Scenario-Based Questions – Java Programming

S. No	Scenario-Based Questions	BTL	CO
1	Write a Java program named AllEqual.java that takes three integer command-line arguments and prints "equal" if all three integers are equal, and "not equal" otherwise. Ensure the program handles edge cases such as non-integer inputs and the incorrect number of arguments gracefully. Sample Input & Output: java AllEqual 4 4 4 equal java AllEqual 4 5 6 not equal Test Cases: Input: java AllEqual 3 3 3 Output: equal Input: java AllEqual 3 3 4 Output:not equal Input: java AllEqual 1 1 1 Output:Invalid input Input: java AllEqual 2.5 3 4 Output:Invalid input Input: java AllEqual abc 2 3 Output:Invalid input Input: java AllEqual 3 Output: Please provide exactly three integer arguments.	K4	CO1
2	**Bank Account Management System** 1. **Description:** Write a program to manage bank accounts. Each account has an account number, a balance, and an account type (Savings or Checking). The program should allow depositing, withdrawing, and viewing the account balance. Savings accounts should earn interest monthly at a rate of 4%, while checking accounts have a minimum balance requirement of $500, and if the balance falls below this, a fee of $50 is charged. 2. **Test Cases:** o Deposit $500 into a savings account with a balance of $1000. o Withdraw $200 from a checking account with a balance of $600. o View the balance of a savings account after applying monthly interest.		

	○ Withdraw $100 from a checking account with a balance of $400 (check for minimum balance and fee).		
3	Write a Java program to count the number of prime and composite numbers from a list of integers entered by the user. The program should handle edge cases such as negative numbers, non-integer inputs, and zero, and it should also handle an empty list gracefully. Sample Input & Output: Enter numbers (separated by spaces): 2 3 4 5 6 7 8 9 10 Number of prime numbers: 4 Number of composite numbers: 5 Test Cases: Input: 11 13 17 19 Output: Number of prime numbers: 4 Number of composite numbers: 0 Input: 4 6 8 9 10 Output: Number of prime numbers: 0 Number of composite numbers: 5 Input: 0 1 2 3 5 Output: Number of prime numbers: 3 Number of composite numbers: 0 Input: -7 -11 -13 Output: Number of prime numbers: 0 Number of composite numbers: 0		
4	Write a Java program to find the Mth maximum number and Nth minimum number in an array. After identifying these numbers, calculate the sum and difference between them. Ensure the program handles edge cases, such as invalid M or N values, appropriately. Sample Input & Output: Enter the array elements: [3, 1, 4, 9, 2, 7, 6] Enter the value of M (for Mth maximum): 2 Enter the value of N (for Nth minimum): 3 Mth maximum number: 7 Nth minimum number: 3 Sum: 10 Difference: 4 Test Cases:		

	Input: Array: [5, 2, 8, 1, 9, 7, 3] M = 1, N = 1 Output: Mth maximum number: 9 Nth minimum number: 1 Sum: 10 Difference: 8 Input: Array: [4, 8, 15, 16, 23, 42] M = 3, N = 2 Output: Mth maximum number: 16 Nth minimum number: 8 Sum: 24 Difference: 8 Input: Array: [12, 7, 22, 14, 9] M = 5, N = 1 Output: Mth maximum number: 7 Nth minimum number: 7 Sum: 14 Difference: 0 Input: Array: [6, 1, 3, 7, 2] M = 0, N = 2 Output: Invalid input for M or N		
5	**Palindrome Number Check** • **Description:** Write a Java program to check if a given integer is a palindrome. An integer is a palindrome if it reads the same backward as forward. • **Input:** A single integer. • **Output:** Print "Palindrome" if the number is a palindrome, otherwise print "Not a Palindrome". • **Example:** o Input: `121` o Output: `Palindrome` o Input: `123` o Output: `Not a Palindrome`		
6	**Anagram Checker**		

	• **Description:** Write a Java program to check if two given strings are anagrams of each other. An anagram of a string is another string that contains the same characters, only the order of characters can be different.
	• **Input:** Two strings.
	• **Output:** Print "Anagrams" if the strings are anagrams, otherwise print "Not Anagrams".
	• **Example:**
	  ○ Input: `"listen"`, `"silent"`
	  ○ Output: `Anagrams`
	  ○ Input: `"hello"`, `"world"`
	  ○ Output: `Not Anagrams`		

