Java Stream Operations

Intermediate Operations

INTERMEDIATE OPERATIONS (Lazy until terminal op is called)

- filter() : Filters elements based on a predicate (like 'if')
- map() : Transforms each element
- flatMap() : Flattens nested streams
- sorted() : Sorts the stream
- distinct() : Removes duplicates
- limit(n) : Takes only the first 'n' elements
- skip(n) : Skips the first 'n' elements
- peek() : For debugging/side-effects (like .forEach() but lazy)

Example:

```
names.stream()
    .filter(name -> name.startsWith("I"))
    .map(String::toUpperCase)
    .sorted()
    .forEach(System.out::println);
```

Terminal Operations

TERMINAL OPERATIONS (Eager - they execute the pipeline)

- forEach() : Performs an action for each element
- collect() : Reduces the stream to a collection or string
- toArray() : Converts stream to an array
- reduce() : Aggregates elements (e.g. sum)
- count() : Returns the number of elements
- min(), max() : Returns min/max with comparator
- anyMatch() : Returns true if any match the predicate

Java Stream Operations

- allMatch() : Returns true if all match the predicate
- noneMatch() : Returns true if none match the predicate
- findFirst() : Returns the first element (Optional)
- findAny() : Returns any element (Optional)

Example:

```
long count = names.stream()
                .filter(n -> n.length() > 5)
                .count();
```

Example Pipeline

BONUS: Stream Pipeline Example

```
List<Integer> numbers = List.of(1, 2, 3, 4, 5, 6);

int sumOfSquaresOfEvenNumbers = numbers.stream()
    .filter(n -> n % 2 == 0)
    .map(n -> n * n)
    .reduce(0, Integer::sum);

System.out.println(sumOfSquaresOfEvenNumbers); // Output: 56
```