1. Find the odd and even numbers in the array

→

```
public void printEvenNumbers(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
       if (arr[i] % 2 == 0)  System.out.print(arr[i] + " ");
    }
  }
  public void printOddNumbers(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
       if (arr[i] % 2 != 0)  System.out.print(arr[i] + " ");
    }
  }
```

2. Find the vowels and consonants in a word/sentence

→

```
Public void VowelsAndConsonantsCounter(String str){
                        str = str.toLowerCase();
    int vowelCount = 0;
    int consonantCount = 0;

    for (int i = 0; i < str.length(); i++) {
       char ch = str.charAt(i);

       if (ch >= 'a' && ch <= 'z') {
          if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u')
             vowelCount++;
           else
             consonantCount++;
       }
    }

    System.out.println("Vowels: " + vowelCount);
    System.out.println("Consonants: " + consonantCount);

}
```

3. Reverse a string without using built in methods

→

```
  public String reverseString(String input) {
    int length = input.length();
    StringBuilder reversed = new StringBuilder();

    for (int i = length - 1; i >= 0; i--)
       reversed.append(input.charAt(i));
    return reversed.toString();
  }
```

4. Split the words in sentence

→

```
public void splitSentenceIntoWords(String sentence) {
    StringBuilder wordBuilder = new StringBuilder();
    boolean inWord = false;
    for (int i = 0; i < sentence.length(); i++) {
       char c = sentence.charAt(i);
       if (Character.isWhitespace(c)) {
          if (inWord) {
             System.out.println(wordBuilder.toString());
             wordBuilder.setLength(0); // Clear the word builder
             inWord = false;
          }
       } else {
          wordBuilder.append(c);
          inWord = true;
       }
    }
    if (inWord)
       System.out.println(wordBuilder.toString());
}
```

5. Find the prime numbers in an array

→

```
public int[] findPrimes(int[] numbers) {
    int[] primeNumbers = new int[numbers.length];
    int primeCount = 0;

    for (int num : numbers) {
       if (isPrime(num)) {
          primeNumbers[primeCount] = num;
          primeCount++;
       }
    }
    int[] result = new int[primeCount];
    System.arraycopy(primeNumbers, 0, result, 0, primeCount);
    return result;
 }
   public boolean isPrime(int num) {
    if (num <= 1)     return false;
    if (num <= 3)     return true;
    if (num % 2 == 0 || num % 3 == 0)    return false;

    for (int i = 5; i * i <= num; i += 6) {
       if (num % i == 0 || num % (i + 2) == 0)      return false;
    return true;
  }
```

6. Find the highest number in an array

→

```
public int findHighestNumber(int[] arr) {
    if (arr.length == 0) return -1;

    int highest = arr[0];

    for (int i = 1; i < arr.length; i++) {
       if (arr[i] > highest)   highest = arr[i];
    }
    return highest;
 }
```

7. Find the 3rd lowest number in an array

→

```
public int findThirdLowestNumber(int[] arr) {
    if (arr.length < 3) return -1;

    int lowest = Integer.MAX_VALUE;
    int secondLowest = Integer.MAX_VALUE;
    int thirdLowest = Integer.MAX_VALUE;

    for (int num : arr) {
       if (num < lowest) {
          thirdLowest = secondLowest;
          secondLowest = lowest;
          lowest = num;
       } else if (num < secondLowest) {
          thirdLowest = secondLowest;
          secondLowest = num;
       } else if (num < thirdLowest) {
          thirdLowest = num;
       }
    }
    return thirdLowest;
}
```

8. Generate Fibonacci series

→

```
  public void generateFibonacciSeries(int n) {
    int first = 0, second = 1;
    for (int i = 0; i < n; i++) {
       System.out.print(first + " ");
       int next = first + second;
       first = second;
       second = next;
    }  }
```