7	**Find the Missing Number in an Array**
	• **Description:** Given an array containing `n-1` distinct numbers taken from the range `1` to `n`, find the missing number.
	• **Input:** An array of integers.
	• **Output:** The missing integer.
	• **Example:**
	  ○ Input: `[1, 2, 4, 6, 3, 7, 8]`
	  ○ Output: `5`		

8	**Matrix Multiplication**
	• **Description:** Write a Java program to multiply two matrices. The program should read two matrices and print their product.
	• **Input:** Two 2D arrays (matrices).
	• **Output:** The product matrix.
	• **Example:**
	  ○ Input: `A = [[1, 2], [3, 4]],B = [[2, 0], [1, 2]]`
	  ○ Output: `[[4, 4], [10, 8]]`		

9	**Calculate the Frequency of Characters in a String** • **Description:** Write a Java program to calculate the frequency of each character in a given string. • **Input:** A single string. • **Output:** Print each character and its frequency. • **Example:** ○ Input: `"hello"`		

	Output h: 1 e: 1 l: 2 o: 1		

10	**Binary Search Implementation**
	
	• **Description:** Implement the binary search algorithm to find the index of a target value within a sorted array.
	• **Input:** A sorted array and a target value.
	• **Output:** The index of the target value if found, otherwise `-1`.
	• **Example:**
	  ○ Input: `[1, 3, 5, 7, 9], 5`
	  ○ Output: `2`
	  ○ Input: `[1, 3, 5, 7, 9], 6`
	  ○ Output: `-1`		

11	**Check for Balanced Parentheses**
	
	• **Description:** Write a Java program to check whether the parentheses in a given expression are balanced.
	• **Input:** A string containing parentheses.
	• **Output:** Print "Balanced" if the parentheses are balanced, otherwise print "Not Balanced".
	• **Example:**
	  ○ Input: `"(())"`
	  ○ Output: `Balanced`
	  ○ Input: `"()"`
	  ○ Output: `Not Balanced`		

12	**Remove Duplicates from a Sorted Array** • **Description:** Write a Java program to remove duplicates from a sorted array and return the new length of the array. • **Input:** A sorted array of integers. • **Output:** The new length of the array after removing duplicates. • **Example:** ○ Input: `[1, 1, 2, 2, 3]`		

	○ Output: `3` (the array becomes `[1, 2, 3]`)		
13	**Calculate Factorial Recursively** • **Description:** Write a recursive Java program to calculate the factorial of a given number. • **Input:** A single integer. • **Output:** The factorial of the integer. • **Example:** ○ Input: `5` ○ Output: `120` ○ Input: `0` ○ Output: `1`		
14	**Longest Common Subsequence** • **Description:** Write a Java program to find the length of the longest common subsequence between two strings. • **Input:** Two strings. • **Output:** The length of the longest common subsequence. • **Example:** ○ Input: `"abcde"`, `"ace"` ○ Output: `3` (LCS is `"ace"`)		
15	**Reverse a String Without Using Built-in Functions** • **Description:** Write a Java program to reverse a given string without using any built-in string manipulation functions. • **Input:** A single string. • **Output:** The reversed string. • **Example:** ○ Input: `"hello"` ○ Output: `"olleh"`		
16	**Check Armstrong Number** • **Description:** Write a Java program to check if a given number is an Armstrong number. An Armstrong number for a given number of digits is a number whose sum of its own digits each raised to the power of the number of digits is equal to the number itself. • **Input:** A single integer.		

