



Vikash Singh
@full_stack_geek

Java Stream API



Vikash Singh
@full_stack_geek

Stream API is used to process collections of objects. A stream in Java is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

The uses of Stream in Java are mentioned below:

- 1) Stream API is a way to express and process collections of objects.
- 2) Enable us to perform operations like filtering, mapping, reducing and sorting.



Vikash Singh
@full_stack_geek

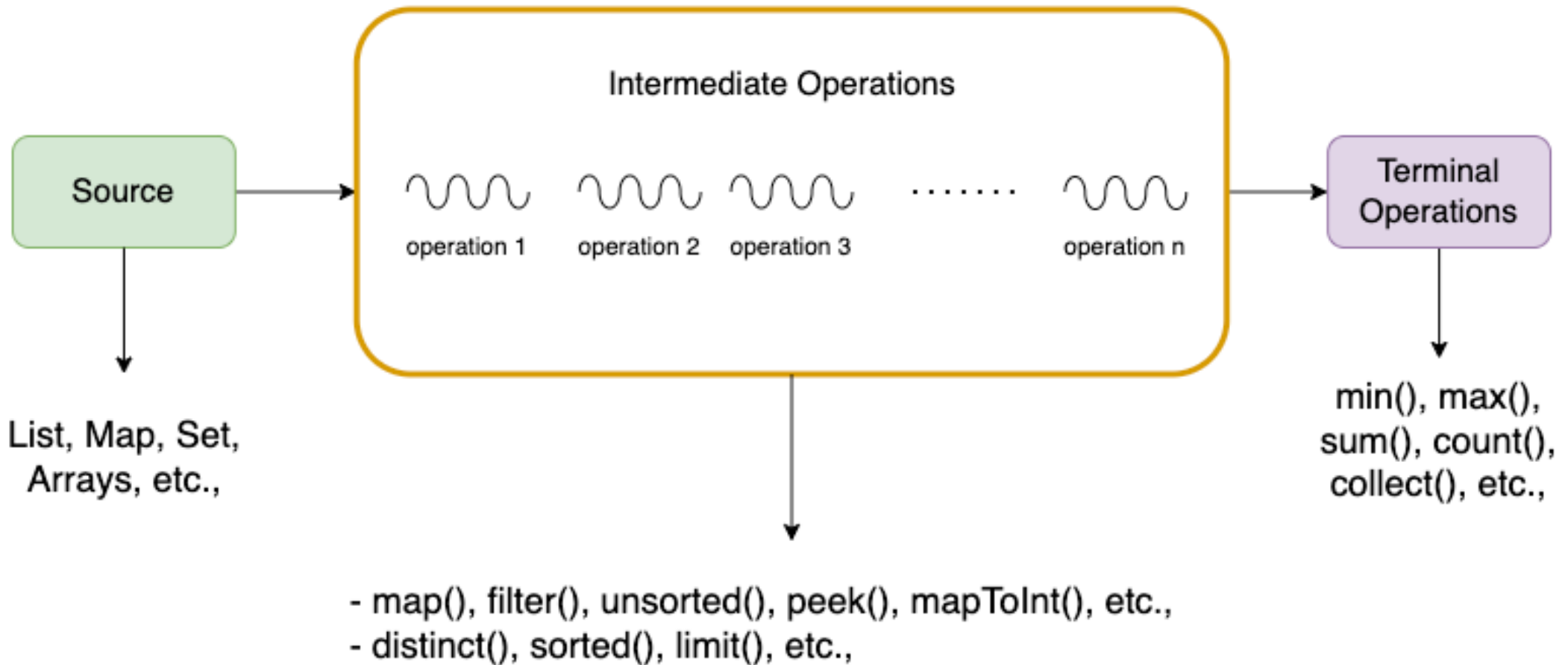
Java Stream Features

The features of Java stream are mentioned below:

- 1) A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.
 - 2) Streams don't change the original data structure, they only provide the result as per the pipelined methods.
 - 3) Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined.
- Terminal operations mark the end of the stream and return the result.



Vikash Singh
@full_stack_geek



Different Operations On Streams

There are two types of Operations in Streams:

- 1) Intermediate Operations
- 2) Terminate Operations



Vikash Singh

@full_stack_geek

intermediate Operations are the types of operations in which multiple methods are chained in a row.

There are a few Intermediate Operations mentioned below:

1) Map

```
List number = Arrays.asList(2,3,4,5);
```

```
List square = number.stream().map(x->x*x).collect(Collectors.toList());
```

2) filter

```
List names = Arrays.asList("Reflection","Collection","Stream");
```

```
List result = names.stream().filter(s-
```

```
>s.startsWith("S")).collect(Collectors.toList());
```

Terminal Operations

1) collect

```
List number = Arrays.asList(2,3,4,5,3);
```

```
Set square = number.stream().map(x->x*x).collect(Collectors.toSet());
```

2) forEach()

```
List number = Arrays.asList(2,3,4,5);
```

```
number.stream().map(x->x*x).forEach(y->System.out.println(y));
```



Vikash Singh
@full_stack_geek

Examples

```
// create a list of integers
List<Integer> number = Arrays.asList(2, 3, 4, 5);

// demonstration of map method
List<Integer> square = number.stream().map(x -> x * x) .collect(Collectors.toList());

// create a list of String
List<String> names = Arrays.asList("Reflection", "Collection", "Stream");

// demonstration of filter method
List<String> result= names.stream() .filter(s -> s.startsWith("S")).collect(Collectors.toList())

// demonstration of sorted method
List<String> sho = names.stream() .sorted().collect(Collectors.toList());

// collect method returns a set
Set<Integer> squareSet = number .stream().map(x -> x * x).collect(Collectors.toSet());

// demonstration of forEach method
number.stream().map(x -> x * x).forEach(y -> System.out.println(y));

// demonstration of reduce method
int even = number.stream() .filter(x -> x % 2 == 0).reduce(0, (ans, i) -> ans + i);
```




