**Question 1: Grouping By Length and Counting**

Given a list of strings, write a Java 8 Stream program to group the strings by their length and count the number of strings in each group.

**Input:**

List<String> words = Arrays.asList("apple", "banana", "cherry", "date", "fig", "grape");

Answer

import java.util.\*;  
import java.util.function.Function;  
import java.util.stream.Collectors;  
  
public class GroupByLengthAndCount {  
 public static void main(String[] args) {  
 List<String> words = Arrays.asList("apple", "banana", "cherry", "date", "fig", "grape");  
  
 // Grouping by length and counting occurrences in each group  
 Map<Integer, Long> groupedByLength = words.stream()  
 .collect(Collectors.groupingBy(String::length, Collectors.counting()));  
  
 // Printing the result  
 System.out.println(groupedByLength);  
 }  
}

# Explanation of Each Method:

1. **words.stream()**:

* This converts the List<String> into a stream. The stream allows us to process elements of the list using stream operations such as map, filter, collect, etc.

**2. Collectors.groupingBy(String::length)**:

* groupingBy is a collector that groups elements by a classifier function. In this case, the classifier is String::length, which groups the strings based on their lengths.
* The result is a Map<Integer, List<String>>, where the keys are the string lengths and the values are lists of strings of that length.

**3. Collectors.counting()**:

* This is a downstream collector that counts the number of elements in each group. It is used in conjunction with groupingBy to get the count of strings for each length.
* The result is a Map<Integer, Long>, where the keys are the string lengths and the values are the counts of strings of that length.

**4. System.out.println(groupedByLength)**:

* This simply prints the resulting map, which contains the grouped data with the count for each length.

# Question 2: Partitioning By Even and Odd

Given a list of integers, write a Java 8 Stream program to partition the numbers into two groups: even and odd.

**Input:**

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

**Expected Output:**

{  
 false: [1, 3, 5, 7, 9], // Odd numbers  
 true: [2, 4, 6, 8, 10] // Even numbers  
}

import java.util.\*;  
import java.util.stream.Collectors;  
  
public class PartitionByEvenOdd {  
 public static void main(String[] args) {  
 List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);  
  
 // Partitioning by even and odd  
 Map<Boolean, List<Integer>> partitioned = numbers.stream()  
 .collect(Collectors.partitioningBy(n -> n % 2 == 0));  
  
 // Printing the result  
 System.out.println(partitioned);  
 }  
}

**Explanation of Each Method:**

1. **numbers.stream()**:

* This converts the List<Integer> into a stream, allowing us to process the elements using stream operations like filter, map, collect, etc.

**2. Collectors.partitioningBy(n -> n % 2 == 0)**:

* partitioningBy is a special collector that partitions the elements into two groups based on a predicate.
* The predicate here is n -> n % 2 == 0, which checks if the number is even. This results in two groups:
* true for even numbers.
* false for odd numbers.
* The result is a Map<Boolean, List<Integer>>, where the key is either true or false (even or odd), and the value is a list of integers that satisfy the corresponding condition.

3. **System.out.println(partitioned)**:

* This simply prints the resulting map, which contains the partitioned lists for even and odd numbers.

# Question 3: Find Second Largest Number

Given a list of integers, write a Java 8 Stream program to find the second largest number in the list.

**Input:**

List<Integer> numbers = Arrays.asList(10, 20, 5, 30, 15);

import java.util.\*;  
import java.util.stream.Collectors;  
  
public class SecondLargestNumber {  
 public static void main(String[] args) {  
 List<Integer> numbers = Arrays.asList(10, 20, 5, 30, 15);  
  
 // Finding the second largest number  
 Optional<Integer> secondLargest = numbers.stream()  
 .distinct() // Remove duplicates (if any)  
 .sorted(Comparator.reverseOrder()) // Sort in descending order  
 .skip(1) // Skip the first (largest) element  
 .findFirst(); // Get the second element  
  
 // Printing the result  
 secondLargest.ifPresent(System.out::println);  
 }  
}

**Explanation of Each Method:**

1. **numbers.stream()**

* Converts the List<Integer> into a Stream.

**2. distinct()**

* Ensures that duplicate values are removed, so we don’t get duplicate largest values.

**3. sorted(Comparator.reverseOrder())**

* Sorts the numbers in **descending order**, ensuring the largest number appears first.

**4. skip(1)**

* Skips the **first element** (which is the largest number), so that the next element in the sorted order is the second largest.

**5. findFirst()**

* Retrieves the first element after skipping, which is the **second largest number**.

**6. secondLargest.ifPresent(System.out::println);**

* Prints the second largest number if it exists

# Question 4: Custom Collector (Concatenate Strings)

Given a list of strings, write a Java 8 Stream program to concatenate all the strings into a single string, separated by a comma.

**Input:**

List<String> words = Arrays.asList("apple", "banana", "cherry", "date");

**Expected Output:**

"apple,banana,cherry,date"

import java.util.\*;  
import java.util.stream.Collectors;  
  
public class JoinStrings {  
 public static void main(String[] args) {  
 List<String> words = Arrays.asList("apple", "banana", "cherry", "date");  
  
 // Joining the list elements into a comma-separated string  
 String result = words.stream()  
 .collect(Collectors.joining(","));  
  
 // Printing the result  
 System.out.println(result);  
 }  
}

**Explanation of Each Method:**

**1. words.stream()**

* Converts the List<String> into a Stream.

**2. collect(Collectors.joining(","))**

* Uses Collectors.joining() to concatenate the elements of the stream.
* The **comma ,** is used as a delimiter between the words.

**3. System.out.println(result);**

* Prints the final concatenated string.

# Question 5: Find Longest String

Given a list of strings, write a Java 8 Stream program to find the longest string in the list.

**Input:**

List<String> words = Arrays.asList("apple", "banana", "cherry", "date");

import java.util.\*;  
import java.util.stream.Collectors;  
  
public class LongestStringFinder {  
 public static void main(String[] args) {  
 List<String> words = Arrays.asList("apple", "banana", "cherry", "date");  
  
 // Finding the longest string  
 Optional<String> longestString = words.stream()  
 .max(Comparator.comparingInt(String::length));  
  
 // Printing the result  
 longestString.ifPresent(System.out::println);  
 }  
}

**Explanation of Each Method:**

1. **words.stream()**

* Converts the List<String> into a Stream.

**2. max(Comparator.comparingInt(String::length))**

* Uses max() to find the string with the **maximum length**.
* The comparator Comparator.comparingInt(String::length) compares strings based on their length.

**3. Optional<String> longestString**

* Since max() returns an Optional, we handle it safely.

**4. longestString.ifPresent(System.out::println);**

* If a longest string exists, it gets printed.

# Question 6: Grouping By First Letter

Given a list of strings, write a Java 8 Stream program to group the strings by their first letter.

**Input:**

List<String> words = Arrays.asList("apple", "banana", "cherry", "date", "fig", "grape");

Expected output:

{  
 'a': ["apple"],  
 'b': ["banana"],  
 'c': ["cherry"],  
 'd': ["date"],  
 'f': ["fig"],  
 'g': ["grape"]  
}

import java.util.\*;  
import java.util.stream.Collectors;  
  
public class GroupByFirstLetter {  
 public static void main(String[] args) {  
 List<String> words = Arrays.asList("apple", "banana", "cherry", "date", "fig", "grape");  
  
 // Grouping by first letter  
 Map<Character, List<String>> groupedWords = words.stream()  
 .collect(Collectors.groupingBy(word -> word.charAt(0)));  
  
 // Printing the result  
 System.out.println(groupedWords);  
 }  
}