	• **Output:** Print "Armstrong" if the number is an Armstrong number, otherwise print "Not Armstrong".
	• **Example:**
	  o Input: `153`
	  o Output: `Armstrong`		

17	**Count Vowels and Consonants in a String**
	
	• **Description:** Write a Java program to count the number of vowels and consonants in a given string.
	• **Input:** A single string.
	• **Output:** Print the count of vowels and consonants.
	• **Example:**
	  o Input: `"OpenAI"`
	  o Output: `Vowels: 3, Consonants: 3`		

18	**Find the GCD (Greatest Common Divisor) of Two Numbers**
	
	• **Description:** Write a Java program to find the greatest common divisor (GCD) of two given numbers using Euclid's algorithm.
	• **Input:** Two integers.
	• **Output:** The GCD of the two integers.
	• **Example:**
	  o Input: `48, 18`
	  o Output: `6`		

19	**Bubble Sort Implementation**
	
	• **Description:** Write a Java program to implement the bubble sort algorithm to sort an array of integers in ascending order.
	• **Input:** An unsorted array of integers.
	• **Output:** The sorted array.
	• **Example:**
	  o Input: `[64, 34, 25, 12, 22, 11, 90]`
	  o Output: `[11, 12, 22, 25, 34, 64, 90]`		

20	**Check if a String is a Valid Palindrome (Ignoring Spaces and Case)** • **Description:** Write a Java program to check if a given string is a palindrome, ignoring spaces and case. • **Input:** A single string.		

	• **Output:** Print "Palindrome" if the string is a palindrome, otherwise print "Not a Palindrome".
	• **Example:**
	  o Input: `"A man a plan a canal Panama"`
	  o Output: `Palindrome`		

21	**Find All Prime Numbers Up to a Given Number**
	
	• **Description:** Write a Java program to find all prime numbers up to a given number using the Sieve of Eratosthenes.
	• **Input:** A single integer `n`.
	• **Output:** A list of prime numbers less than or equal to `n`.
	• **Example:**
	  o Input: `30`
	  o Output: `[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]`		

22	**Remove Duplicates from a String**
	
	• **Description:** Write a Java program to remove duplicate characters from a given string.
	• **Input:** A single string.
	• **Output:** The string with duplicates removed.
	• **Example:**
	  o Input: `"programming"`
	  o Output: `"progamin"`		

23	**Find the Second Largest Element in an Array**
	
	• **Description:** Write a Java program to find the second largest element in an array of integers.
	• **Input:** An array of integers.
	• **Output:** The second largest element.
	• **Example:**
	  o Input: `[12, 35, 1, 10, 34, 1]`
	  o Output: `34`		

24	**Find the Longest Substring Without Repeating Characters** • **Description:** Write a Java program to find the length of the longest substring without repeating characters in a given string. • **Input:** A single string.		

	• **Output:** The length of the longest substring without repeating characters.
	• **Example:**
	  ○ Input: `"abcabcbb"`
	  ○ Output: `3` (substring is `"abc"`)		

25	**Calculate Power Without Using Built-in Functions**
	
	• **Description:** Write a Java program to calculate `x` raised to the power `y` (`x^y`) without using built-in math functions.
	• **Input:** Two integers `x` and `y`.
	• **Output:** The result of `x^y`.
	• **Example:**
	  ○ Input: `2, 3`
	  ○ Output: `8`		

26	**Merge Two Sorted Arrays**
	
	• **Description:** Write a Java program to merge two sorted arrays into a single sorted array.
	• **Input:** Two sorted arrays of integers.
	• **Output:** A merged sorted array.
	• **Example:**
	  ○ Input: `[1, 3, 5]`, `[2, 4, 6]`
	  ○ Output: `[1, 2, 3, 4, 5, 6]`		

27	**Check if Two Strings are Rotation of Each Other**
	
	• **Description:** Write a Java program to check if two strings are rotations of each other.
	• **Input:** Two strings.
	• **Output:** Print "Rotation" if one string is a rotation of the other, otherwise print "Not Rotation".
	• **Example:**
	  ○ Input: `"ABCD"`, `"CDAB"`
	  ○ Output: `Rotation`		

28	**Fibonacci Series Using Recursion** • **Description:** Write a Java program to print the first `n` Fibonacci numbers using recursion. • **Input:** An integer `n`. • **Output:** The first `n` Fibonacci numbers. • **Example:**		

