HowToDoInJava

Primitive Type Streams in Java

By: Lokesh Gupta Java Streams Java Stream Basics

Learn to create and operate on the streams of primitive types in Java with examples.

Table Of Contents

1. Primitives and Wrapper Classes

Java is not a true object-oriented programming language and supports primitive types that are not objects. We have **7 primitives** in Java that are `byte`, `short`, `int`, `long`, `double`, `float`, `char`.

Java allows to wrap them in objects (wrapper classes) so these types can be represented as objects when required. The corresponding wrapper classes are *Byte*, *Short*, *Integer*, *Long*, *Double*, *Float* and *Char*.

The process of converting a primitive to an object is called **auto-boxing** and converting an object to a primitive is called **unboxing**.

2. Support for Primitive Streams

Java Stream API, similar to *Collections API*, has been designed to work upon objects and not primitive types.

The stream API has inbuilt support for representing primitive streams using the following specialized classes. All these classes support the sequential and parallel aggregate operations on stream items.

- **IntStream** : represents sequence of primitive int-valued elements.
- **LongStream** : represents sequence of primitive long-valued elements.
- **DoubleStream** : represents sequence of primitive double-valued elements.

These classes help in avoiding a lot of unnecessary object creation, auto-boxing and unboxing operations if we decide to do these operations on our own.

For other primitive types, Java does not provide similar stream support classes as it was not found useful to have so many classes. The `int`, `long` and `double` are very highly used types so support was added for them.

3. Creating Streams of Primitives

3.1. Creating Stream of Specified Values

If we have a few specified values of *int*, *long* or *double* then we can create the stream using the **of()** factory method.

```
IntStream stream = IntStream.of(1, 2, 3, 4, 5);
LongStream stream = LongStream.of(1, 2, 3, 4, 5);
DoubleStream stream = DoubleStream.of(1.0, 2.0, 3.0, 4.0, 5.0);
```

3.2. Stream.range() Fatory Method

The range() method returns a sequential ordered `IntStream` or `LongStream` from *startInclusive (inclusive)* to *endExclusive (exclusive)* by an incremental step of 1.

```
IntStream stream = IntStream.range(1, 10);  //1,2,3,4,5,6,7,8,9
LongStream stream = LongStream.range(10, 100);
```

A similar method `rangeClosed()` also returns a sequential ordered stream but the **end item is inclusive** in the stream.

```
IntStream stream = IntStream.rangeClosed(1, 10);  //1,2,3,4,5,6,7,8,9
```

3.3. Arrays.stream()

We can directly call the `stream()` method on an *array* that will return an instance of *Stream* class corresponding to the type of array.

For example, if we call `array.stream()` on an `int[]` then it will return an instance of `IntStream`.

```
// int[] -> Stream
int[] array = new int[]{1, 2, 3, 4, 5};
IntStream stream = Arrays.stream(array);

// long[] -> Stream
long[] array = new long[]{1, 2, 3, 4, 5};
```

```
LongStream stream = Arrays.stream(array);

// double[] -> Stream
double[] array = new double[]{1.0, 2.0, 3.0, 4.0, 5.0};
DoubleStream stream = Arrays.stream(array);
```

3.4. Stream mapToInt(), mapToLong() and mapToDouble()

Another technique to get the primitive stream is using the **mapTo()** function for the corresponding type.

For example, if we have a stream of `Integer` or any other type of object with a field of `Integer` type such as person's age) then we can get the stream of all such values as a stream of `int` values.

```
List<Integer> integerList = List.of(1, 2, 3, 4, 5);
IntStream stream = integerList.stream().mapToInt(i -> i);

Stream<Employee> streamOfEmployees = getEmployeeStream();
DoubleStream stream = streamOfEmployees.mapToDouble(e -> e.getSalary(
```

4. Finding Sum, Average, Max and Min

4.1. Built-in Methods

All three classes, *IntStream*, *LongStream* and *DoubleStream*, consist of numerical values and it makes sense to provide built-in support for common aggregate operations on items of the stream.

These classes provide the following methods for these operations. The return types are corresponding to the type of the stream. The following methods are from `IntStream` class:

- *sum()* – returns the sum of items in the stream.
- *average()* – returns an `OptionalDouble` describing the arithmetic mean of items of the stream.
- *max()* – returns an `OptionalInt` describing the maximum item of the stream.
- *min()* – returns an `OptionalInt` describing the mimimum item of the stream.
- *count()* – returns the count of items in the stream.

Let's see a few examples of how to use these methods.

```
int max = IntStream.of(10, 18, 12, 70, 5)
  .max()
  .getAsInt();

double avg = IntStream.of(1, 2, 3, 4, 5)
  .average()
  .getAsDouble();

int sum = IntStream.range(1, 10)
  .sum();
```

4.2. Summary Statistics

Another way to find the above statistical data is by using the `summaryStatistics()` method that returns one of the following classes:

- `IntSummaryStatistics`
- `LongSummaryStatistics`
- `DoubleSummaryStatistics`

Now we can use its methods to get the required value.

- *getAverage()*
- *getCount()*

- *getMax()*
- *getMin()*
- *getSum()*

```
IntSummaryStatistics summary = IntStream.of(10, 18, 12, 70, 5)
    .summaryStatistics();

int max = summary.getMax();
```

5. Primitive Stream to Object Stream

Using the `boxed()` method, we can convert a primitive stream to an object stream of the corresponding type.

For example, to **get Stream<Long> from a `LongStream`**, we can call the `boxed()` method:

```
Stream<Integer> boxedStream1 = IntStream.of(1, 2, 3, 4, 5).boxed();
Stream<Long> boxedStream = LongStream.of(1, 2, 3, 4, 5).boxed();
Stream<Double> boxedStream2 =
    DoubleStream.of(1.0, 2.0, 3.0, 4.0, 5.0).boxed();
```

6. Conclusion

In this tutorial, we understood the support available in Java for a stream of primitives. We learned the different ways to create primitive streams and then we learned to perform some common numerical operations of the stream items.

We also learned to get the boxed streams and summary statistics.

Happy Learning !!

Sourcecode on Github

Was this post helpful?

Let us know if you liked the post. That's the only way we can improve.

Yes

No

Recommended Reading:

1. Python Type Conversion and Type Casting
2. Java Primitive Data Types
3. Java Streams API
4. Creating Streams in Java
5. Boxed Streams in Java
6. Using 'if-else' Conditions with Java Streams
7. Creating Infinite Streams in Java
8. Sorting Streams in Java
9. Applying Multiple Conditions on Java Streams
0. Finding Max and Min from List using Streams

Join 7000+ Awesome Developers

Get the latest updates from industry, awesome resources, blog updates and much more.

Email Address

Subscribe

** We do not spam !!*

Leave a Comment

Name *

Email *

Website

☐ Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search ...

- Guest Posts

Blogs

REST API Tutorial