List/ArrayList Programs

1. Basic Operations on ArrayList

Problem: Write a Java program to perform the following operations on an `ArrayList`:

- Create an `ArrayList` of integers.
- Add elements to the `ArrayList`.
- Access and print elements using both index and enhanced for-loop.
- Remove an element from the `ArrayList`.
- Check if a specific element exists in the list.

Sample Input/Output:

```
ArrayList: [10, 20, 30, 40, 50]
After removing element at index 2: [10, 20, 40, 50]
Contains 20? true
```

2. Find the Maximum Element in an ArrayList

Problem: Write a Java program to find the largest element in an `ArrayList` of integers.

Sample Input/Output:

```
Input: [5, 9, 12, 7, 3]
Output: Maximum element: 12
```

3. Remove Duplicates from an ArrayList

Problem: Write a Java program to remove duplicate elements from an `ArrayList` of integers.

Sample Input/Output:

```
Input: [1, 2, 2, 3, 4, 4, 5]
Output: [1, 2, 3, 4, 5]
```

4. Merge Two ArrayLists

Problem: Write a Java program to merge two `ArrayList`s into a single `ArrayList`.

Sample Input/Output:

```
Input: list1 = [1, 2, 3], list2 = [4, 5, 6]
Output: Merged list: [1, 2, 3, 4, 5, 6]
```

5. ArrayList Sorting

Problem: Write a Java program to sort an `ArrayList` of integers in ascending and descending order.

Sample Input/Output:

```
Input: [4, 1, 7, 2, 9]
Output (ascending): [1, 2, 4, 7, 9]
Output (descending): [9, 7, 4, 2, 1]
```

6. Find Common Elements Between Two ArrayLists

Problem: Write a Java program to find the common elements between two `ArrayList`s of strings.

Sample Input/Output:

```
Input: list1 = ["apple", "banana", "orange"], list2 = ["banana", "kiwi",
"orange"]
Output: Common elements: [banana, orange]
```

7. Reverse an ArrayList

Problem: Write a Java program to reverse the elements of an `ArrayList`.

Sample Input/Output:

```
Input: [10, 20, 30, 40]
Output: [40, 30, 20, 10]
```

8. Check if an ArrayList is Empty

Problem: Write a Java program to check if an `ArrayList` is empty or not.

Sample Input/Output:

```
Input: []
Output: Is the ArrayList empty? true
```

9. Convert ArrayList to Array

Problem: Write a Java program to convert an `ArrayList` of integers into an array.

```
Input: [1, 2, 3, 4]
Output (array): [1, 2, 3, 4]
```

10. Insert an Element at a Specific Index

Problem: Write a Java program to insert an element at a specific index in an `ArrayList`.

Sample Input/Output:

```
Input: [10, 20, 30, 40]
Insert 25 at index 2
Output: [10, 20, 25, 30, 40]
```

11. Remove Elements from an ArrayList

Problem: Write a Java program to remove all elements from an `ArrayList` that are divisible by 3.

Sample Input/Output:

```
Input: [3, 5, 9, 12, 14, 18]
Output: [5, 14]
```

12. Shuffle Elements of an ArrayList

Problem: Write a Java program to randomly shuffle the elements of an `ArrayList`.

Sample Input/Output:

```
Input: [1, 2, 3, 4, 5]
Output (shuffled): [4, 2, 5, 1, 3]
```

13. Convert Array to ArrayList

Problem: Write a Java program to convert an array of integers into an `ArrayList`.

Sample Input/Output:

```
Input: [1, 2, 3, 4]
Output: [1, 2, 3, 4]
```

14. Find the Index of a Specific Element

Problem: Write a Java program to find the index of a specific element in an `ArrayList`. If the element is not found, return `-1`.

Sample Input/Output:

```
Input: [10, 20, 30, 40], element = 30
Output: Index of 30: 2
```

15. Remove All Occurrences of a Specific Element

Problem: Write a Java program to remove all occurrences of a specific element from an `ArrayList`.

Sample Input/Output:

```
Input: [1, 2, 2, 3, 4, 2]
Remove all occurrences of 2
Output: [1, 3, 4]
```

16. Replace Elements in an ArrayList

Problem: Write a Java program to replace all occurrences of a specific element in an `ArrayList` with another element.

Sample Input/Output:

```
Input: [10, 20, 30, 20, 40], replace 20 with 50
Output: [10, 50, 30, 50, 40]
```

17. Find the Second Largest Element in an ArrayList

Problem: Write a Java program to find the second largest element in an `ArrayList` of integers.