	○ Input: `7` ○ Output: `[0, 1, 1, 2, 3, 5, 8]`		
29	**Check if a Number is Perfect** • **Description:** Write a Java program to check if a given number is a perfect number. A perfect number is a positive integer that is equal to the sum of its proper divisors, excluding itself. • **Input:** A single integer. • **Output:** Print "Perfect Number" if the number is perfect, otherwise print "Not a Perfect Number". • **Example:** ○ Input: `28` ○ Output: `Perfect Number`		
30	Java program that converts a given decimal number into its binary and octal equivalents. The program also includes input validation to ensure the user enters a valid non-negative integer. **Test Cases:** 1. **Test Case 1:** ○ **Input:** Decimal Number: 15 ○ **Output:** ▪ Binary Number = 1111 ▪ Octal Number = 17 2. **Test Case 2:** ○ **Input:** Decimal Number: 0 ○ **Output:** ▪ Binary Number = 0 ▪ Octal Number = 0 3. **Test Case 3:** ○ **Input:** Decimal Number: 255 ○ **Output:** ▪ Binary Number = 11111111 ▪ Octal Number = 377 4. **Test Case 4:** ○ **Input:** Decimal Number: -10 ○ **Output:** Please enter a non-negative integer. 5. **Test Case 5:** ○ **Input:** Decimal Number: abc ○ **Output:** Invalid input. Please enter a valid integer.		

31	**Student Grade Calculation** • **Description:** Write a program that takes the marks of five subjects and calculates the total, average, and grade of a student. The grade is determined as follows: `A` for average >= 90, `B` for average >= 80 and < 90, `C` for average >= 70 and < 80, `D` for average >= 60 and < 70, and `F` for average < 60. • **Test Cases:** 1. Marks: 95, 92, 88, 90, 91 (Expected Grade: A) 2. Marks: 75, 80, 72, 68, 74 (Expected Grade: C) 3. Marks: 60, 59, 62, 58, 57 (Expected Grade: F) 4. Marks: 85, 89, 90, 87, 84 (Expected Grade: B)		
32	**Electricity Bill Calculator** • **Description:** Write a program to calculate the electricity bill based on the units consumed. The charges per unit are as follows: o First 100 units: $1.50 per unit o Next 200 units: $2.00 per unit o Above 300 units: $3.00 per unit • Additionally, if the total bill exceeds $500, a surcharge of 10% is added. • **Test Cases:** 1. Units Consumed: 150 (Expected Bill: $275) 2. Units Consumed: 350 (Expected Bill: $675 + 10% surcharge = $742.50) 3. Units Consumed: 90 (Expected Bill: $135) 4. Units Consumed: 600 (Expected Bill: $1500 + 10% surcharge = $1650)		
33	**Library Management System** • **Description:** Write a program to manage a library system. The system should track books, which have a title, author, and availability status. It should allow users to borrow and return books, and it should ensure that books are not borrowed if they are already checked out. • **Test Cases:** 1. Borrow a book that is available. 2. Try to borrow a book that is already borrowed. 3. Return a book that was borrowed. 4. List all available books in the library.		

34	**Tax Calculator**
	
	• **Description:** Write a program to calculate income tax based on the following slab:
	  ○ Income up to $10,000: No tax
	  ○ Income from $10,001 to $20,000: 10% tax
	  ○ Income from $20,001 to $50,000: 20% tax
	  ○ Income above $50,000: 30% tax
	• The program should take the user's income as input and calculate the tax accordingly.
	• **Test Cases:**
	  1. Income: $9,000 (Expected Tax: $0)
	  2. Income: $15,000 (Expected Tax: $500)
	  3. Income: $35,000 (Expected Tax: $4,000)
	  4. Income: $70,000 (Expected Tax: $15,000)		

35	**Simple ATM System**
	
	• **Description:** Create a simple ATM system where users can check their balance, deposit money, and withdraw money. The system should ensure that withdrawals do not exceed the available balance and should update the balance after each transaction.
	• **Test Cases:**
	  1. Initial Balance: $5,000, Withdraw $1,000 (Expected Balance: $4,000)
	  2. Deposit $2,000 (Expected Balance: $6,000)
	  3. Try to withdraw $10,000 (Expected Output: Insufficient funds)
	  4. Check balance after transactions.		

