I

Java Streams

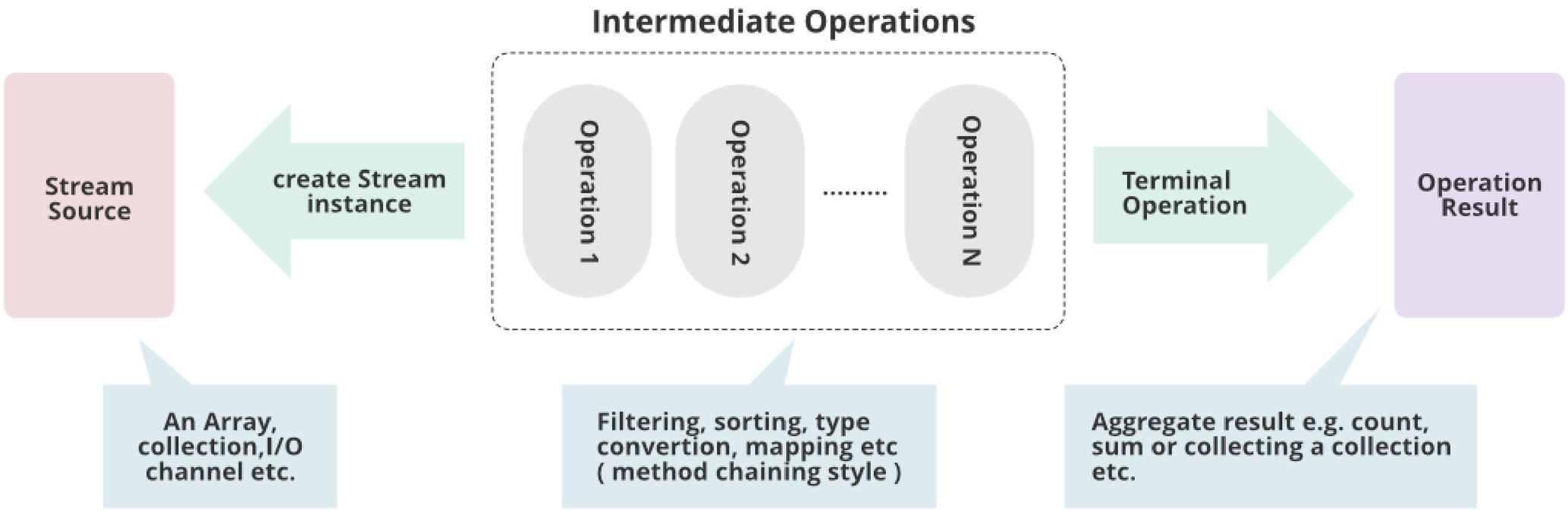
09/09/2024

BAAZIZ WISSAL

Introduction to Java Streams

- Java Streams were introduced in Java 8 as part of the Java Collections Framework to enable functional-style operations on sequences of elements. Streams allow you to express complex data processing queries in a concise and readable manner.

Java Streams



Key Concepts

IV

Stream

A sequence of elements from a source that supports aggregate operations.

Functional Programming

Using functions to perform operations like filtering, mapping, and reducing data.

Lazy Evaluation

Operations on a stream are not executed until a terminal operation is invoked.

VI

Why Use Streams?

Concise Code

Streams reduce boilerplate code.

Parallel Execution

Streams can be processed in parallel, improving performance on large data sets.

Functional Operations

Allows for chaining operations like map, filter, reduce, etc.