9. Find the odd and even numbers in the array

→

```
public int[] findEvenNumbers(int[] arr) {
   int count = 0;
   for (int num : arr) {
      if (num % 2 == 0) count++;
   }
   int[] evenNumbers = new int[count];
   int index = 0;
   for (int num : arr) {
      if (num % 2 == 0) {
         evenNumbers[index] = num;
         index++;
      }
   }
   return evenNumbers;
}

public int[] findOddNumbers(int[] arr) {
   int count = 0;
   for (int num : arr) {
      if (num % 2 != 0)  count++;
   }

   int[] oddNumbers = new int[count];
   int index = 0;
   for (int num : arr) {
      if (num % 2 != 0) {
         oddNumbers[index] = num;
         index++;
      }
   }
   return oddNumbers;
}
```

10. Find the missing numbers in series

→

```
public static void findAndPrintMissingNumbers(int[] series) {
   int min = series[0];
   int max = series[series.length - 1];

   System.out.print("Missing numbers in the series: ");

   for (int num = min; num <= max; num++) {
      boolean found = false;
      for (int value : series) {
         if (value == num) {
```

```
                found = true;
                break;
            }
        }
        if (!found)     System.out.print(num + " ");
            }
        System.out.println();
    }
```

11. Find the factorial of a number using recursion

→

```
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();

        if (num < 0) {
            System.out.println("Factorial is not possible for negative numbers.");
        } else {
            long fac = Factorial(num);
            System.out.println("The factorial of " + num + " is: " + fac);
        }
    }

    public static long Factorial(int n) {
        if (n == 0 || n == 1) {
            return 1;
        }
        return n * Factorial(n - 1);
    }
}
```

12. Find if the word is a palindrome

→

```
    public static boolean isPalindrome(String s) {
        int l = 0;
        int h = s.length() - 1;

        while (l < h) {
            if (s.charAt(l) != s.charAt(h))       return false;
            l++; h--;
        }

        return true;
    }
```

13. Write a program to compare two strings

→

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

      System.out.print("Enter the first string: ");
      String str1 = scanner.nextLine();

      System.out.print("Enter the second string: ");
      String str2 = scanner.nextLine();

      if (areStringsEqual(str1, str2)) {
         System.out.println("Both strings are equal");
      } else {
         System.out.println("Both strings are not equal");
      }
   }

   public static boolean areStringsEqual(String str1, String str2) {
      return str1.equals(str2);
   }
}
```

14. Sort the numbers/words in ascending/descending order

→

```
public class Solution {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      int n = sc.nextInt();
      int[] a = new int[n];

      for (int i = 0; i < n; i++)
         a[i] = sc.nextInt();

      bubbleSort(a);

      for (int i = 0; i < n; i++)
         System.out.print(a[i] + " ");
   }

   public static void bubbleSort(int[] arr) {
      int n = arr.length;
      int temp = 0;

      for (int i = 0; i < n; i++) {
```

```
            for (int j = 0; j < n - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }

    }
}
```

15. Understand and implement all the Searching (Binary, Linear, Jump), Sorting (Quick, Bubble, Merge, Insertion, Selection, heap) algorithms

→

Searching Algorithms:

1. Linear Search:

It checks each element in a list until the target element is found or the end of the list is reached.

Time Complexity: O(n)

```
public static int linearSearch(int[] arr, int target) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == target) {
            return i; // Element found at index i
        }
    }
    return -1; // Element not found
}
```

2. Binary Search:

Works on sorted arrays by repeatedly dividing the search interval in half.

Time Complexity: O(log n)

```
public static int binarySearch(int[] arr, int target) {
    int left = 0, right = arr.length - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            return mid; // Element found at mid
        } else if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1; // Element not found
}
```

3. Jump Search:

A searching algorithm for ordered lists. It jumps ahead by a fixed number of steps in each iteration.

Time Complexity: O(√n)