36	**Payroll System**
	
	• **Description:** Develop a payroll system that calculates the net salary of employees. The salary should include base pay and various allowances (house rent allowance, dearness allowance) and deductions (tax, provident fund). The allowances are a percentage of the base pay, and the deductions are either fixed or percentage-based.
	• **Test Cases:**
	  1. Base Pay: $5,000, HRA: 20%, DA: 10%, Tax: 10%, PF: $500 (Expected Net Salary: $5,900)
	  2. Base Pay: $10,000, HRA: 25%, DA: 15%, Tax: 12%, PF: $1000 (Expected Net Salary: $11,200)		

	3. Calculate net salary for different combinations of allowances and deductions.		
37	**Online Shopping Cart** • **Description:** Implement an online shopping cart where users can add and remove items, view the total price of items in the cart, and apply discounts if applicable (e.g., a discount code). The program should handle different product prices and quantities. • **Test Cases:** 1. Add items to the cart and check the total. 2. Remove an item from the cart and update the total. 3. Apply a discount code (e.g., 10% off) and check the updated total. 4. Empty the cart and check that the total is $0.		
38	Write a Java program that safely handles the division of two numbers and specifically checks for division by zero. If division by zero is attempted, the program should catch the exception and provide an appropriate error message. **Test Cases:** 1. **Test Case 1: Valid Division** o **Input:** Numerator: 10, Denominator: 2 o **Expected Output:** Result: 5, Division operation complete. 2. **Test Case 2: Division by Zero** o **Input:** Numerator: 10, Denominator: 0 o **Expected Output:** Error: Division by zero is not allowed., Division operation complete. 3. **Test Case 3: Negative Numbers** o **Input:** Numerator: -20, Denominator: 4 o **Expected Output:** Result: -5, Division operation complete. 4. **Test Case 4: Zero Numerator** o **Input:** Numerator: 0, Denominator: 5 o **Expected Output:** Result: 0, Division operation complete. 5. **Test Case 5: Large Numbers** o **Input:** Numerator: 1000000, Denominator: 1000 o **Expected Output:** Result: 1000, Division operation complete.		

39	**Handling Multiple Exceptions** **Scenario:** Write a Java program that takes an array of integers from the user and an index position. The program should attempt to access the element at the given index and divide it by another number provided by the user. Handle the following exceptions: • `ArrayIndexOutOfBoundsException`: If the index is out of the array bounds. • `ArithmeticException`: If there is an attempt to divide by zero. • Any other general exception. **Test Case Scenarios:** 1. **Valid Input:** o **Input:** Array: [10, 20, 30], Index: 1, Divisor: 2 o **Expected Output:** Result: 10, Execution complete. 2. **Array Index Out of Bounds:** o **Input:** Array: [10, 20, 30], Index: 5, Divisor: 2 o **Expected Output:** Error: Index out of bounds, Execution complete. 3. **Division by Zero:** o **Input:** Array: [10, 20, 30], Index: 1, Divisor: 0 o **Expected Output:** Error: Division by zero, Execution complete. 4. **Negative Index:** o **Input:** Array: [10, 20, 30], Index: -1, Divisor: 2 o **Expected Output:** Error: Index out of bounds, Execution complete. 5. **Empty Array:** o **Input:** Array: [], Index: 0, Divisor: 1 o **Expected Output:** Error: Index out of bounds, Execution complete.		
40	**Using Throw and Throws** **Scenario:** Write a Java program to simulate an age validation system. The program should throw a custom `InvalidAgeException` if the user enters an age less than 18. Use `throw` to explicitly throw the exception and `throws` to declare it.		

	Test Case Scenarios:
	
	1. **Valid Age:**
	  - **Input:** Age: 20
	  - **Expected Output:** Valid age for registration.
	2. **Invalid Age (Below 18):**
	  - **Input:** Age: 16
	  - **Expected Output:** Caught exception: Invalid age: Must be 18 or older.
	3. **Boundary Case (Exactly 18):**
	  - **Input:** Age: 18
	  - **Expected Output:** Valid age for registration.
	4. **Another Valid Age:**
	  - **Input:** Age: 25
	  - **Expected Output:** Valid age for registration.
	5. **Multiple Invalid Ages:**
	  - **Input:** Ages: 14, 16
	  - **Expected Output:** Caught exception: Invalid age: Must be 18 or older. (for both)		

41	## Custom Exception
	
	Scenario:
	Write a Java program that simulates a bank account. The program should throw a custom `InsufficientFundsException` if a withdrawal amount exceeds the account balance. The program should also throw a custom `NegativeAmountException` if the withdrawal or deposit amount is negative.
	
