In Java, filtering, sorting, and mapping are common operations performed on collections or streams of data. These operations can be achieved using various methods available in the **Stream** class or using the functional programming features introduced in Java 8. Here's an overview of how to perform filtering, sorting, and mapping in Java:

Filtering: Filtering allows you to select elements from a collection or stream based on a specified condition. In Java, you can use the **filter()** method to filter elements based on a predicate. The **filter()** method takes a predicate as an argument, and it returns a new stream containing only the elements that satisfy the predicate.

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

List<Integer> evenNumbers = numbers.stream()

.filter(n -> n % 2 == 0)

.collect(Collectors.toList());

In this example, the **filter()** method is used to select only the even numbers from the list.

Sorting: Sorting allows you to arrange elements in a specific order. In Java, you can use the **sorted()** method to sort elements in a stream. The **sorted()** method returns a new stream with the elements sorted according to their natural order or a provided comparator.

Example:

List<Integer> numbers = Arrays.asList(5, 2, 4, 1, 3);

List<Integer> sortedNumbers = numbers.stream()

.sorted()

.collect(Collectors.toList());

In this example, the **sorted()** method is used to sort the numbers in ascending order.

Mapping: Mapping allows you to transform elements from one type to another. In Java, you can use the **map()** method to perform mapping operations. The **map()** method takes a function as an argument and applies the function to each element in the stream, returning a new stream containing the results of the function.

Example:

List<String> names = Arrays.asList("John", "Jane", "Mike");

List<Integer> nameLengths = names.stream()

.map(String::length)

.collect(Collectors.toList());

In this example, the **map()** method is used to transform each string element into its corresponding length.

These are some basic examples of filtering, sorting, and mapping operations in Java. By combining these operations with other stream operations, you can perform more complex data transformations and manipulations on collections or streams of data.

Terminal operations in Java streams are operations that produce a result or a side effect when applied to a stream. Terminal operations are the final step in a stream pipeline and trigger the processing of the stream elements. Here are some commonly used terminal operations in Java streams:

**forEach()**: Performs an action for each element in the stream. It takes a consumer as an argument and applies it to each element. It doesn't produce any result.

Example:

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

numbers.stream()

.forEach(System.out::println);

**collect()**: Aggregates the elements of a stream into a collection or a single value. It takes a collector as an argument that specifies how to accumulate elements.

Example:

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

List<Integer> evenNumbers = numbers.stream()

.filter(n -> n % 2 == 0)

.collect(Collectors.toList());

**count()**: Returns the count of elements in the stream as a **long** value.

Example:

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

long count = numbers.stream()

.filter(n -> n % 2 == 0)

.count();

**findFirst()** and **findAny()**: Returns the first element or any element of the stream, respectively, as an **Optional**. They are useful when you want to retrieve an element from a stream.

Example:

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

Optional<Integer> firstEvenNumber = numbers.stream()

.filter(n -> n % 2 == 0)

.findFirst();

**min()** and **max()**: Returns the minimum or maximum element of the stream based on a comparator, respectively. They return an **Optional** containing the result.

Example:

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

Optional<Integer> maxNumber = numbers.stream()

.max(Comparator.naturalOrder());