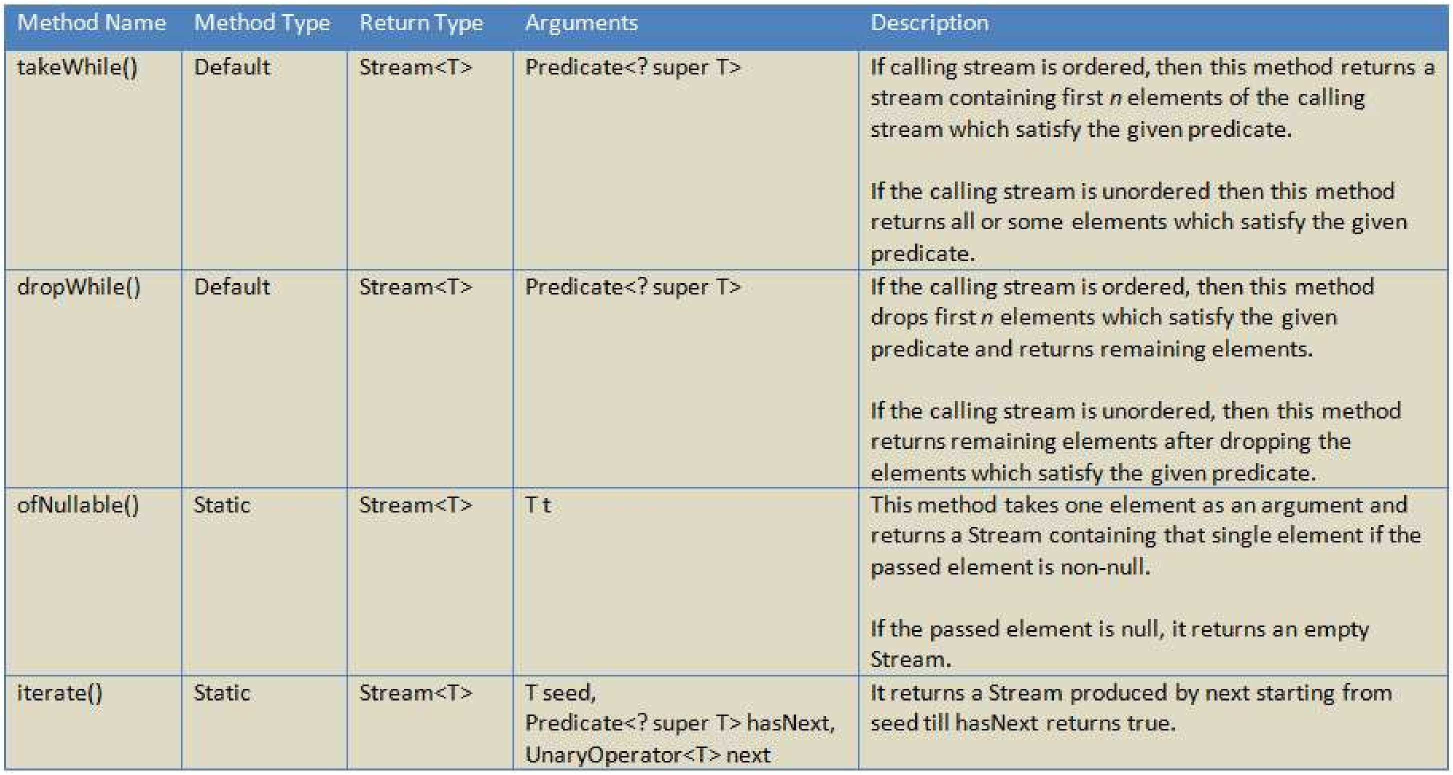
Java 9 Stream API Improvements : takeWhile(), dropWhile(), ofNullable() And iterate()

pramodbablad September 14, 2021 0

Streams in Java are introduced from Java 8. The operations which operate on streams are kept in java.util.stream.Stream interface. From Java 9, four new operations are added to this interface. They are – takeWhile(), dropWhile(), ofNullable() and iterate() methods in which takeWhile() and dropWhile() methods are default methods and ofNullable() and iterate() methods are static methods. Let's take a look at Java 9 Stream API improvements.