	Test Case Scenarios:
	
	1. **Valid Deposit and Withdrawal:**
	  - **Input:** Deposit: 500, Withdrawal: 200
	  - **Expected Output:** Deposited: 500, Withdrew: 200, Current Balance: 1300
	2. **Insufficient Funds:**
	  - **Input:** Withdrawal: 1500
	  - **Expected Output:** Exception: Insufficient funds for withdrawal.
	3. **Negative Deposit:**
	  - **Input:** Deposit: -50
	  - **Expected Output:** Exception: Deposit amount cannot be negative.
	4. **Negative Withdrawal:**
	  - **Input:** Withdrawal: -100		

	- **Expected Output:** Exception: Withdrawal amount cannot be negative. 5. **Multiple Valid Transactions:** - **Input:** Deposit: 100, Withdrawal: 50, Withdrawal: 20 - **Expected Output:** Deposited: 100, Withdrew: 50, Withdrew: 20, Current Balance: 1030		
42	**Nested Try-Catch Blocks** **Scenario:** Write a Java program that demonstrates nested try-catch blocks, where the outer block handles a general exception and the inner block handles specific exceptions like `ArithmeticException` and `ArrayIndexOutOfBoundsException`. **Test Cases:** 1. **No Exception:** - **Array:** [10, 20, 2, 40] - **Output:** Result: 10, Execution continues... 2. **Arithmetic Exception:** - **Array:** [10, 20, 0, 40] - **Output:** Inner catch: ArithmeticException caught, Execution continues... 3. **Array Index Out of Bounds:** - **Array:** [10, 20, 30] - **Output:** Outer catch: ArrayIndexOutOfBoundsException caught, Execution continues...		
43	**Try-With-Resources for Automatic Resource Management** **Scenario:** Write a Java program that reads from a file using try-with-resources, ensuring that the file resource is closed automatically even if an exception occurs. **Test Cases:** 1. **Valid File Path:** - **File Content:** "Hello, World!" - **Output:** Hello, World! 2. **Invalid File Path:** - **File Path:** "nonexistent.txt"		

	○ **Output:** Error reading the file: nonexistent.txt (No such file or directory)		
44	**Custom Exception Chaining** **Scenario:** Write a Java program that demonstrates exception chaining, where a custom exception wraps another exception. **Test Cases:** 1. **Exception Chaining Demonstration:** ○ **Input:** "invalid" string ○ **Output:** Caught: Invalid input encountered in method1, Original cause: NumberFormatException		
45	**Using Finally Block** **Scenario:** Write a Java program to demonstrate the use of a `finally` block, which executes regardless of whether an exception is thrown or not. **Test Cases:** 1. **Exception Occurs:** ○ **Input:** Division by zero ○ **Output:** Exception caught, Finally block executed, Rest of the code. 2. **No Exception:** ○ **Input:** Valid division ○ **Output:** Result printed, Finally block executed, Rest of the code.		
46	**Rethrowing an Exception** **Scenario:** Write a Java program that catches an exception, processes it, and then rethrows it to be handled by another catch block or by the calling method. **Test Cases:** 1. **Rethrown Exception:** ○ **Input:** Division by zero		