```
public static int jumpSearch(int[] arr, int target) {
    int n = arr.length;
    int step = (int) Math.sqrt(n);
    int prev = 0;
    while (arr[Math.min(step, n) - 1] < target) {
        prev = step;
        step += (int) Math.sqrt(n);
        if (prev >= n)    return -1;
    }
    while (arr[prev] < target) {
        prev++;

        if (prev == Math.min(step, n))     return -1;
    }
    if (arr[prev] == target)    return prev;

    return -1;
}
```

Sorting Algorithms:

1. Bubble Sort:

Repeatedly compares adjacent elements and swaps them if they are in the wrong order.
Time Complexity: O(n^2)

2. Quick Sort:

A divide-and-conquer algorithm that chooses a 'pivot' element and partitions the array based on the pivot.
Time Complexity: O(n log n) on average

3. Merge Sort:

A divide-and-conquer algorithm that divides the array into two halves, sorts them, and then merges the two sorted halves.
Time Complexity: O(n log n)

4. Insertion Sort:

Builds the final sorted array one item at a time.
Time Complexity: O(n^2)

5. Selection Sort:

Repeatedly finds the minimum element from the unsorted part of the array and swaps it with the first unsorted element.
Time Complexity: O(n^2)

6. Heap Sort:

A comparison-based sorting algorithm that uses a binary heap data structure.
Time Complexity: O(n log n)

16. Understand and implement Arrays, data types, and linked-lists

→

1. Arrays:

An array is a fixed-size, ordered collection of elements of the same data type. Arrays are indexed, which means elements are accessed by their position (index) in the array.

2. Data Types:

Data types define the type of data that a variable can hold. Java has primitive data types (int, float, char, etc.) and reference data types (classes, interfaces, arrays, etc.).

3. Linked Lists:

A linked list is a data structure that consists of a sequence of elements where each element points to the next element. Linked lists are dynamic and can grow or shrink as needed.

17. Implement as many programs as possible in the areas Fibonacci series, Factorial of a number using Recursion

→

```
public class Main {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int n = sc.nextInt();

      for (int i = 0; i < n; i++) {
         int fibonacciValue = fibonacci(i);
         System.out.print(fibonacciValue + " ");
      }
   }

   public static int fibonacci(int n) {
      if (n <= 1)    return n;

      return calculateFibonacci(n);
   }

   private static int calculateFibonacci(int n) {
      int[] fib = new int[n + 1];
      fib[0] = 0;
      fib[1] = 1;

      for (int i = 2; i <= n; i++) {
         fib[i] = fib[i - 1] + fib[i - 2];
      }
      return fib[n];
   }}
```

19. OOPS concepts (Abstraction, Encapsulation, Inheritance, Polymorphism) with examples

→

Object-Oriented Programming (OOP) is a programming paradigm that revolves around the concept of objects, which are instances of classes.

1. Abstraction:

Abstraction is the process of simplifying complex reality by modeling classes based on real-world objects.

It hides the complex implementation details while exposing only the necessary features.

Real-Time Example

Abstraction: A remote control abstracts the complex functionality of electronic devices (e.g., TV, DVD player) into a simple interface with buttons (e.g., power, volume, channel). Users don't need to understand the inner workings of each device; they interact with the remote control to perform actions. Abstraction hides the complexity and exposes only the essential features, making it user-friendly.

2. Encapsulation:

Encapsulation is the concept of bundling data (attributes) and methods (functions) that operate on that data into a single unit called a class.

It provides data hiding and restricts access to certain components of the object.

Real-Time Example

Encapsulation: In a bank account, we have data (e.g., balance) and methods (e.g., deposit and withdraw) that operate on that data. Encapsulation ensures that the data (balance) is protected from unauthorized access and modification. we can only interact with the account through well-defined methods (deposit and withdraw), which apply access control and maintain the integrity of the account's data.

3. Inheritance:

Inheritance allows a class to inherit properties and methods from another class.

It promotes code reuse and hierarchy in the class structure.

Real-Time Example

Inheritance: In a school system, we have a class order where we have a base class like "Person" and derived classes like "Student" and "Teacher." Inheritance allows the "Student" and "Teacher" classes to inherit properties and methods from the "Person" class.

4. Polymorphism:

Polymorphism allows objects of different classes to be treated as objects of a common superclass.

It enables flexibility and dynamic method dispatch.

Real-Time Example

Polymorphism: In a media player application, you can play various types of media (e.g., audio and video). Polymorphism allows you to use a common interface (e.g., play()) for different media types. At runtime, the correct method (e.g., playAudio() or playVideo()) is invoked based on the actual type of media being played. This flexibility simplifies the code .

20. DB concepts: Schema design, Inner, Outer joins, Complex SQLs, Migrations, problems with migration

→

1. Schema Design:

- Schema design involves structuring the database by defining tables, their relationships, and constraints.
- A well-designed schema ensures data integrity and efficient querying.

Example:

```
CREATE TABLE Customers (
   CustomerID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50)
);

CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   CustomerID INT,
   OrderDate DATE,
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

In this example, we define a schema with two tables, "Customers" and "Orders," and establish a foreign key relationship.

