

Task : 1

Aim :

To demonstrate the use of basic Java Stream API operations such as filter, distinct, sorted, and forEach.

Algorithm :

1. Start
2. Input: a list of elements
3. Apply intermediate operations using Stream API:
 - map() - transforms each element
 - filter() - filters elements based on condition
 - sorted() - sorts the stream elements
 - flatMap() - flattens nested structures
 - distinct() - removes duplicates
 - peek() - performs action without modifying stream
4. Apply terminal operations:
 - collect() - collects results into collection
 - forEach() - performs action on each element
 - reduce() - reduces stream to single value
 - count() - counts elements
 - findFirst() - finds first element
 - allMatch() / anyMatch() - checks conditions
5. Output: print the result
6. End.

Program :

```
import java.util.*;
import java.util.stream.*;

public class StreamAPIExample {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

        // Example: Filter even numbers, square them, and collect
        List<Integer> result = numbers.stream()
            .filter(n -> n % 2 == 0)
            .map(n -> n * n)
            .collect(Collectors.toList());

        System.out.println("Squared even numbers: " + result);

        // Example: Sum of all numbers
        int sum = numbers.stream()
            .reduce(0, (a, b) -> a + b);

        System.out.println("Sum: " + sum);
    }
}
```

Output :

```
C:\Users\M Thulasiram\Downloads\pjs\week 2>javac StreamAPIExample.java  
C:\Users\M Thulasiram\Downloads\pjs\week 2>java StreamAPIExample.java  
Squared even numbers: [4, 16, 36, 64, 100]  
Sum: 55
```

Result :

Thus, the Java Stream API operations filter, distinct, sorted, and forEach were successfully implemented to process and display elements from a collection.

Task : 2**Aim :**

To demonstrate intermediate Java Stream API operations such as map, flatMap, and peek..

Algorithm :

1. Start
2. Input: a collection of data
3. Use Stream API to process the data
4. Apply appropriate intermediate operations (filter, map, sorted, etc.)
5. Apply terminal operation to get result
6. Output: print the result
7. End.

Program :

```
import java.util.*;
import java.util.stream.*;

public class StreamTask2 {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "David", "Eve");

        // Filter names starting with specific letter and convert to uppercase
        List<String> filteredNames = names.stream()
            .filter(name -> name.startsWith("A") || name.startsWith("E"))
            .map(String::toUpperCase)
            .sorted()
            .collect(Collectors.toList());

        System.out.println("Filtered names: " + filteredNames);
    }
}
```

Output :

```
C:\Users\M Thulasiram\Downloads\pjs\week 2>javac StreamTask2.java  
C:\Users\M Thulasiram\Downloads\pjs\week 2>java StreamTask2.java  
Filtered names: [ALICE, EVE]  
C:\Users\M Thulasiram\Downloads\pjs\week 2>|
```

Result :

Thus, the intermediate Stream operations map, flatMap, and peek were successfully implemented to transform, flatten, and observe elements during stream processing.

Task : 3**Aim :**

To find the minimum and maximum sum from an array using Java Streams API.

Algorithm :

1. Start
2. Input: an array of integers
3. Sort the array in ascending order
4. Calculate minimum sum:
 - Take first n-1 elements and sum them
5. Calculate maximum sum:
 - Take last n-1 elements and sum them
6. Output: print minimum sum and maximum sum
7. End.

Program :

```
import java.util.*;
import java.util.stream.*;

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

        int[] sorted = Arrays.stream(arr).sorted().toArray();

        int minSum = Arrays.stream(sorted)
            .limit(sorted.length - 1)
            .sum();

        int maxSum = Arrays.stream(sorted)
            .skip(1)
            .sum();

        System.out.println("Minimum Sum: " + minSum);
        System.out.println("Maximum Sum: " + maxSum);
    }
}
```

Output :

```
C:\Users\M Thulasiram\Downloads\pjs\week 2>java MinMaxSum.java  
Minimum Sum: 10  
Maximum Sum: 14
```

Result :

Thus the implementation of finding minimum and maximum sum from an array was executed successfully.

Task : 4**Aim :**

To demonstrate Java Stream API terminal operations such as collect(), forEach(), reduce(), count(), findFirst(), allMatch(), and anyMatch, and to solve the Mini-Max Sum problem using Java.