	○ **Output:** Exception caught in method1, Exception rethrown and caught in main.		
47	**Java Program to Print the First N Perfect Numbers** **Scenario:** Write a Java program that prints the first `n` perfect numbers. A perfect number is a positive integer that is equal to the sum of its proper divisors (excluding itself). **Test Case Scenarios:** 1. **Valid Input - First 3 Perfect Numbers:** ○ **Input:** N = 3 ○ **Output:** First 3 perfect numbers are: 6, 28, 496 2. **Valid Input - First 5 Perfect Numbers:** ○ **Input:** N = 5 ○ **Output:** First 5 perfect numbers are: 6, 28, 496, 8128, 33550336 3. **Invalid Input - Zero or Negative Number:** ○ **Input:** N = 0 ○ **Output:** The number must be a positive integer. 4. **Large Input - First 1 Perfect Number:** ○ **Input:** N = 1 ○ **Output:** First 1 perfect number is: 6 5. **Edge Case - Input Greater Than 100:** ○ **Input:** N = 100 ○ **Output:** (The program will print the first 100 perfect numbers if they exist within a reasonable range, otherwise it will print available perfect numbers within limits.)		
48	**Find the Largest and Smallest Element in an Array** **Scenario:** Write a Java program to find the largest and smallest elements in an array of integers. **Test Case Scenarios:** 1. **Positive Numbers:** ○ **Input:** Array: [3, 5, 7, 2, 8] ○ **Output:** Largest element: 8, Smallest element: 2 2. **Negative Numbers:** ○ **Input:** Array: [-3, -5, -1, -8, -2]		

	○ **Output:** Largest element: -1, Smallest element: -8 3. **Mixed Numbers:** ○ **Input:** Array: [3, -5, 7, 2, -8] ○ **Output:** Largest element: 7, Smallest element: -8		
49	**Reverse a String** **Scenario:** Write a Java program to reverse a given string. **Test Case Scenarios:** 1. **Normal String:** ○ **Input:** "hello" ○ **Output:** "olleh" 2. **Empty String:** ○ **Input:** "" ○ **Output:** "" 3. **Palindrome String:** ○ **Input:** "racecar" ○ **Output:** "racecar" 4. **String with Special Characters:** ○ **Input:** "abc@123" ○ **Output:** "321@cba"		
50	**Calculate Fibonacci Series Up to N Terms** **Scenario:** Write a Java program to calculate and print the Fibonacci series up to `n` terms. **Test Case Scenarios:** 1. **Positive Number of Terms:** ○ **Input:** N = 5 ○ **Output:** 0 1 1 2 3 2. **Single Term:** ○ **Input:** N = 1 ○ **Output:** 0 3. **Zero Terms:** ○ **Input:** N = 0 ○ **Output:** The number of terms must be positive. 4. **Large Number of Terms (e.g., N = 10):** ○ **Input:** N = 10		

	○ **Output:** 0 1 1 2 3 5 8 13 21 34		
51	**Check for Palindrome** **Scenario:** Write a Java program to check if a given string is a palindrome. **Test Case Scenarios:** 1. **Normal Palindrome:** ○ **Input:** "Madam" ○ **Output:** The string is a palindrome. 2. **Not a Palindrome:** ○ **Input:** "Hello" ○ **Output:** The string is not a palindrome. 3. **Palindrome with Special Characters:** ○ **Input:** "A man, a plan, a canal, Panama" ○ **Output:** The string is a palindrome. 4. **Empty String:** ○ **Input:** "" ○ **Output:** The string is a palindrome.		
52	**Sort an Array Using Bubble Sort** **Scenario:** Write a Java program to sort an array using the bubble sort algorithm. **Test Case Scenarios:** 1. **Unsorted Array:** ○ **Input:** Array: [5, 2, 9, 1, 5, 6] ○ **Output:** Sorted array: 1 2 5 5 6 9 2. **Already Sorted Array:** ○ **Input:** Array: [1, 2, 3, 4, 5] ○ **Output:** Sorted array: 1 2 3 4 5 3. **Array with Duplicates:** ○ **Input:** Array: [3, 5, 2, 2, 8] ○ **Output:** Sorted array: 2 2 3 5 8 4. **Single Element Array:** ○ **Input:** Array: [10] ○ **Output:** Sorted array: 10 5. **Empty Array:** ○ **Input:** Array: [] ○ **Output:** Sorted array: (no output)		

