Operation	Return Type	Type Of Operation	What It Does?
filter()	Stream<T>	Intermediate	Returns a stream of elements which satisfy the given predicate.
map()	Stream<R>	Intermediate	Returns a stream consisting of results after applying given function to elements of the stream.
distinct()	Stream<T>	Intermediate	Returns a stream of unique elements.
sorted()	Stream<T>	Intermediate	Returns a stream consisting of elements sorted according to natural order.
limit()	Stream<T>	Intermediate	Returns a stream containing first *n* elements.
skip()	Stream<T>	Intermediate	Returns a stream after skipping first *n* elements.
forEach()	void	Terminal	Performs an action on all elements of a stream.
toArray()	Object[]	Terminal	Returns an array containing elements of a stream.
reduce()	type T	Terminal	Performs reduction operation on elements of a stream using initial value and binary operation.
collect()	Container of type R	Terminal	Returns mutable result container such as List or Set.
min()	Optional<T>	Terminal	Returns minimum element in a stream wrapped in an Optional object.
max()	Optional<T>	Terminal	Returns maximum element in a stream wrapped in an Optional object.
count()	long	Terminal	Returns the number of elements in a stream.
anyMatch()	boolean	Terminal	Returns true if any one element of a stream matches with given predicate.
allMatch()	boolean	Terminal	Returns true if all the elements of a stream matches with given predicate.
noneMatch()	boolean	Terminal	Returns true only if all the elements of a stream doesn't match with given predicate.
findFirst()	Optional<T>	Terminal	Returns first element of a stream wrapped in an Optional object.
findAny()	Optional<T>	Terminal	Randomly returns any one element in a stream.

Intermediate Operations

Intermediate-Operationen verändern den Stream und geben ebenfalls einen Stream zurück. Sie können miteinander verkettet werden und werden erst ausgeführt, wenn eine terminale Operation aufgerufen wird.

1. **filter(Predicate)**:
    - Gibt einen Stream von Elementen zurück, die die gegebene Bedingung (Prädikat) erfüllen.
    - Beispiel: `stream.filter(x -> x > 5)` gibt alle Elemente größer als 5 zurück.
2. **map(Function)**:
    - Wandelt jedes Element des Streams basierend auf der angegebenen Funktion in ein anderes Objekt um und gibt einen neuen Stream mit den Ergebnissen zurück.
    - Beispiel: `stream.map(x -> x * 2)` verdoppelt alle Elemente im Stream.
3. **distinct()**:
    - Entfernt doppelte Elemente und gibt einen Stream mit einzigartigen Elementen zurück.
    - Beispiel: `stream.distinct()` filtert doppelte Werte heraus.
4. **sorted()**:
    - Gibt einen Stream mit den Elementen in natürlicher oder angegebener Reihenfolge zurück.
    - Beispiel: `stream.sorted()` sortiert die Elemente in aufsteigender Reihenfolge.
5. **limit(long maxSize)**:
    - Begrenzung der Anzahl der Elemente im Stream auf eine maximale Anzahl.
    - Beispiel: `stream.limit(3)` gibt die ersten drei Elemente zurück.
6. **skip(long n)**:
    - Überspringt die ersten `n` Elemente und gibt den restlichen Stream zurück.
    - Beispiel: `stream.skip(2)` überspringt die ersten zwei Elemente.

Terminal Operations

Terminal-Operationen beenden die Verarbeitung eines Streams und geben entweder ein Ergebnis zurück oder führen eine Aktion aus.

7. **forEach(Consumer)**:
    - Führt eine Aktion für jedes Element im Stream aus.
    - Beispiel: `stream.forEach(System.out::println)` gibt jedes Element in der Konsole aus.
8. **toArray()**:
    - Konvertiert den Stream in ein Array und gibt es zurück.
    - Beispiel: `stream.toArray()` gibt ein Array mit den Elementen des Streams zurück.

9. **reduce(BinaryOperator)**:
    - Führt eine Reduktionsoperation auf den Elementen im Stream durch und gibt ein einzelnes Ergebnis zurück.
    - Beispiel: `stream.reduce(0, Integer::sum)` summiert alle Elemente.
10. **collect(Collector)**:
    - Sammelt die Elemente im Stream in einer angegebenen Datenstruktur (z.B. Liste, Set).
    - Beispiel: `stream.collect(Collectors.toList())` sammelt alle Elemente in einer Liste.
11. **min(Comparator)** und **max(Comparator)**:
    - Finden das Minimum oder Maximum eines Streams basierend auf einem angegebenen Kriterium und geben das Ergebnis in einem `Optional`-Objekt zurück.
    - Beispiel: `stream.min(Integer::compareTo)` findet das kleinste Element.
12. **count()**:
    - Gibt die Anzahl der Elemente im Stream zurück.
    - Beispiel: `stream.count()` zählt alle Elemente im Stream.
13. **anyMatch(Predicate)**:
    - Gibt `true` zurück, wenn mindestens ein Element im Stream das Prädikat erfüllt.
    - Beispiel: `stream.anyMatch(x -> x > 10)` überprüft, ob mindestens ein Element größer als 10 ist.
14. **allMatch(Predicate)**:
    - Gibt `true` zurück, wenn alle Elemente das Prädikat erfüllen.
    - Beispiel: `stream.allMatch(x -> x > 10)` prüft, ob alle Elemente größer als 10 sind.
15. **noneMatch(Predicate)**:
    - Gibt `true` zurück, wenn kein Element das Prädikat erfüllt.
    - Beispiel: `stream.noneMatch(x -> x < 0)` überprüft, ob kein Element kleiner als 0 ist.
16. **findFirst()**:
    - Gibt das erste Element im Stream als `Optional` zurück, falls vorhanden.
    - Beispiel: `stream.findFirst()` gibt das erste Element zurück.
17. **findAny()**:
    - Gibt ein beliebiges Element im Stream als `Optional` zurück, falls vorhanden. Kann bei parallelen Streams nützlich sein, um die erste gefundene Übereinstimmung zu erhalten.
    - Beispiel: `stream.findAny()` gibt ein zufälliges Element zurück (bei parallelen Streams nicht deterministisch).