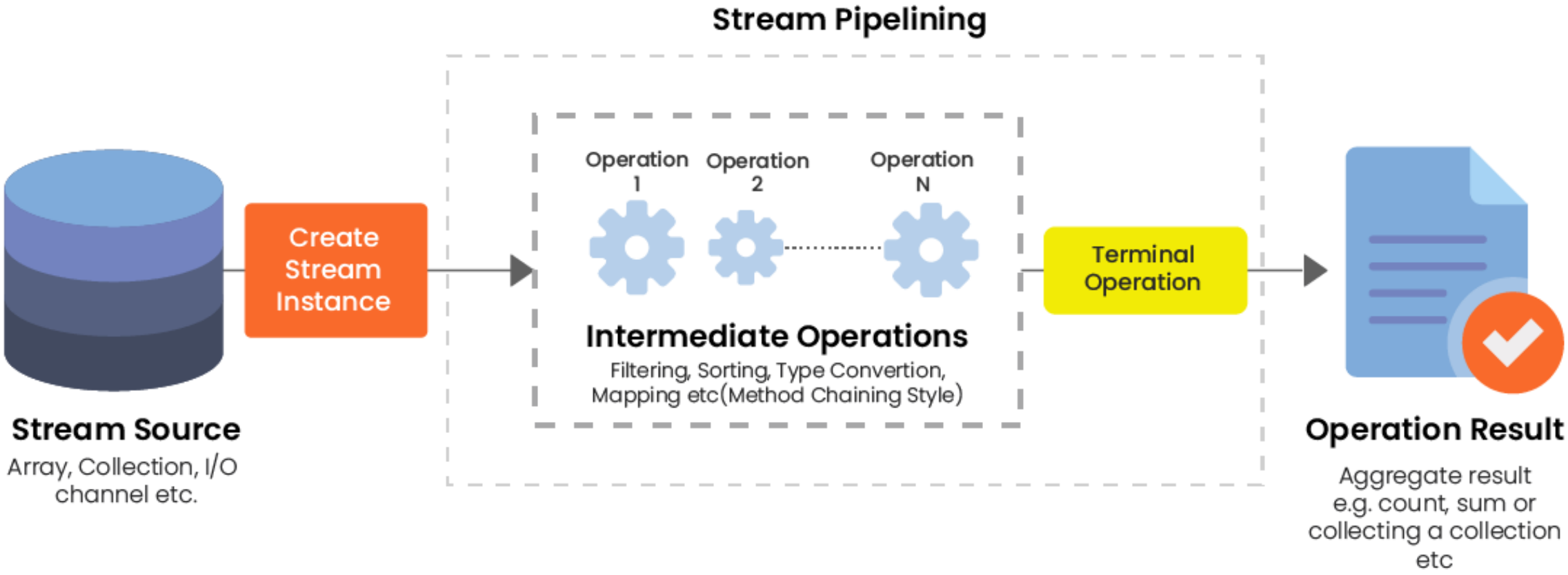
Java Streams



Streams

- Stream represents a sequence of objects from a source, which supports aggregate operations.

- Streams are one of the most important additions on JDK, it allows you to leverage other changes like lambda expression, method reference, functional interface and internal iteration introduced via the forEach() method.

- By the way, this stream is a bit different than your Java IO streams, e.g. InputStream and OutputStream.

Start Stream

⇒ Stream.of(a, b,c)

or

⇒ source.stream()

⇒ a b c

Source – Stream takes Collections, Arrays, or I/O resources as input source..

Intermediate operations are functions that produce another stream from the existing stream like filter, map, sorted, etc.

- Here is a list of commonly used Stream operations.
  - distinct – Intermediate
  - filter – Intermediate
  - limit – Intermediate
  - map – Intermediate
  - peek – Intermediate
  - skip – Intermediate
  - sorted – Intermediate

Map

⇒ source.stream()

⇒ map(data -> function)

⇒ new stream after function operation

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

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

Filter

⇒ source.stream()

⇒ map(data -> function)

⇒ filter(data -> comparison with data)

⇒ new stream if data passes comparison

Returns a stream consisting of the elements of this stream that match the given predicate.

Stream<T> filter(Predicate<? super T> predicate);

Sort

⇒ source.stream()

⇒ map(data -> function)

⇒ filter(data -> comparison with data)

⇒ sorted()

⇒ new sorted stream

Returns a stream consisting of the elements of this stream, sorted according to natural order.

Stream<T> sorted();

Terminal operations are functions that produce a non-stream result from the Stream like collect(toList()) , forEach, count etc.

- List is continued here…
  - allMatch – Terminal
  - anyMatch – Terminal
  - findAny – Terminal
  - findFirst – Terminal
  - noneMatch – Terminal
  - forEach – Terminal
  - reduce – Terminal

Terminate

⇒ source

⇒ map(data -> function)