53	**Merge Two Sorted Arrays** **Scenario:** Write a Java program to merge two sorted arrays into a single sorted array. **Test Case Scenarios:** 1. **Basic Merge:** o **Input:** `array1 = {1, 3, 5}`, `array2 = {2, 4, 6}` o **Output:** Merged array: `[1, 2, 3, 4, 5, 6]` 2. **Empty Array:** o **Input:** `array1 = {}`, `array2 = {1, 2, 3}` o **Output:** Merged array: `[1, 2, 3]` 3. **One Empty Array:** o **Input:** `array1 = {1, 2, 3}`, `array2 = {}` o **Output:** Merged array: `[1, 2, 3]`		
54	**Implement Quick Sort** **Scenario:** Write a Java program to sort an array using the Quick Sort algorithm. **Test Case Scenarios:** 1. **Basic Array:** o **Input:** `array = {10, 7, 8, 9, 1, 5}` o **Output:** Sorted array: `[1, 5, 7, 8, 9, 10]` 2. **All Elements Same:** o **Input:** `array = {5, 5, 5, 5}` o **Output:** Sorted array: `[5, 5, 5, 5]` 3. **Reversed Array:** o **Input:** `array = {9, 8, 7, 6, 5}` o **Output:** Sorted array: `[5, 6, 7, 8, 9]`		
55	**Find All Prime Numbers Up to N** **Scenario:** Write a Java program to find all prime numbers up to a given number `n` using the Sieve of Eratosthenes algorithm. **Test Case Scenarios:** 1. **Small Number:**		

	○ **Input:** `n = 10` ○ **Output:** Prime numbers up to 10: `2 3 5 7` 2. **Large Number:** ○ **Input:** `n = 50` ○ **Output:** Prime numbers up to 50: `2 3 5 7 11 13 17 19 23 29 31 37 41 43 47` 3. **No Prime Numbers:** ○ **Input:** `n = 1` ○ **Output:** There are no prime numbers less than 2.		
56	**Find the Longest Common Prefix** **Scenario:** Write a Java program to find the longest common prefix string amongst an array of strings. **Test Case Scenarios:** 1. **Common Prefix:** ○ **Input:** `{"flower", "flow", "flight"}` ○ **Output:** Longest common prefix: "fl" 2. **No Common Prefix:** ○ **Input:** `{"dog", "racecar", "car"}` ○ **Output:** Longest common prefix: "" 3. **All Identical Strings:** ○ **Input:** `{"same", "same", "same"}` ○ **Output:** Longest common prefix: "same"		
57	**Find the Kth Largest Element in an Array** **Scenario:** Write a Java program to find the Kth largest element in an unsorted array. **Test Case Scenarios:** • **Input:** ○ Array: [3, 2, 1, 5, 6, 4] ○ K: 2 • **Output:** ○ The 2nd largest element is: 5 *2. K Greater Than Array Size* • **Input:** ○ Array: [1, 2] ○ K: 3 • **Output:**		

	- The 3rd largest element does not exist. (Program can be modified to handle this case by returning a specific value or throwing an exception.) *3. K Equals 1* **Input:** - Array: [7, 10, 4, 3, 20, 15] - K: 1 **Output:** - The 1st largest element is: 20 *4. All Elements Same* **Input:** - Array: [5, 5, 5, 5] - K: 2 **Output:** - The 2nd largest element is: 5 *5. Array with Negative Numbers* **Input:** - Array: [-1, -3, -2, -5, -4] - K: 3 **Output:** - The 3rd largest element is: -3		
58	**Basic Thread Creation** *Question:* Write a Java program to create and start two threads. Each thread should print a message indicating which thread it is, and the current thread ID. Use both Thread class and Runnable interface approaches. *Test Case Scenarios:* **General Case:** **Input:** No input required. **Expected Output** **Thread is running: [Thread ID 1]** **Thread is running: [Thread ID 2]**		
59	**Thread Synchronization** *Question:* Write a Java program with a class Counter that has a synchronized method increment() to increment a count. Create two threads that each increment the count 1000 times. Print the final count value.		

	Test Case Scenarios: **General Case:** • **Input:** No input required. • **Expected Output:** Final count: 2000		
60	**Scheduled Tasks** *Question:* Write a Java program to use `ScheduledExecutorService` to schedule a task that prints the current time every 2 seconds. Schedule another task to shut down the scheduler after 10 seconds. *Test Case Scenarios:* **General Case:** • **Input:** No input required. • **Expected Output:** **Current time: [Time 1]** **Current time: [Time 2]** **...** **Scheduler shutdown.**		