HowToDoInJava

# Java Stream flatMap()

Last Updated: March 29, 2022 | By: Lokesh Gupta | Java 8 | Java Stream Basics, Java Stream Methods

The **Stream flatMap()** method is **used to *flatten a Stream of collections to a Stream of objects***. The objects are combined from all the collections in the original Stream.

The **flatMap()** operation has the effect of applying a one-to-many transformation to the elements of the *Stream* and then flattening the resulting elements into a new *Stream*.

> `Stream.flatMap()` helps in converting `Stream<Collection<T>>` to `Stream<T>`.
>
> **flatMap() = Flattening + map()**

Table Of Contents



## 1. What is Flattening?

In very layman's terms, **flattening is referred to as merging multiple collections/arrays into one**. Consider the following example.

In this example, we have an array of 3 arrays. After the flattening effect, we will have one result array with all the items from the 3 arrays.

**Flattening Example**

```
Before flattening   : [[1, 2, 3], [4, 5], [6, 7, 8]]

After flattening    : [1, 2, 3, 4, 5, 6, 7, 8]
```

In the following example, `lines` is a stream of lines in the file. Each line consists of multiple words. The `words` stream is a fattened version of all streams into a single stream – consisting of all the words in all the lines.

**Flattening example 2**

```
Stream<String> lines = Files.lines(path, StandardCharsets.UTF_8);

Stream<String> words = lines.flatMap(line -> Stream.of(line.split(" +
```

## 2. Stream flatMap() Method

### 2.1. Method Syntax

The stream `flatMap()` method has the following syntax.

**Syntax**

```
<R> Stream<R> flatMap(Function<? super T,? extends Stream<? extends R:
```

- 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 `flatMap()` operation. If the streams created after `flatMap()` operations return above given types then consider using these functions directly.

**Similar flatMap() Methods**

```
IntStream flatMapToInt(Function<? super T,? extends IntStream> mapper
LongStream flatMapToLong(Function<? super T,? extends LongStream> map
DoubleStream flatMapToDouble(Function<? super T,? extends DoubleStrea
```

## 2.2. Description

- `flatMap()` is an **intermediate** operation and return a new *Stream*.
- It returns a *Stream* consisting of the results of replacing each element of the given stream with the contents of a mapped stream produced by applying the provided mapping function to each element.
- The `mapper` function used for transformation in `flatMap()` is a stateless function and returns only a stream of new values.
- Each mapped *Stream* is closed after its contents have been placed into new *Stream*.
- `flatMap()` operation flattens the stream; opposite to `map()` operation which does not apply flattening.

# 3. Stream flatMap() Examples

## Example 1: Converting Nested Lists into a Single List

Java 8 example of **Stream.flatMap()** function to get a single `List` containing all elements from a list of lists.

This program uses `flatMap()` operation to convert `List<List<Integer>>` to `List<Integer>`.

**Merging Lists into a Single List**

```
List<Integer> list1 = Arrays.asList(1,2,3);
List<Integer> list2 = Arrays.asList(4,5,6);
List<Integer> list3 = Arrays.asList(7,8,9);

List<List<Integer>> listOfLists = Arrays.asList(list1, list2, list3);

List<Integer> listOfAllIntegers = listOfLists.stream()
          .flatMap(x -> x.stream())
          .collect(Collectors.toList());

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

Program output.

**Output**

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

## Example 2: Collecting Nested Arrays into a Sinle List

Java 8 example of **Stream.flatMap()** function to get a single `List` containing all elements from an array of arrays.

**Merging Arrays into a Single List**

```
String[][] dataArray = new String[][]{{"a", "b"},
        {"c", "d"}, {"e", "f"}, {"g", "h"}};

List<String> listOfAllChars = Arrays.stream(dataArray)
              .flatMap(x -> Arrays.stream(x))
              .collect(Collectors.toList());

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

Program output.

**Output**

```
[a, b, c, d, e, f, g, h]
```

Drop me your questions related to **Stream flatMap() 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. Java Stream map() vs flatMap()

2. Java Stream reuse – traverse stream multiple times?
3. Java Stream count() Matches with filter()
4. Java Stream forEach()
5. Java Stream sorted()
6. Java Stream max()
7. Java Stream limit()
8. Java Stream skip()
9. Java Stream findFirst()
0. Java Stream findAny()

** We do not spam !!*

## 1 thought on “Java Stream flatMap()”

### Carlos

January 9, 2020 at 9:40 pm

Congratulations, it was very well explained!

Reply

## Leave a Comment

Name *

Email *

Website

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

**Post Comment**

Search …

## 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