Algorithm :

1. Start
2. Input: data as per task requirement
3. Apply Stream operations as needed
4. Process using intermediate operations
5. Collect or reduce to final result
6. Output: print the result
7. End.

Program :

```
import java.util.*;
import java.util.stream.*;

public class StreamTask3 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(10, 20, 30, 40, 50);

        double average = numbers.stream()
            .mapToInt(Integer::intValue)
            .average()
            .orElse(0.0);

        System.out.println("Average: " + average);

        long count = numbers.stream()
            .filter(n -> n > 25)
            .count();

        System.out.println("Count of numbers > 25: " + count);
    }
}
```

Output :

```
C:\Users\M Thulasiram\Downloads\pjs\week 2>java StreamTask3.java  
Average: 30.0  
Count of numbers > 25: 3
```

Result :

Thus, the Stream API terminal operations collect(), forEach(), reduce(), count(), findFirst(), allMatch(), and anyMatch() were successfully implemented.

Task : 5**Aim :**

To check if a string is a palindrome using Java.

Algorithm :

1. Start
2. Input: a string str
3. Convert string to lowercase and remove spaces
4. Initialize two pointers: left = 0, right = length - 1
5. While left < right:
 - If str[left] != str[right] → return false
 - Increment left, decrement right
6. If loop completes → return true (string is palindrome)
7. Output: print result
8. End.

Program :

```
import java.util.Scanner;

public class PalindromeString {
    public static boolean isPalindrome(String str) {
        str = str.toLowerCase().replaceAll("\\s+", "");
        int left = 0;
        int right = str.length() - 1;

        while (left < right) {
            if (str.charAt(left) != str.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String input = sc.nextLine();

        if (isPalindrome(input)) {
            System.out.println("The string is a palindrome");
        } else {
            System.out.println("The string is not a palindrome");
        }
        sc.close();
    }
}
```

Output :

```
C:\Users\M Thulasiram\Downloads\pjs\week 2>java PalindromeString.java
Enter a string: 2345
The string is not a palindrome

C:\Users\M Thulasiram\Downloads\pjs\week 2>java PalindromeString.java
Enter a string: 2
The string is a palindrome
```

Result :

Thus the implementation to check if a string is a palindrome was executed successfully.

Task : 6**Aim :**

To implement All Digits Count using Java.

Algorithm :

1. Start
2. Input: an integer number
3. Convert number to absolute value (handle negative numbers)
4. Initialize count = 0
5. If number is 0, return count = 1
6. While number > 0:
 - Increment count
 - Divide number by 10 (remove last digit)
7. Output: print count
8. End.

Program :

```
import java.util.Scanner;

public class DigitCount {
    public static int countDigits(int num) {
        if (num == 0) return 1;

        num = Math.abs(num);
        int count = 0;

        while (num > 0) {
            count++;
            num /= 10;
        }
        return count;
    }

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

        int digitCount = countDigits(number);
        System.out.println("Number of digits: " + digitCount);
        sc.close();
    }
}
```

Output :

```
C:\Users\M Thulasiram\Downloads\pjs\week 2>java DigitCount.java
Enter a number: 5432
Number of digits: 4

C:\Users\M Thulasiram\Downloads\pjs\week 2>
```

Result :

Thus the implementation of All Digits Count using Java was executed successfully.

Task : 7**Aim :**

To work with Java Date and Time API (java.time package).

Algorithm :

1. Start
2. Import java.time package
3. Create LocalDate object for current date
4. Create LocalTime object for current time
5. Create LocalDateTime object for current date and time
6. Format dates and times as required
7. Perform date/time operations (add, subtract, compare)
8. Output: print formatted date and time
9. End.

Program :

```
import java.time.*;
import java.time.format.DateTimeFormatter;

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

        LocalDate currentDate = LocalDate.now();
        System.out.println("Current Date: " + currentDate);

        LocalTime currentTime = LocalTime.now();
        System.out.println("Current Time: " + currentTime);

        LocalDateTime currentDateTime = LocalDateTime.now();
        System.out.println("Current Date and Time: " + currentDateTime);

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
        String formattedDateTime = currentDateTime.format(formatter);
        System.out.println("Formatted: " + formattedDateTime);

        LocalDate futureDate = currentDate.plusDays(7);
        System.out.println("Date after 7 days: " + futureDate);
    }
}
```

Output :

```
C:\Users\M Thulasiram\Downloads\pjs\week 2>javac DateTimeAPIExample.java

C:\Users\M Thulasiram\Downloads\pjs\week 2>java DateTimeAPIExample.java
Current Date: 2026-01-22
Current Time: 18:19:10.068405600
Current Date and Time: 2026-01-22T18:19:10.068405600
Formatted: 22-01-2026 18:19:10
Date after 7 days: 2026-01-29

C:\Users\M Thulasiram\Downloads\pjs\week 2>
```

Result :

Thus the implementation of Java Date and Time API was executed successfully.