Sample Input/Output:

```
Input: [1, 3, 4, 5, 2]
Output: Second largest element: 4
```

18. Check if Two ArrayLists Are Equal

Problem: Write a Java program to check if two `ArrayList`s are equal. Two `ArrayList`s are equal if they contain the same elements in the same order.

Sample Input/Output:

```
Input: list1 = [1, 2, 3], list2 = [1, 2, 3]
Output: Are they equal? true
```

19. Copy One ArrayList into Another

Problem: Write a Java program to copy one `ArrayList` into another.

Sample Input/Output:

```
Input: [1, 2, 3, 4]
Copied ArrayList: [1, 2, 3, 4]
```

20. ArrayList of Custom Objects

Problem: Create a custom `Student` class with fields `name` and `age`. Write a Java program to:

- Create an `ArrayList` of `Student` objects.
- Add `Student` objects to the `ArrayList`.
- Print all `Student` names whose age is greater than 18.

Sample Input/Output:

```
Input: [Student(name=John, age=20), Student(name=Mary, age=17),
Student(name=Tom, age=19)]
Output: John, Tom
```

1. Store and Display Book Information

Problem: Create a class `Book` with the following properties:

- `title` (String)
- `author` (String)
- `price` (double)

Write a program to:

- Create an `ArrayList` to store `Book` objects.
- Add at least 5 books to the list.
- Display the details of all books stored in the `ArrayList`.

Sample Input/Output:

```
Title: "The Alchemist", Author: "Paulo Coelho", Price: 299.99
Title: "To Kill a Mockingbird", Author: "Harper Lee", Price: 399.50
...
```

2. Find Employees by Department

Problem: Create a class `Employee` with the following properties:

- `name` (String)
- `department` (String)
- `salary` (double)

Write a program to:

- Add 5 employees to an `ArrayList`.
- Create a method that takes a department name and returns a list of employees who belong to that department.

Sample Input/Output:

```
Department: IT
Employees in IT: [John, Alice]
```

3. Student Grading System

Problem: Create a class `Student` with the following properties:

- `id` (int)
- `name` (String)
- `marks` (double)

Write a program to:

- Create an `ArrayList` to store multiple `Student` objects.
- Add at least 5 students to the list.
- Write a method to calculate the average marks of all students.
- Write another method to find the top scorer and display the student's details.

Sample Input/Output:

```
Top Scorer: Name - "Alice", Marks - 95.5
Average Marks: 78.4
```

4. Manage an Inventory System

Problem: Create a class `Product` with the following properties:

- `productId` (int)
- `productName` (String)
- `price` (double)
- `quantity` (int)

Write a program to:

- Create an `ArrayList` to store `Product` objects.
- Add at least 5 products to the list.
- Write methods to:
    - Add stock to a product.

- Reduce stock when a product is sold.
- Calculate the total value of the inventory.

Sample Input/Output:

```
Product ID: 101, Name: "Laptop", Quantity: 10, Total Value: 450000.0
```

5. Library Management System

Problem: Create a class `LibraryMember` with the following properties:

- `memberId` (int)
- `memberName` (String)
- `booksBorrowed` (ArrayList<Book>)

Write a program to:

- Allow a member to borrow up to 3 books.
- Track the books borrowed by each member using an `ArrayList<Book>` for each member.
- Implement methods to:
  - Borrow a book (add a book to the member's borrowed list).
  - Return a book (remove a book from the borrowed list).

Sample Input/Output:

```
Member: Alice, Borrowed Books: ["The Alchemist", "To Kill a Mockingbird"]
```

6. Hotel Reservation System

Problem: Create a class `Reservation` with the following properties:

- `reservationId` (int)
- `customerName` (String)
- `roomNumber` (int)
- `checkInDate` (String)
- `checkOutDate` (String)

Write a program to:

- Create an `ArrayList` of `Reservation` objects.
- Implement methods to:
  - Add a new reservation.
  - Cancel a reservation (remove from the list).
  - Display all reservations for a specific date range.

Sample Input/Output:

```
Reservation ID: 1, Customer: "John", Room: 101, Check-in: 2024-10-01, Check-out: 2024-10-05
```

7. Car Rental Service

Problem: Create a class `Car` with the following properties:

- `carId` (int)
- `model` (String)
- `brand` (String)
- `isAvailable` (boolean)

Write a program to:

- Create an `ArrayList` of cars.
- Implement methods to:
  - Rent a car (mark it as unavailable).
  - Return a car (mark it as available).
  - List all available cars.