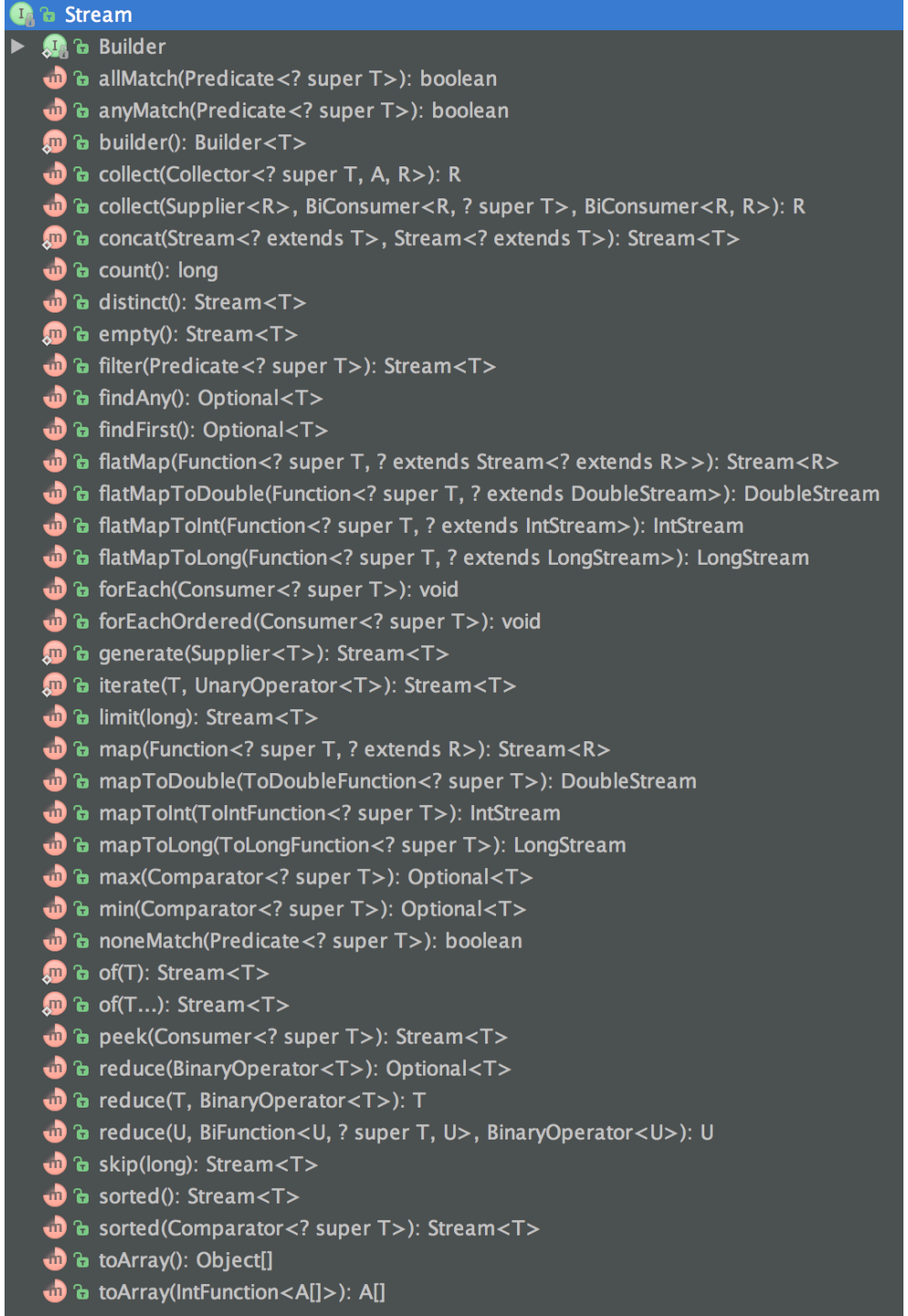
⇒ filter(data -> comparison with data)

⇒ sorted()

⇒ collect(Collectors.toList())

Streams are not evaluated until a terminal operation is called on them.

Terminal operations are used to produce a result, and after that, you cannot reuse them.


Stream
Builder
allMatch(Predicate<? super T>): boolean
anyMatch(Predicate<? super T>): boolean
builder(): Builder<T>
collect(Collector<? super T, A, R>): R
collect(Supplier<R>, BiConsumer<R, ? super T>, BiConsumer<R, R>): R
concat(Stream<? extends T>, Stream<? extends T>): Stream<T>
count(): long
distinct(): Stream<T>
empty(): Stream<T>
filter(Predicate<? super T>): Stream<T>
findAny(): Optional<T>
findFirst(): Optional<T>
flatMap(Function<? super T, ? extends Stream<? extends R>>): Stream<R>
flatMapToDouble(Function<? super T, ? extends DoubleStream>): DoubleStream
flatMapToInt(Function<? super T, ? extends IntStream>): IntStream
flatMapToLong(Function<? super T, ? extends LongStream>): LongStream
forEach(Consumer<? super T>): void
forEachOrdered(Consumer<? super T>): void
generate(Supplier<T>): Stream<T>
iterate(T, UnaryOperator<T>): Stream<T>
limit(long): Stream<T>
map(Function<? super T, ? extends R>): Stream<R>
mapToDouble(ToDoubleFunction<? super T>): DoubleStream
mapToInt(ToIntFunction<? super T>): IntStream
mapToLong(ToLongFunction<? super T>): LongStream
max(Comparator<? super T>): Optional<T>
min(Comparator<? super T>): Optional<T>
noneMatch(Predicate<? super T>): boolean
of(T): Stream<T>
of(T...): Stream<T>
peek(Consumer<? super T>): Stream<T>
reduce(BinaryOperator<T>): Optional<T>
reduce(T, BinaryOperator<T>): T
reduce(U, BiFunction<U, ? super T, U>, BinaryOperator<U>): U
skip(long): Stream<T>
sorted(): Stream<T>
sorted(Comparator<? super T>): Stream<T>
toArray(): Object[]
toArray(IntFunction<A[]>): A[]