

## Java Stream map()



Last Updated: March 29,  
2022



By: Lokesh  
Gupta



Java  
8



Java Stream Basics, Java Stream  
Methods

Java 8 `Stream.map()` converts `Stream<X>` to `Stream<Y>`. For each object of type `X`, a new object of type `Y` is created and put in the new `Stream`.

### Table Of Contents

#### [1. Stream map\(\) Method](#)

##### [1.1. Method Syntax](#)

##### [1.2. Description](#)

#### [2. Stream map\(\) Example](#)

[Example 1: Converting a Stream of Strings to a Stream of Integers](#)

[Example 2: Finding all distinct salaries among all employees](#)

## 1. Stream map() Method

### 1.1. Method Syntax

The `Stream.map()` method has the following syntax.

#### Method Syntax

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

- **R** represents the element type of the new stream.
- **mapper** is a non-interfering, stateless function to apply to each element which produces a stream of new values.
- The method returns a new stream of objects of type **R**.

**Stream** interface has three more similar methods which produce **IntStream**, **LongStream** and **DoubleStream** respectively after the map operation.

If the streams created after **map()** operations are given return types then consider using these functions directly.

### Similar methods

```
IntStream mapToInt(ToIntFunction<? super T> mapper)
LongStream mapToLong(ToLongFunction<? super T> mapper)
DoubleStream mapToDouble(ToDoubleFunction<? super T> mapper)
```

## 1.2. Description

- The **map()** is an **intermediate operation**. It returns a new **Stream** as return value.
- The **map()** operation takes a **Function**, which is called for each value in the input stream and produces one result value, which is sent to the output stream.
- The mapper function used for transformation is a stateless function (does not store the information of previously processed objects) and returns only a **single value**.
- The **map()** method is used when we want to convert a Stream of **X** to Stream of **Y**.
- The mapped stream is closed after its contents have been placed into the new output stream.
- **map()** operation does not flatten the stream as **flatMap()** operation does.

## 2. Stream map() Example

### Example 1: Converting a Stream of Strings to a Stream of Integers

In this example, we will convert a `Stream<String>` to `Stream<Integer>`. Here the *mapper function* `Integer::valueOf()` takes one string from the Stream at a time, and convert the `String` to an `Integer`.

It then put the `Integer` into another stream which is then collected using `Collectors.toList()`.

```
List<String> listOfStrings = Arrays.asList("1", "2", "3", "4", "5");

List<Integer> listOfIntegers = listOfStrings.stream()
    .map(Integer::valueOf)
    .collect(Collectors.toList());

System.out.println(listOfIntegers);
```

Program output.

```
[1, 2, 3, 4, 5]
```

### Example 2: Finding all distinct salaries among all employees

Java example to find all possible `distinct` salaries for a `List` of employees.

```
List<Employee> employeesList = Arrays.asList(
    new Employee(1, "Alex", 100),
    new Employee(2, "Brian", 100),
    new Employee(3, "Charles", 200),
    new Employee(4, "David", 200),
    new Employee(5, "Edward", 300),
    new Employee(6, "Frank", 300)
```

```
);
```

```
List<Double> distinctSalaries = employeesList.stream()  
    .map( e -> e.getSalary() )  
    .distinct()  
    .collect(Collectors.toList());  
  
System.out.println(distinctSalaries);
```

Program output.

```
[100.0, 200.0, 300.0]
```

Drop me your questions related to **Stream map() method** in [Java Stream API](#).

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. [Jackson – Convert JSON to Map and Map to JSON](#)
2. [Java Stream map\(\) vs flatMap\(\)](#)

3. [Collecting Stream Items into Map in Java](#)
4. [Java Stream reuse – traverse stream multiple times?](#)
5. [Sorting a Map by Keys in Java](#)
6. [Java Sort Map by Values \(ascending and descending orders\)](#)
7. [\[Solved\] org.codehaus.jackson.map.JsonMappingException: No suitable constructor found for type](#)
8. [TypeScript Map](#)
9. [Java Stream skip\(\)](#)
0. [Java Stream findFirst\(\)](#)

**Promoted by [usertesting.com](#)**

Sponsored



**A message from our sponsor**

## Join 7000+ Awesome Developers

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

**Email Address**

**Subscribe**

*\* We do not spam !!*

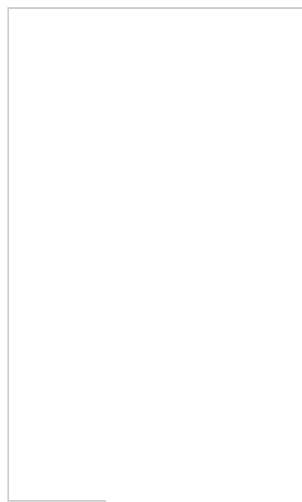
## Leave a Comment

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

## Post Comment







## 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](#)