

## Guide to IntStream in Java

📅 Last Updated: March 4, 2022    👤 By: Lokesh Gupta    📁 Java 8    🔖 Java 8, Java Stream Basics

**Java IntStream** class is a specialization of **Stream** interface for **int** primitive. It represents a **stream of primitive int-valued elements** supporting sequential and parallel aggregate operations.

**IntStream** is part of the `java.util.stream` package and implements **AutoCloseable** and **BaseStream** interfaces.

### Table Of Contents ▼

1. Creating IntStream
  - 1.1. With Specified Values
  - 1.2. Generating ints in Range
  - 1.3. Infinite Streams with Iteration
  - 1.4. Infinite Streams with IntSupplier
2. Iterating Over Values
3. Filtering the Values
4. Converting IntStream to Array
5. Converting IntStream to List

## 1. Creating IntStream

There are several ways of creating an **IntStream**.

### 1.1. With Specified Values

This function returns a sequential ordered stream whose elements are the specified values.

It comes in two versions i.e. single element stream and multiple values stream.

- **IntStream of(int t)** – Returns stream containing a single specified element.
- **IntStream of(int... values)** – Returns stream containing specified all elements.

```
IntStream.of(10);           //10
IntStream.of(1, 2, 3);      //1,2,3
```

## 1.2. Generating ints in Range

The `IntStream` produced by `range()` methods is a sequential ordered stream of int values which is the equivalent sequence of increasing int values in a for-loop and value incremented by 1. This class supports two methods.

- `range(int start, int end)` – Returns a sequential ordered int stream from startInclusive (*inclusive*) to endExclusive (*exclusive*) by an incremental step of 1.
- `rangeClosed(int start, int end)` – Returns a sequential ordered int stream from startInclusive (*inclusive*) to endInclusive (*inclusive*) by an incremental step of 1.

```
IntStream.range(1, 5);          //1,2,3,4
```

```
IntStream.rangeClosed(1, 5);    //1,2,3,4,5
```

## 1.3. Infinite Streams with Iteration

The `iterator()` function is useful for creating [infinite streams](#). Also, we can use this method to produce streams where values are increment by any other value than 1.

Given example produces first 10 even numbers starting from 0.

```
IntStream.iterate(0, i -> i + 2).limit(10);  
  
//0,2,4,6,8,10,12,14,16,18
```

## 1.4. Infinite Streams with IntSupplier

The `generate()` method looks a lot like `iterator()`, but differs by not calculating the int values by incrementing the previous value. Rather an [IntSupplier](#) is provided which is a [functional interface](#) is used to generate an infinite sequential **unordered** stream of int values.

Following example create a stream of 10 random numbers and then print them in the console.

```
IntStream stream = IntStream  
    .generate(() -> { return (int)(Math.random() * 10000); });  
  
stream.limit(10).forEach(System.out::println);
```

## 2. Iterating Over Values

To loop through the elements, stream support the `forEach()` operation. To replace simple [for-loop](#) using `IntStream`, follow the same approach.

```
IntStream.rangeClosed(0, 4)
    .forEach( System.out::println );
```

### 3. Filtering the Values

We can apply filtering on *int* values produced by the stream and use them in another function or collect them for further processing.

For example, we can iterate over int values and filter/collect all prime numbers up to a certain limit.

```
IntStream stream = IntStream.range(1, 100);

List<Integer> primes = stream.filter(ThisClass::isPrime)
    .boxed()
    .collect(Collectors.toList());

public static boolean isPrime(int i)
{
    IntPredicate isDivisible = index -> i % index == 0;
    return i > 1 && IntStream.range(2, i).noneMatch(isDivisible);
}
```

### 4. Converting IntStream to Array

Use **IntStream.toArray()** method to convert from the stream to *int* array.

```
int[] intArray = IntStream.of(1, 2, 3, 4, 5).toArray();
```

### 5. Converting IntStream to List

Collections in Java can not store the primitive values directly. They can store only instances/objects.

Using **boxed()** method of **IntStream**, we can get a stream of wrapper objects which can be collected by **Collectors** methods.

```
List<Integer> list = IntStream.of(1,2,3,4,5)
    .boxed()
    .collect(Collectors.toList());
```

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. [Boxed Streams in Java](#)
2. [Using 'if-else' Conditions with Java Streams](#)
3. [Java Stream sorted\(\)](#)
4. [Java Stream toArray\(\)](#)
5. [Java Stream findFirst\(\)](#)
6. [Java Stream findAny\(\)](#)
7. [Getting the Last Item of a Stream](#)
8. [Java 8 Comparator example with lambda](#)
9. [Stream of Random Numbers in Java](#)
0. [Java Stream map\(\)](#)

## 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

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

## Post Comment

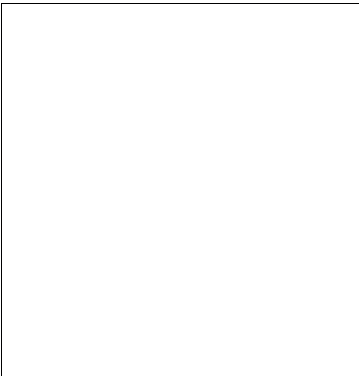


Promoted by **usertesting.com**

Sponsored



A message from our sponsor



A message from our sponsor

## HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

### Meta Links

- › [About Me](#)
- › [Contact Us](#)
- › [Privacy policy](#)
- › [Advertise](#)
- › [Guest Posts](#)

### Blogs

[REST API Tutorial](#)



Copyright © 2022 · Hosted on [Cloudways](#) · [Sitemap](#)