Sample Input/Output:

```
Available Cars: [CarID: 102, Model: "Civic", Brand: "Honda"]
```

8. Event Management System

Problem: Create a class `Event` with the following properties:

- `eventId` (int)
- `eventName` (String)
- `eventDate` (String)
- `attendees` (ArrayList<String>)

Write a program to:

- Create an `ArrayList` to store `Event` objects.
- Add at least 3 events.
- Write a method to allow users to register for an event (add their name to the `attendees` list).
- Display the list of attendees for a specific event.

Sample Input/Output:

```
Event: "Tech Conference", Attendees: ["John", "Alice"]
```

9. Train Ticket Booking System

Problem: Create a class `TrainTicket` with the following properties:

- `ticketId` (int)
- `passengerName` (String)
- `trainNumber` (int)
- `seatNumber` (String)
- `isConfirmed` (boolean)

Write a program to:

- Create an `ArrayList` to store `TrainTicket` objects.

- Implement methods to:
    - Book a ticket (add to the list and mark as confirmed).
    - Cancel a ticket (remove from the list).
    - Display all confirmed tickets for a specific train.

Sample Input/Output:

```
Train: 101, Confirmed Tickets: [Passenger: John, Seat: 1A]
```

10. Course Enrollment System

Problem: Create a class `Course` with the following properties:

- `courseId` (int)
- `courseName` (String)
- `studentsEnrolled` (ArrayList<Student>)

Write a program to:

- Add multiple `Course` objects to an `ArrayList`.
- Allow students to enroll in a course (add a `Student` object to the `studentsEnrolled` list).
- Write a method to display all the students enrolled in a particular course.

Sample Input/Output:

```
Course: "Java Programming", Enrolled Students: [Alice, Bob, Charlie]
```

1. Managing a Library

A library needs to manage a list of books. Each book is identified by its title, and new books are added as they are acquired. The library also allows for removing books no longer available. The librarian needs to:

Add new books to the collection.
Remove a specific book.
Check if a book is already in the collection.

Question: Write a Java program using the List interface to implement the above requirements. Which implementation (ArrayList or LinkedList) is more suitable, and why?

2. Employee Attendance System

In a company, employees mark their attendance each day. The attendance system stores employee IDs in the order they mark attendance.

At the end of the day, the manager wants to view all the marked IDs.
A duplicate entry of an employee ID may occur, which needs to be removed without disturbing the order.
Question: Write a program to store and process employee IDs using the List interface. How would you ensure no duplicates exist in the list?

3. Playlist Manager

A music streaming app allows users to create playlists. A playlist should:

Add songs to the end of the list.
Allow reordering of songs by swapping positions.
Remove a song by title if the user no longer wants it.

Question: Design a Java class using the List interface to represent a playlist. How would you efficiently reorder songs using an ArrayList?

4. Student Roll Call

In a classroom, a teacher maintains a roll call list in the order students arrive. The teacher wants to:

Add students to the list as they arrive.
Retrieve the student at a specific position to mark them present.
Remove a student from the list if they leave the class.

Question: Write a Java program using the List interface to manage the roll call list. Discuss whether ArrayList or LinkedList is better for frequent additions and removals.

5. Shopping Cart

An e-commerce platform uses a shopping cart feature to store items that customers want to purchase. The cart:

Allows adding items in the order they are selected.
Displays the total number of items in the cart.
Lets users remove an item by its index.

Question: Implement the shopping cart functionality using the List interface. Explain how the get() method helps in retrieving an item at a specific index.

6. Airport Queue System

An airport tracks the order in which passengers check in for a flight. The list of passengers:

Grows dynamically as more passengers check in.
Allows passengers to be removed if they cancel their tickets.
Maintains the order of check-in for boarding.

Question: Write a program using the List interface to implement the queue. What challenges would arise if you used an ArrayList for frequent removals?

7. Exam Results Announcement

A university announces exam results for students in alphabetical order of their names. The result-processing system:

Adds students' names as they register for the exam.
Sorts the list alphabetically before announcing results.
Allows retrieval of a student's position in the list.

Question: Using the List interface, write a program to add and sort student names. How would you sort the list, and which List implementation is more efficient for this purpose?

8. Movie Theater Seat Booking

A movie theater tracks seat bookings in rows. Each row is represented by a list of booked seat numbers. The system:

Adds seat numbers as they are booked.
Allows cancellation of a specific seat.
Displays all booked seats in order.