Vikash Singh

@full_stack_geek

Modifier and Type	Method and Description
boolean	allMatch (Predicate<? super T> predicate) Returns whether all elements of this stream match the provided predicate.
boolean	anyMatch (Predicate<? super T> predicate) Returns whether any elements of this stream match the provided predicate.
static <T> Stream.Builder<T>	builder () Returns a builder for a Stream.
<R,A> R	collect (Collector<? super T,A,R> collector) Performs a mutable reduction operation on the elements of this stream using a Collector.
<R> R	collect (Supplier<R> supplier, BiConsumer<R,? super T> accumulator, BiConsumer<R,R> combiner) Performs a mutable reduction operation on the elements of this stream.
static <T> Stream<T>	concat (Stream<? extends T> a, Stream<? extends T> b) Creates a lazily concatenated stream whose elements are all the elements of the first stream followed by all the elements of the second stream.
long	count () Returns the count of elements in this stream.
Stream<T>	distinct () Returns a stream consisting of the distinct elements (according to Object.equals(Object)) of this stream.
static <T> Stream<T>	empty () Returns an empty sequential Stream.



Vikash Singh

@full_stack_geek

<code>Stream<T></code>	<code>filter(Predicate<? super T> predicate)</code> Returns a stream consisting of the elements of this stream that match the given predicate.
<code>Optional<T></code>	<code>findAny()</code> Returns an Optional describing some element of the stream, or an empty Optional if the stream is empty.
<code>Optional<T></code>	<code>findFirst()</code> Returns an Optional describing the first element of this stream, or an empty Optional if the stream is empty.
<code><R> Stream<R></code>	<code>flatMap(Function<? super T,? extends Stream<? extends R>> mapper)</code> Returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element.
<code>DoubleStream</code>	<code>flatMapToDouble(Function<? super T,? extends DoubleStream> mapper)</code> Returns an DoubleStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element.
<code>IntStream</code>	<code>flatMapToInt(Function<? super T,? extends IntStream> mapper)</code> Returns an IntStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element.
<code>LongStream</code>	<code>flatMapToLong(Function<? super T,? extends LongStream> mapper)</code> Returns an LongStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element.
<code>void</code>	<code>forEach(Consumer<? super T> action)</code> Performs an action for each element of this stream.
<code>void</code>	<code>forEachOrdered(Consumer<? super T> action)</code> Performs an action for each element of this stream, in the encounter order



Vikash Singh

@full_stack_geek

Returning an element for each element of this stream, in the encounter order of the stream if the stream has a defined encounter order.

static <T> Stream<T>

generate(Supplier<T> s)

Returns an infinite sequential unordered stream where each element is generated by the provided Supplier.

static <T> Stream<T>

iterate(T seed, UnaryOperator<T> f)

Returns an infinite sequential ordered Stream produced by iterative application of a function f to an initial element seed, producing a Stream consisting of seed, f(seed), f(f(seed)), etc.

Stream<T>

limit(long maxSize)

Returns a stream consisting of the elements of this stream, truncated to be no longer than maxSize in length.

<R> Stream<R>

map(Function<? super T,? extends R> mapper)

Returns a stream consisting of the results of applying the given function to the elements of this stream.

DoubleStream

mapToDouble(ToDoubleFunction<? super T> mapper)

Returns a DoubleStream consisting of the results of applying the given function to the elements of this stream.

IntStream

mapToInt(ToIntFunction<? super T> mapper)

Returns an IntStream consisting of the results of applying the given function to the elements of this stream.

LongStream

mapToLong(ToLongFunction<? super T> mapper)

Returns a LongStream consisting of the results of applying the given function to the elements of this stream.

Optional<T>

max(Comparator<? super T> comparator)

Returns the maximum element of this stream according to the provided Comparator.

Optional<T>

min(Comparator<? super T> comparator)

Returns the minimum element of this stream according to the provided Comparator.



Vikash Singh

@full_stack_geek

<code>boolean</code>	<code>noneMatch(Predicate<? super T> predicate)</code> Returns whether no elements of this stream match the provided predicate.
<code>static <T> Stream<T></code>	<code>of(T... values)</code> Returns a sequential ordered stream whose elements are the specified values.
<code>static <T> Stream<T></code>	<code>of(T t)</code> Returns a sequential Stream containing a single element.
<code>Stream<T></code>	<code>peek(Consumer<? super T> action)</code> Returns a stream consisting of the elements of this stream, additionally performing the provided action on each element as elements are consumed from the resulting stream.
<code>Optional<T></code>	<code>reduce(BinaryOperator<T> accumulator)</code> Performs a reduction on the elements of this stream, using an associative accumulation function, and returns an Optional describing the reduced value, if any.
<code>T</code>	<code>reduce(T identity, BinaryOperator<T> accumulator)</code> Performs a reduction on the elements of this stream, using the provided identity value and an associative accumulation function, and returns the reduced value.
<code><U> U</code>	<code>reduce(U identity, BiFunction<U,? super T,U> accumulator, BinaryOperator<U> combiner)</code> Performs a reduction on the elements of this stream, using the provided identity, accumulation and combining functions.
<code>Stream<T></code>	<code>skip(long n)</code> Returns a stream consisting of the remaining elements of this stream after discarding the first n elements of the stream.
<code>Stream<T></code>	<code>sorted()</code> Returns a stream consisting of the elements of this stream, sorted according to their natural ordering.