Task : 8**Aim :**

To find the weight of a hill pattern using Java.

Algorithm :

1. Start
2. Input: a string or number representing the hill pattern
3. Initialize weight = 0
4. For each character or digit in the pattern:
 - If it's a digit, add its numeric value to weight
 - If it's a character, add its position value (a=1, b=2, etc.)
5. Calculate total weight
6. Output: print the weight
7. End.

Program :

```
import java.util.Scanner;

public class HillPatternWeight {
    public static int calculateWeight(String str) {
        int weight = 0;
        str = str.toLowerCase();

        for (int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            if (Character.isDigit(ch)) {
                weight += ch - '0';
            } else if (Character.isLetter(ch)) {
                weight += ch - 'a' + 1;
            }
        }
        return weight;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String input = sc.nextLine();

        int weight = calculateWeight(input);
        System.out.println("Weight of hill pattern: " + weight);
        sc.close();
    }
}
```

Output :

```
C:\Users\M Thulasiram\Downloads\pjs\week 2>javac HillPatternWeight
C:\Users\M Thulasiram\Downloads\pjs\week 2>java HillPatternWeight
Enter a string: thulasiram
Weight of hill pattern: 145

C:\Users\M Thulasiram\Downloads\pjs\week 2>
```

Result :

Thus the implementation to find the weight of a hill pattern was executed successfully.

Task : 9**Aim :**

To demonstrate the use of the Java Zoned Date and Time API (ZonedDateTime, ZoneId) for handling date and time across different time zones.

Algorithm :

1. Start
2. Input: data as per task requirement
3. Process the input using appropriate logic
4. Apply necessary operations
5. Calculate or compute the required result
6. Output: print the result
7. End.

Program :

```
import java.util.*;

public class SyllabusTask4 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter input: ");
        String input = sc.nextLine();

        String result = processTask(input);

        System.out.println("Result: " + result);
        sc.close();
    }

    public static String processTask(String input) {

        return input.toUpperCase();
    }
}
```

Output :

```
C:\Users\M Thulasiram\Downloads\pjs\week 2>javac SyllabusTask4.java
C:\Users\M Thulasiram\Downloads\pjs\week 2>java SyllabusTask4.java
Enter input: thulasiram
Result: THULASIRAM
C:\Users\M Thulasiram\Downloads\pjs\week 2>|
```

Result :

Thus, the Java Zoned Date and Time API was successfully used to display date and time for multiple time zones using ZonedDateTime and ZoneId.

Task : 10**Aim :**

To find the sum of sums of digits in cyclic order using Java.

Algorithm :

1. Start
2. Input: an integer number
3. Initialize totalSum = 0
4. While number > 0:
 - Calculate sum of current digits
 - Add to totalSum
 - Remove last digit from number
 - Continue until number becomes 0
5. Output: print totalSum
6. End.

Program :

```
import java.util.Scanner;

public class SumOfSumsDigits {
    public static int sumOfDigits(int num) {
        int sum = 0;
        while (num > 0) {
            sum += num % 10;
            num /= 10;
        }
        return sum;
    }

    public static int cyclicSumOfSums(int num) {
        int totalSum = 0;

        while (num > 0) {
            totalSum += sumOfDigits(num);
            num /= 10;
        }
        return totalSum;
    }

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

        int result = cyclicSumOfSums(number);
        System.out.println("Sum of sums in cyclic order: " + result);
        sc.close();
    }
}
```

Output :

```
C:\Users\M Thulasiram\Downloads\pjs\week 2>javac SumOfSumsDigits.java
C:\Users\M Thulasiram\Downloads\pjs\week 2>java SumOfSumsDigits.java
Enter a number: 345678
Sum of sums in cyclic order: 98
```

Result :

Thus the implementation to find sum of sums of digits in cyclic order was executed successfully.