Question: Write a Java program using the List interface to manage seat bookings for one row. Discuss whether LinkedList is a good choice if bookings and cancellations are frequent.

9. Tournament Rankings

A gaming tournament maintains a ranking list of players. The system:

Updates rankings as new players score higher.
Removes players who drop out of the competition.
Retrieves the current top player.

Question: Implement a program using the List interface to manage rankings. Explain how inserting elements at a specific position affects the performance of ArrayList versus LinkedList.

10. Grocery Store Inventory

A grocery store manages its inventory of items. The system:

Adds items as they are restocked.
Removes items that are sold out.
Retrieves the item at a specific index to update its quantity.

Question: Write a Java program using the List interface to manage the inventory. How does the choice of List implementation affect retrieval and update operations?

SET

Unique Library Books

Story: You are managing a library system where every book has a unique ISBN number. However, users sometimes try to add duplicate ISBNs mistakenly.

Question:

Write a Java program using a Set to store unique ISBNs of books. If a duplicate ISBN is attempted to be added, notify the user.

2. Team Formation

Story: A sports club is forming a team. The club collects a list of players' names, but duplicate entries are present due to multiple registrations.

Question:
Use a Set to store player names, ensuring no duplicate names exist in the final team list. Simulate this process by inputting player names into your program.

3. Event Guest List
Story: An event manager is creating a guest list for a wedding. The manager wants to avoid duplicate entries in the guest list.

Question:
Write a program that uses a Set to maintain the guest list. Add sample entries, including duplicates, and display the final unique guest list.

4. Social Media Followers
Story: A social media platform keeps track of the followers for each user. However, a person can follow another user only once.

Question:
Simulate this functionality using a Set. Write a program to store followers of a user and demonstrate how duplicates are handled.

5. Unique Product Codes
Story: An e-commerce platform assigns a unique product code to each item. A developer accidentally allows duplicate codes to be entered.

Question:

Write a program to store product codes in a Set. Show how the Set prevents duplicate entries, and display the final list of unique codes.

6. Conference Badge IDs

Story: At a tech conference, each attendee is assigned a unique badge ID. If duplicate IDs are found, they need to be rejected.

Question:

Implement a program using a Set to store badge IDs. Demonstrate adding valid and duplicate IDs, and print the final unique list.

7. Duplicate Word Detection

Story: You are building a text editor with a feature to highlight duplicate words in a document.

Question:

Write a program using a Set to find and print duplicate words from a given paragraph of text.

8. Classroom Attendance

Story: A school maintains attendance for each class. Each student's roll number must be unique.

Question:

Simulate the attendance system using a Set to store roll numbers. Write a program to add, check for duplicates, and print the final attendance list.

9. Music Playlist

Story: A music app allows users to create playlists. Songs must not be repeated in a playlist.

Question:

Write a program using a Set to manage a playlist. Demonstrate adding duplicate songs and show how they are handled.

10. Voter Registration
Story: A government system is registering voters. Each voter is identified by a unique voter ID.

Question:
Create a Java program using a Set to store voter IDs. Demonstrate adding duplicate IDs and ensure the system only keeps unique IDs.

MAP/HASHMAP

1. Basic Operations on HashMap

Problem: Write a Java program to:

- Create a HashMap to store student names and their grades.
- Add five student records.
- Retrieve and print the grade of a specific student by name.
- Update the grade of a student.
- Remove a student from the map.
- Print all the students and their grades.

Sample Input/Output:

```
Student: Alice, Grade: 85
Student: Bob, Grade: 90
Student: Charlie, Grade: 75
...
Enter student name to retrieve grade: Bob
Bob's grade is: 90

Updated Grade: Bob - 95

After removing Charlie:
{Alice=85, Bob=95, Eve=92, David=88}
```

2. Frequency of Characters

Problem: Write a Java program that takes a string as input and uses a `HashMap` to count the frequency of each character in the string. Ignore spaces.

Sample Input/Output:

```
Input: "hello world"
Output: {h=1, e=1, l=3, o=2, w=1, r=1, d=1}
```

3. Find Duplicates in an Array

Problem: Given an array of integers, use a `HashMap` to find and print the duplicate elements.

Sample Input/Output:

```
Input: [1, 2, 3, 1, 4, 5, 2]
Output: [1, 2]
```

4. Word Count in a Sentence

Problem: Write a Java program that reads a sentence and uses a `HashMap` to count the occurrences of each word in the sentence. Ignore case sensitivity.