2. Inner Joins:

Inner joins retrieve rows from multiple tables where there's a matching value in both tables.

Example:

```
SELECT Customers.CustomerID, Customers.FirstName, Orders.OrderDate
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

This SQL query retrieves customer information along with their order dates from the "Customers" and "Orders" tables.

3. Outer Joins:

Outer joins return rows from one table even if there's no matching row in the other table.
Common types are LEFT OUTER JOIN, RIGHT OUTER JOIN, and FULL OUTER JOIN.

Example:

```
SELECT Customers.CustomerID, Customers.FirstName, Orders.OrderDate
FROM Customers
LEFT OUTER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

This query retrieves customer information along with their order dates, including customers with no orders.

4. Complex SQL Queries:

Complex SQL queries involve multiple operations, subqueries, and conditions to retrieve specific data.
They often require a deep understanding of SQL and the database schema.

Example:

```
SELECT ProductName, SUM(Quantity * Price) AS TotalRevenue
FROM Products
JOIN OrderDetails ON Products.ProductID = OrderDetails.ProductID
GROUP BY ProductName
HAVING TotalRevenue > 10000
ORDER BY TotalRevenue DESC;
```

This complex SQL query calculates the total revenue for each product and filters products with revenue over $10,000.

5. Database Migrations:

Database migrations involve updating the database schema or data to accommodate changes in the application. Migration scripts are used to make changes in a controlled manner.

Example (using a migration tool like Flyway):

```
-- Version 1: Initial schema
CREATE TABLE Customers (
   CustomerID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50)
);

-- Version 2: Add email column to Customers
ALTER TABLE Customers
ADD Email VARCHAR(100);
```

6. Problems with Migrations:

- Data Loss: Migrations can lead to data loss if not handled carefully.
- Downtime: Performing migrations on a live system can cause downtime.
- Rollbacks: In case of migration failure, reverting to a previous state can be challenging.

Database migrations require careful planning and testing to avoid these problems.

(20)1. Create a table with (ID, Phy, Chem, Maths) with ID as primary key.

→

```
CREATE TABLE Scores (
   ID INT PRIMARY KEY,
   Phy INT,
   Chem INT,
   Maths INT
);
```

(20)2. Add another column Biology

→

```
ALTER TABLE Scores
ADD Biology INT;
```

(20)3. Create 4 records

→

```
-- record 1
INSERT INTO Scores (ID, Phy, Chem, Maths, Biology)
VALUES (1, 85, 90, 78, 92);

-- record 2
INSERT INTO Scores (ID, Phy, Chem, Maths, Biology)
VALUES (2, 92, 88, 76, 85);

-- record 3
INSERT INTO Scores (ID, Phy, Chem, Maths, Biology)
VALUES (3, 78, 94, 82, 90);

-- record 4
INSERT INTO Scores (ID, Phy, Chem, Maths, Biology)
VALUES (4, 88, 76, 85, 80);
```

21. Change data for a record or two

→

```
UPDATE Scores
SET Chem = 95
WHERE ID = 1;
```

1. Find the ID with the highest Phy score

```
SELECT ID
FROM Scores
ORDER BY Phy DESC
LIMIT 1;
```

2. Find the ID with highest total score

```
SELECT ID
FROM Scores
ORDER BY (Phy + Chem + Maths + Biology) DESC
LIMIT 1;
```

3. Ascending order based on total score

```
SELECT *
FROM Scores
ORDER BY (Phy + Chem + Maths + Biology) ASC;
```

4. Descending order based on total score

```
SELECT *
FROM Scores
ORDER BY (Phy + Chem + Maths + Biology) DESC;
```

5. 2nd highest Maths scorer

```
SELECT ID
FROM Scores
ORDER BY Maths DESC
LIMIT 1 OFFSET 1;
```