Java 9 Stream API Improvements :

Four new methods are introduced in `java.util.stream.Stream` interface from Java 9 to improve the working with streams. Below table shows all four methods with their description.



Method Name	Method Type	Return Type	Arguments	Description
takeWhile()	Default	Stream<T>	Predicate<? super T>	If calling stream is ordered, then this method returns a stream containing first *n* elements of the calling stream which satisfy the given predicate. If the calling stream is unordered then this method returns all or some elements which satisfy the given predicate.
dropWhile()	Default	Stream<T>	Predicate<? super T>	If the calling stream is ordered, then this method drops first *n* elements which satisfy the given predicate and returns remaining elements. If the calling stream is unordered, then this method returns remaining elements after dropping the elements which satisfy the given predicate.
ofNullable()	Static	Stream<T>	T t	This method takes one element as an argument and returns a Stream containing that single element if the passed element is non-null. If the passed element is null, it returns an empty Stream.
iterate()	Static	Stream<T>	T seed, Predicate<? super T> hasNext, UnaryOperator<T> next	It returns a Stream produced by next starting from seed till hasNext returns true.

Let's see these methods with some simple examples one by one.

1) takeWhile() :

syntax :

default Stream<T> takeWhile(Predicate<? super T> predicate)

`takeWhile()` is a default method, takes one `Predicate` as an argument and returns a `Stream`. This method is a short-circuiting intermediate operation.

If calling stream is ordered, then this method returns a stream containing first `n` elements of the calling stream which satisfy the given predicate. It immediately terminates the operation as soon as it sees an element which doesn't satisfy the predicate and it doesn't evaluate remaining elements even though there may be elements which satisfy the given predicate.

If the calling stream is unordered then this method returns all or some elements which satisfy the given predicate. In such conditions, behavior of this method becomes non-deterministic.

For example, in the below code snippet, `[1, 10, 100, 1000, 10000, 1000, 100, 10, 1, 0, 10000]` is the calling stream and `i<5000` is the predicate then `takeWhile()` returns first 4 elements `[1, 10, 100, 1000]` which satisfy the given predicate. When it sees `10000` which doesn't satisfy `i<5000`, it breaks the operation and doesn't evaluate remaining elements even though there are elements which satisfy `i<5000`.

```
IntStream.of(1, 10, 100,
                .takeWhi
                .forEach
```

Output :

```
1
10
100
1000
```

2) dropWhile()

Syntax :

default Stream<T> dropWhile(Predicate<? super T> predicate)

`dropWhile()` is also a default method, takes one `Predicate` as an argument and returns a `Stream`. It is also a short-circuiting intermediate operation.

This method is total opposite of `takeWhile()`. This method drops first `n` elements which satisfy the given predicate and returns remaining elements if the calling stream is ordered.

For example, if we apply `dropWhile()` in the above example, we get the output as follows.

```
IntStream.of(1, 10, 100,
                .dropWhi
                .forEach
```

Output :

```
10000
1000
100
10
1
0
10000
```

If the calling stream is unordered, then this method returns remaining elements after dropping the elements which satisfy the given predicate. In such cases, the behavior of this method becomes unpredictable.

3) ofNullable()

Syntax :

static Stream<T> ofNullable(T t)

`ofNullable()` is a static method which takes one element as an argument and returns a `Stream` containing that single element if the passed element is non-null. If the passed element is null, it returns an empty `Stream`.

```
long count = Stream.ofNu

System.out.println(count

count = Stream.ofNullabl

System.out.println(count
```

4) iterate()

Syntax :

static Stream<T> iterate(T seed, Predicate<? super T> hasNext, UnaryOperator<T> next)

`iterate()` method is already there in `Stream` interface from Java 8. But, Java 9 provides another version of `iterate()` method which takes an extra argument `hasNext` of type `Predicate` which decides when to terminate the operation.

`iterate()` method is also a static method.

```
Stream.iterate(1, i -> i
```

Output :

```
1
10
100
1000
10000
100000
```

Also Rea