Contents

[Java Stream API 1](#_Toc193797278)

[1. Convert an Array to Stream 1](#_Toc193797279)

[2. Filter Elements 1](#_Toc193797280)

[3. Map and Transform Elements 1](#_Toc193797281)

[4. Reduce to a Single Value 1](#_Toc193797282)

[5. Collect Results 2](#_Toc193797283)

[6. Sorting and Distinct 2](#_Toc193797284)

[7. Find and Match Operations 2](#_Toc193797285)

[8. Convert and Perform Operations on Different Data Types 2](#_Toc193797286)

[9. Find Maximum and Minimum 2](#_Toc193797287)

[10. Convert Array to List Using Stream 3](#_Toc193797288)

[11. Counting Elements with a Condition 3](#_Toc193797289)

[12. Grouping and Partitioning 3](#_Toc193797290)

[13. Skipping and Limiting 3](#_Toc193797291)

[14. Creating a Stream of Random Numbers 4](#_Toc193797292)

[15. Parallel Streams for Faster Processing 4](#_Toc193797293)

Java Stream API

Java Stream API provides a powerful way to process collections and arrays using functional programming concepts.

Here's how you can use Stream API to process arrays efficiently.

## 1. Convert an Array to Stream

You can use Arrays.stream() or Stream.of() to convert an array into a stream.

int[] arr = {1, 2, 3, 4, 5};

Arrays.stream(arr).forEach(System.out::println);

## 2. Filter Elements

Filter elements based on a condition using filter().

int[] arr = {10, 25, 30, 45, 50};

Arrays.stream(arr)

.filter(x -> x > 30)

.forEach(System.out::println); // Output: 45, 50

## 3. Map and Transform Elements

Use map() to transform each element.

int[] arr = {1, 2, 3, 4};

int[] squared = Arrays.stream(arr)

.map(x -> x \* x)

.toArray();

System.out.println(Arrays.toString(squared)); // Output: [1, 4, 9, 16]

## 4. Reduce to a Single Value

Use reduce() to perform operations like sum, product, or finding the maximum.

int[] arr = {1, 2, 3, 4};

int sum = Arrays.stream(arr)

.reduce(0, (a, b) -> a + b);

System.out.println(sum); // Output: 10

## 5. Collect Results

If working with objects or complex data, you can collect results using Collectors.

String[] names = {"John", "Jane", "Jack"};

String result = Arrays.stream(names)

.filter(name -> name.startsWith("J"))

.collect(Collectors.joining(", "));

System.out.println(result); // Output: John, Jane, Jack

## 6. Sorting and Distinct

You can sort or remove duplicates using sorted() and distinct().

int[] arr = {4, 2, 7, 2, 4, 8};

Arrays.stream(arr)

.distinct()

.sorted()

.forEach(System.out::println); // Output: 2, 4, 7, 8

## 7. Find and Match Operations

* findFirst() to get the first element
* anyMatch(), allMatch(), or noneMatch() for conditions

int[] arr = {3, 7, 10, 12};

boolean result = Arrays.stream(arr)

.anyMatch(x -> x > 8);

System.out.println(result); // Output: true

## 8. Convert and Perform Operations on Different Data Types

You can convert between different types using mapToInt(), mapToDouble(), or mapToObj().

String[] strArr = {"1", "2", "3", "4", "5"};

int sum = Arrays.stream(strArr)

.mapToInt(Integer::parseInt)

.sum();

System.out.println(sum); // Output: 15

## 9. Find Maximum and Minimum

You can find the maximum and minimum values using max() and min().

int[] arr = {10, 20, 30, 40, 50};

int max = Arrays.stream(arr).max().orElseThrow();

int min = Arrays.stream(arr).min().orElseThrow();

System.out.println("Max: " + max); // Output: 50

System.out.println("Min: " + min); // Output: 10

## 10. Convert Array to List Using Stream

If you need to convert an array to a list, collect(Collectors.toList()) can be used.

String[] strArr = {"apple", "banana", "cherry"};

List<String> fruitList = Arrays.stream(strArr)

.collect(Collectors.toList());

System.out.println(fruitList); // Output: [apple, banana, cherry]

## 11. Counting Elements with a Condition

You can count elements matching a specific condition using filter() and count().

int[] arr = {1, 3, 5, 7, 8, 10};

long count = Arrays.stream(arr)

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

.count();

System.out.println("Even numbers count: " + count); // Output: 3

## 12. Grouping and Partitioning

You can use Collectors.groupingBy() to group elements.

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

Map<Character, List<String>> grouped = names.stream()

.collect(Collectors.groupingBy(name -> name.charAt(0)));

System.out.println(grouped);

**Output:**

{J=[John, Jane, Jack], A=[Anna, Alice]}

## 13. Skipping and Limiting

You can skip elements using skip() and limit the output using limit().

int[] arr = {10, 20, 30, 40, 50, 60};

Arrays.stream(arr)

.skip(2)

.limit(3)

.forEach(System.out::println); // Output: 30, 40, 50

## 14. Creating a Stream of Random Numbers

You can generate infinite streams using Stream.generate() or IntStream.generate().

Random random = new Random();

IntStream.generate(() -> random.nextInt(100))

.limit(5)

.forEach(System.out::println); // Output: 5 random numbers between 0-99

## 15. Parallel Streams for Faster Processing

For large datasets, you can use parallelStream() to perform operations in parallel.

int[] arr = IntStream.rangeClosed(1, 1000000).toArray();

long sum = Arrays.stream(arr)

.parallel()

.sum();

System.out.println("Sum: " + sum); // Output: 500000500000