22. Build REST APIs

1. Create users
2. Read user data
3. Update user data
4. Delete a few attributes of user data and
5. Delete user records completely

→

https://github.com/Saeed-Mujawar/User-Management-Application

24. Core Java interview questions Collections classes, Static, Volatile, Synchronize, Serialize, Inner classes

→

Collections Classes:

- **What is the difference between List, Set, and Map in Java collections?**

List allows duplicate elements and maintains order.
Set doesn't allow duplicates and doesn't maintain order.
Map stores key-value pairs, and keys are unique.

- **Explain the difference between ArrayList and LinkedList.**

ArrayList uses a dynamic array for storage.
LinkedList uses a doubly-linked list for storage.
ArrayList provides fast random access, while LinkedList is better for frequent insertions and deletions.

- **What is the purpose of the java.util.Collections class?**

The java.util.Collections class provides utility methods for working with collections, such as sorting, searching, and synchronization.

Static:

- **What is the static keyword in Java, and how is it used?**

The static keyword is used to create class-level variables and methods.
Static members are shared among all instances of the class and can be accessed using the class name.

- **What is the difference between a static method and an instance method?**

Static methods are associated with the class itself, while instance methods are associated with objects of the class.
Static methods can be called using the class name, while instance methods are called on instances of the class.

Volatile:

- **What does the volatile keyword do in Java?**

The volatile keyword is used to indicate that a variable's value may be changed by multiple threads.
It ensures that changes to the variable are visible to all threads and prevents compiler optimizations that could reorder reads and writes.

Synchronization:

- **What is synchronization in Java, and why is it needed?**

Synchronization is the process of controlling access to shared resources by multiple threads to avoid data inconsistency.It is needed to prevent race conditions and ensure data integrity in multi-threaded programs.

Serialization:

- **What is object serialization in Java?**

Serialization is the process of converting an object into a byte stream, which can be saved to a file or sent over a network.
Deserialization is the reverse process of reconstructing the object from the byte stream.

- **How do you make a class serializable in Java?**

To make a class serializable, it needs to implement the Serializable interface.
This interface doesn't have any methods but serves as a marker for the Java serialization mechanism.

Inner Classes:

- **What is an inner class in Java?**

An inner class is a class defined within another class.
It can access the outer class's members and provides better encapsulation and organization.

- **What are the types of inner classes in Java?**

There are four types of inner classes: local inner classes, anonymous inner classes, non-static (member) inner classes, and static nested classes.

25. Find first non repeated character in a string.

→

```
public class FirstNonRepeatedCharacter {
   public static void main(String[] args) {
      String input = "programming";
      char firstNonRepeated = findFirstNonRepeatedCharacter(input);

      if (firstNonRepeated != '\0') {
         System.out.println("The first non-repeated character is: " + firstNonRepeated);
      } else {
         System.out.println("No non-repeated character found.");
      }
   }

   public static char findFirstNonRepeatedCharacter(String input) {
      for (int i = 0; i < input.length(); i++) {
         char currentChar = input.charAt(i);
         if (input.indexOf(currentChar) == i && input.lastIndexOf(currentChar) == i) {
            return currentChar;
         }
      }
      return '\0'; // Return '\0' if no non-repeated character is found.
   }
}
```

26. Given int a=2 another number given int b=3, int[]x={1,6,8,9,5} you have to add one number from the given array with a so that it will become equal to b and return the index of the array for which it matches.