Sample Input/Output:

```
Input: "Java is fun and Java is powerful"
Output: {java=2, is=2, fun=1, and=1, powerful=1}
```

5. Anagram Checker

Problem: Write a program to check whether two strings are anagrams of each other using a `HashMap`. Two strings are anagrams if they contain the same characters with the same frequency.

Sample Input/Output:

```
Input: "listen", "silent"
Output: true

Input: "hello", "world"
Output: false
```

6. Group Words with Same Letters

Problem: Given a list of words, group them based on whether they contain the same set of letters. Use a `HashMap` where the key is the sorted version of the word, and the value is a list of words with the same letters.

Sample Input/Output:

```
Input: ["bat", "tab", "rat", "tar", "art"]
Output: {abt=[bat, tab], art=[rat, tar, art]}
```

7. Count the Number of Distinct Elements in Every Window of Size K

Problem: Given an array of integers and a window size `k`, use a `HashMap` to print the count of distinct elements in every window of size `k`.

Sample Input/Output:

```
Input: arr = [1, 2, 1, 3, 4, 2, 3], k = 4
Output: [3, 4, 4, 3]
Explanation:
Window [1, 2, 1, 3] has 3 distinct numbers.
Window [2, 1, 3, 4] has 4 distinct numbers.
...
```

8. Two Sum Problem using HashMap

Problem: Given an array of integers and a target sum, use a `HashMap` to find if there are two numbers in the array that sum up to the target. Return the indices of those numbers.

Sample Input/Output:

```
Input: arr = [2, 7, 11, 15], target = 9
Output: [0, 1] (since arr[0] + arr[1] = 2 + 7 = 9)
```

9. Employee Department Mapping

Problem: Write a Java program to store employee names and their departments using a `HashMap`. Implement the following functionalities:

- Add a new employee and their department.
- Get the department of a specific employee.
- List all employees in a given department.

Sample Input/Output:

```
Input: Add Alice in HR, Bob in IT, Eve in IT
Output: Employees in IT: [Bob, Eve]
```

10. Student Ranking System

Problem: Implement a student ranking system where students and their scores are stored in a `HashMap`. Write a function to:

- Add student names and their scores.
- Get the rank of a student (1st, 2nd, 3rd, etc.) based on their score. Higher scores get a better rank.

Sample Input/Output:

```
Input: Alice - 85, Bob - 90, Charlie - 75
Output: Bob: 1st, Alice: 2nd, Charlie: 3rd
```

11. Intersection of Two Arrays using HashMap

Problem: Write a Java program to find the intersection of two arrays using a `HashMap`. The result should contain all the common elements between the two arrays.

Sample Input/Output:

```
Input: arr1 = [1, 2, 2, 3], arr2 = [2, 2, 4, 5]
Output: [2, 2]
```

12. Sort HashMap by Values

Problem: Write a Java program to sort a `HashMap` by values (ascending or descending order) and print the sorted map.

Sample Input/Output:

```
Input: {Alice=85, Bob=90, Charlie=75}
Output: {Charlie=75, Alice=85, Bob=90} // sorted by value
```

13. Check if Two Maps are Identical

Problem: Write a Java program to check whether two `HashMap` instances are identical in terms of keys and values.

Sample Input/Output:

```
Input: map1 = {Alice=85, Bob=90}, map2 = {Alice=85, Bob=90}
Output: true

Input: map1 = {Alice=85, Bob=90}, map2 = {Alice=85, Charlie=90}
Output: false
```

14. Merge Two HashMaps

Problem: Write a Java program to merge two `HashMap` instances into a new one. If there are any duplicate keys, combine their values (e.g., concatenate strings or sum integers).

Sample Input/Output:

```
Input: map1 = {Alice=85, Bob=90}, map2 = {Bob=10, Charlie=75}
Output: {Alice=85, Bob=100, Charlie=75}
```

15. Top K Frequent Elements

Problem: Given an array of integers, write a Java program using a `HashMap` to find the top `k` most frequent elements.

Sample Input/Output:

```
Input: arr = [1, 1, 1, 2, 2, 3], k = 2
Output: [1, 2] (1 appears 3 times, 2 appears 2 times)
```

These questions cover the core functionalities of Java's `HashMap` and help in developing both theoretical and practical knowledge about its use cases.

