

        System.out.println("\nBooks in LinkedList:");
        for (Book book : booksLinkedList) {
            book.displayDetails();
        }

        System.out.println("\nEnter details for 3 students (id, name, department, cgpa, university):");
        List<Student> studentsArrayList = new ArrayList<>();
        for (int i = 0; i < 3; i++) {
            System.out.print("Enter id: ");
            int id = scanner.nextInt();
            scanner.nextLine();
            System.out.print("Enter name: ");
            String name = scanner.nextLine();
            System.out.print("Enter department: ");
            String department = scanner.nextLine();
            System.out.print("Enter cgpa: ");
            double cgpa = scanner.nextDouble();
            scanner.nextLine();
            System.out.print("Enter university: ");
            String university = scanner.nextLine();
            studentsArrayList.add(new Student(id, name, department, cgpa, university));
        }

        System.out.println("\nStudents in ArrayList:");
        for (Student student : studentsArrayList) {
            student.displayDetails();
        }
        Student.displayTotalStudents();

        List<Student> studentsLinkedList = new LinkedList<>(studentsArrayList);

        System.out.println("\nStudents in LinkedList:");
        for (Student student : studentsLinkedList) {
            student.displayDetails();
        }
    }
}

class Book {
    private String title;
    private String author;
    private int year;
    private static String genre;
    private static int bookCount = 0;

    public Book(String title, String author, int year, String genre) {
        this.title = title;
        this.author = author;
        this.year = year;
        Book.genre = genre;
        bookCount++;
    }

    public void displayDetails() {
        System.out.println("Title: " + title + ", Author: " + author + ", Year: " + year + ", Genre: " + genre);
    }

    public static void displayTotalBooks() {
        System.out.println("Total number of books: " + bookCount);
    }
}
```

```java
class Student {
    private int id;
    private String name;
    private String department;
    private double cgpa;
    private static String university;
    private static int studentCount = 0;

    public Student(int id, String name, String department, double cgpa, String university) {
        this.id = id;
        this.name = name;
        this.department = department;
        this.cgpa = cgpa;
        Student.university = university;
        studentCount++;
    }

    public void displayDetails() {
        System.out.println("ID: " + id + ", Name: " + name + ", Department: " + department + ", CGPA: " + cgpa + ", University: " + university);
    }

    public static void displayTotalStudents() {
        System.out.println("Total number of students: " + studentCount);
    }
}
```

Discussion:

we explore the creation and manipulation of two different classes, Book and Student, each with specific properties. The Book class includes object variables for the title, author, and year, and a class variable for the genre. Similarly, the Student class includes object variables for the id, name, department, and cgpa, with a class variable for the university.

The challenge requires implementing parameterized constructors for both classes to initialize objects based on user input, highlighting the importance of user interaction in object-oriented programming. Additionally, object methods are used to display the details of each book and student, while class methods provide a way to keep track of the total number of instances created, demonstrating the use of static members in Java.

We then store these objects in arrays, ArrayList, and LinkedList, and display their details. This not only reinforces the concept of object storage in different data structures but also provides a practical understanding of how to manipulate these collections.