→

```
public class FindIndexForSum {
   public static void main(String[] args) {
      int a = 2; int b = 3;
      int[] x = {1, 6, 8, 9, 5};
      int result = findIndexForSum(x, a, b);

      if (result != -1) {
```

```
            System.out.println("Index of the matching element: " + result);
        } else {
            System.out.println("No match found.");
        }
    }
    public static int findIndexForSum(int[] arr, int a, int b) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] + a == b)    return i;
        }
        return -1;
    }}
```

27. Check whether a given string is palindrome or not using Time Complexity / HashMap

→

```
public class Main {
    public static void main(String[] args) {
        String input = "racecar";
        boolean isPalindrome = isPalindrome(input);

        if (isPalindrome) {
            System.out.println(input + " is a palindrome.");
        } else {
            System.out.println(input + " is not a palindrome.");
        }
    }

    public static boolean isPalindrome(String str) {
        str = str.toLowerCase();       HashMap<Character, Integer> charCount = new HashMap<>();

        for (char c : str.toCharArray()) {
            charCount.put(c, charCount.getOrDefault(c, 0) + 1);
        }

        int oddCount = 0;
        for (int count : charCount.values()) {
            if (count % 2 != 0) {
                oddCount++;
            }
        }
        return oddCount <= 1;
    }
}
```

28. Find the highest frequent character in a string

→

```
public class Main {
   public static void main(String[] args) {
      String input = "programming";
      char highestFreqChar = findHighestFrequentCharacter(input);

      System.out.println("The highest frequent character is: " + highestFreqChar);
   }

   public static char findHighestFrequentCharacter(String str) {
      int[] charCount = new int[26];

      for (char c : str.toCharArray()) {
         if (Character.isLetter(c))   charCount[c - 'a']++;
      }

      char highestFreqChar = 'a';
      int maxFrequency = 0;

      for (int i = 0; i < 26; i++) {
         if (charCount[i] > maxFrequency) {
            maxFrequency = charCount[i];
            highestFreqChar = (char) (i + 'a');
         }
      }
      return highestFreqChar;
   }
}
```

29. Write a program to get values of union-intersection of two arrays (get uncommon values) A=[1,2,3,4,5] B = [5,6,7]

→

```
public class Main {
   public static void main(String[] args) {
      int[] A = {1, 2, 3, 4, 5};
      int[] B = {5, 6, 7};

      List<Integer> union = getUnion(A, B);
      List<Integer> intersection = getIntersection(A, B);

      System.out.println("Union of A and B: " + union);
      System.out.println("Intersection of A and B: " + intersection);
   }

   public static List<Integer> getUnion(int[] arr1, int[] arr2) {
      List<Integer> unionList = new ArrayList<>();
      for (int num : arr1) {
```

```
            unionList.add(num);
        }
        for (int num : arr2) {
            if (!unionList.contains(num)) {
                unionList.add(num);
            }
        }
        return unionList;
    }

    public static List<Integer> getIntersection(int[] arr1, int[] arr2) {
        List<Integer> intersectionList = new ArrayList<>();
        for (int num : arr1) {
            if (contains(arr2, num)) {
                intersectionList.add(num);
            }
        }
        return intersectionList;
    }

    public static boolean contains(int[] arr, int num) {
        for (int value : arr) {
            if (value == num) {
                return true;
            }
        }
        return false;
    }
}
```

30. Program for nth fibonacci.

→

```
public class Fibonacci {
    public static void main(String[] args) {
        int n = 10;
        int result = calculateFibonacci(n);
        System.out.println("The " + n + "th Fibonacci number is: " + result);
    }
    public static int calculateFibonacci(int n) {
        if (n <= 0) {
            return 0;
        } else if (n == 1) {
            return 1;
        } else {
            return calculateFibonacci(n - 1) + calculateFibonacci(n - 2);
        }
    }}
```

31. Write a program to find the element from an array which has the most frequency?

→

```
public class MostFrequentElement {
   public static void main(String[] args) {
      int[] arr = {1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5};
      int mostFrequentElement = findMostFrequentElement(arr);

      System.out.println("The most frequent element is: " + mostFrequentElement);
   }

   public static int findMostFrequentElement(int[] arr) {
      Map<Integer, Integer> elementCount = new HashMap<>();
      int mostFrequentElement = arr[0];
      int maxFrequency = 0;

      for (int num : arr) {
         int count = elementCount.getOrDefault(num, 0) + 1;
         elementCount.put(num, count);

         if (count > maxFrequency) {
            maxFrequency = count;
            mostFrequentElement = num;
         }
      }
      return mostFrequentElement;
   }
}
```