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

AnagramChecker

```
public class AnagramChecker {

    // Function to check if two strings are anagrams
    public static boolean areAnagrams(String str1, String str2) {
        // Remove spaces and convert both strings to lowercase for case-insensitivity
        str1 = str1.replaceAll("\\s", "").toLowerCase();
        str2 = str2.replaceAll("\\s", "").toLowerCase();
```

```
    // If the lengths are not equal, they cannot be anagrams
    if (str1.length() != str2.length()) {
        return false;
    }

    // Create a HashMap to store the frequency of characters
    HashMap<Character, Integer> charCountMap = new HashMap<>();

    // Count the frequency of each character in the first string
    for (char c : str1.toCharArray()) {
        charCountMap.put(c, charCountMap.getOrDefault(c, 0) + 1);
    }

    // Decrease the frequency of characters based on the second string
    for (char c : str2.toCharArray()) {
        if (!charCountMap.containsKey(c)) {
            return false;  // If a character is not found in the map, they are not anagrams
        }
        charCountMap.put(c, charCountMap.get(c) - 1);
    }

    // Check if all values in the map are zero, meaning both strings have the same characters
    for (int count : charCountMap.values()) {
        if (count != 0) {
            return false;
        }
    }
```

```
        return true;
    }

    public static void main(String[] args) {
        // Example test cases
        String str1 = "Listen";
        String str2 = "Silent";

        if (areAnagrams(str1, str2)) {
            System.out.println(str1 + " and " + str2 + " are anagrams.");
        } else {
            System.out.println(str1 + " and " + str2 + " are not anagrams.");
        }

        // Another test case
        String str3 = "Hello";
        String str4 = "World";

        if (areAnagrams(str3, str4)) {
            System.out.println(str3 + " and " + str4 + " are anagrams.");
        } else {
            System.out.println(str3 + " and " + str4 + " are not anagrams.");
        }
    }
}
```

Java Code: Group Words with Same Letters Using HashMap

```
import java.util.*;

public class GroupAnagrams {

    // Function to group words with the same letters
    public static List<List<String>> groupAnagrams(String[]
words) {
        // Create a HashMap to store groups of anagrams
        HashMap<String, List<String>> map = new HashMap<>();

        for (String word : words) {
            // Convert word to char array and sort it
            char[] chars = word.toCharArray();
            Arrays.sort(chars);
            String sortedWord = new String(chars);

            // If the sorted version of the word is already a
key in the map, add the word to the list
            if (!map.containsKey(sortedWord)) {
                map.put(sortedWord, new ArrayList<>());
            }
            map.get(sortedWord).add(word);
        }

        // Return the grouped anagrams as a list of lists
        return new ArrayList<>(map.values());
    }

    public static void main(String[] args) {
        // Sample input
        String[] words = {"bat", "tab", "rat", "tar", "art",
"bats"};

        // Grouping words with same letters
        List<List<String>> result = groupAnagrams(words);

        // Output the grouped anagrams
        for (List<String> group : result) {
```

```
                System.out.println(group);
            }
        }
}
```

Count the Number of Distinct Elements in Every Window of Size K

```
import java.util.*;

public class DistinctElementsInWindow {

    public static List<Integer> countDistinctInWindow(int[] arr, int k) {
        List<Integer> result = new ArrayList<>();
        HashMap<Integer, Integer> map = new HashMap<>();

        // Traverse the first window of size k
        for (int i = 0; i < k; i++) {
            map.put(arr[i], map.getOrDefault(arr[i], 0) + 1);
        }
        result.add(map.size());  // Add the count of distinct elements in the first window

        // Slide the window from index k to the end of the array
        for (int i = k; i < arr.length; i++) {
            // Remove the element going out of the window
```

```
            int outgoingElement = arr[i - k];
            if (map.get(outgoingElement) == 1) {
                map.remove(outgoingElement);
            } else {
                map.put(outgoingElement, map.get(outgoingElement)
- 1);
            }

            // Add the new element coming into the window
            int incomingElement = arr[i];
            map.put(incomingElement,
map.getOrDefault(incomingElement, 0) + 1);

            // Add the count of distinct elements for the current
window
            result.add(map.size());
        }

        return result;
    }

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

```
        Scanner scanner = new Scanner(System.in);

        // Input array and window size
        System.out.println("Enter the number of elements in the array:");
        int n = scanner.nextInt();
        int[] arr = new int[n];

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

        System.out.println("Enter the window size (k):");
        int k = scanner.nextInt();

        if (k > n) {
            System.out.println("Window size should not be greater than array size.");
            return;
        }
```

```
        // Get the result and print
        List<Integer> result = countDistinctInWindow(arr, k);
        System.out.println("Distinct elements count in each
window:");
        for (int count : result) {
            System.out.print(count + " ");
        }
    }
}
```