Java 8 Streams API – Deep Dive Interview Guide

This document is a comprehensive, interview-focused deep dive into Java 8 Streams API. It is designed for mid-to-senior Java developers and covers internals, core operations, advanced usage, performance considerations, and interview problems.

1. What is a Stream?

A Stream represents a sequence of elements supporting aggregate operations. Streams do not store data and do not modify the underlying data source. They process data in a declarative and functional style.

2. Stream Pipeline

A stream pipeline consists of a source, zero or more intermediate operations, and a terminal operation. Streams are lazy and execute only when a terminal operation is invoked.

3. Stream Creation

Streams can be created from collections, arrays, files, or static factory methods.

Examples: collection.stream(), Arrays.stream(array), Stream.of(values)

4. Intermediate Operations

map(): Transforms each element.

flatMap(): Flattens nested structures.

filter(): Filters elements using Predicate.

distinct(): Removes duplicates using equals().

sorted(): Sorts elements.

limit() and skip(): Control stream size.

5. Terminal Operations

forEach(): Performs action on each element.

collect(): Converts stream to collection or map.

reduce(): Combines elements into single result.

count(), min(), max(), findFirst(), findAny()

6. Collectors Deep Dive

Collectors.toList(), toSet(), toMap()

groupingBy(): Groups elements by key.

partitioningBy(): Splits elements into two groups.

mapping(), averagingInt(), counting()

7. flatMap() in Detail

flatMap() is used when each element produces another stream. It flattens multiple streams into a single stream.

Common use cases: List of lists, Optional, splitting strings, entity relationships.

8. Short-Circuit Operations

anyMatch(), allMatch(), noneMatch(), limit(), findFirst() allow early termination of stream processing.

9. Parallel Streams

Parallel streams split work across multiple threads using ForkJoinPool. They should be used only for CPU-intensive, stateless, and independent operations.

10. Performance & Best Practices

Avoid stateful lambdas.

Prefer method references for readability.

Do not modify shared mutable state.

Avoid parallel streams for I/O or small data sets.

11. Common Stream Interview Problems

• Second highest number

• First non-repeating character

• Group by and aggregation

• Flatten nested collections

• Check conditions using allMatch/anyMatch

12. Common Interview Questions

Difference between Stream and Collection

Why streams are lazy?

Can a stream be reused?

map vs flatMap

reduce vs collect

When to use parallel streams

13. How to Explain Streams in Interview

Streams provide a declarative way to process data using a pipeline of transformations and terminal operations with lazy evaluation, improving readability and maintainability.