VIII

Stream Operations

V

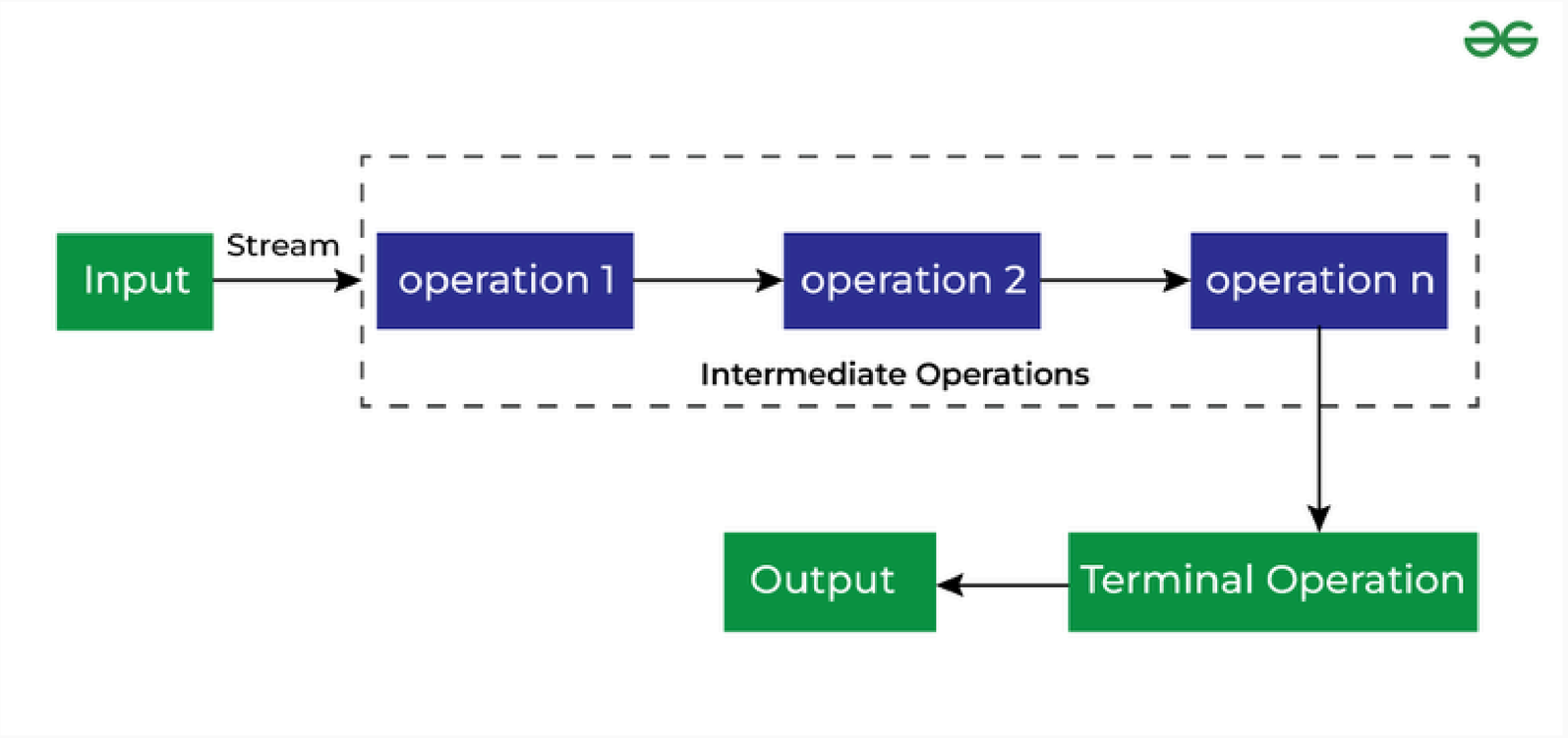
Streams support two types of operations:

- Intermediate Operations

:These return another stream, allowing method chaining. They include operations like map(), filter(), and sorted().

- Terminal Operations:

These produce a result or a side effect. Common terminal operations include forEach(), collect(), and reduce().


Input
Stream
operation 1
operation 2
operation n
Intermediate Operations
Output
Terminal Operation

V

Common Intermediate Operations

- filter(Predicate): Filters elements based on a condition.
- map(Function): Transforms elements.
- sorted(): Sorts elements.

Common Terminal Operations

- collect(Collector): Converts a stream into a collection or another data type.
- forEach(Consumer): Performs an action for each element.
- reduce(BinaryOperator): Reduces the stream to a single value.

Example of Stream Operations

```java
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class StreamExample {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

        // Filter even numbers, square them, and collect to a list
        List<Integer> evenSquares = numbers.stream()
                                           .filter(n -> n % 2 == 0) // Filter even numbers
                                           .map(n -> n * n)         // Square each number
                                           .collect(Collectors.toList()); // Collect to lis

        System.out.println(evenSquares); // Output: [4, 16, 36, 64, 100]
    }
}
```

XIII

Thank you!