

[Trending Now](#) [Data Structures](#) [Algorithms](#) [Topic-wise Practice](#) [Python](#) [Machine Learning](#) [Data Science](#) [JavaScript](#) [Java](#) [Web Developme](#)

## Stream flatMap() in Java with examples

[Read](#)[Discuss](#)[Practice](#)

**Stream flatMap(Function mapper)** returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Stream flatMap(Function mapper) is an *intermediate operation*. These operations are always lazy. Intermediate operations are invoked on a Stream instance and after they finish their processing, they give a Stream instance as output.

**Note :** Each mapped stream is closed after its contents have been placed into this stream. If a mapped stream is null, an empty stream is used, instead.

**flatMap() V/s map() :**

- 1) [map\(\)](#) takes a Stream and transform it to another Stream. It applies a function on each element of Stream and store return value into new Stream. It does not flatten the stream. But flatMap() is the combination of a map and a flat operation i.e, it applies a function to elements as well as flatten them.
- 2) [map\(\)](#) is used for transformation only, but flatMap() is used for both transformation and flattening.


**Syntax :**



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

where, R is the element type of the new stream.  
Stream is an interface and T is the type  
of stream elements. mapper is a stateless function  
which is applied to each element and the function  
returns the new stream.

**Example 1 :** flatMap() function with provided mapping function.





```
// Java code for Stream flatMap
// (Function mapper) to get a stream by
// replacing the stream with a mapped
// stream by applying the provided mapping function.
import java.util.*;
import java.util.stream.Stream;

class GFG {

    // Driver code
    public static void main(String[] args)
    {

        // Creating a List of Strings
        List<String> list = Arrays.asList("6", "7.4", "4",
```



```
        "1", "2.3");

    // Using Stream flatMap(Function mapper)
    list.stream().flatMap(num -> Stream.of(num)).
        forEach(System.out::println);
    }
}
```

Output :

```
5.6
7.4
4
1
2.3
```

**Example 2 :** flatMap() function with provided operation of mapping string with character at position 2.

```
// Java code for Stream flatMap
// (Function mapper) to get a stream by
// replacing the stream with a mapped
// stream by applying the provided mapping function.
import java.util.*;
import java.util.stream.Stream;

class GFG {

    // Driver code
    public static void main(String[] args)
    {

        // Creating a List of Strings
        List<String> list = Arrays.asList("Geeks", "GFG",
                                         "GeeksforGeeks", "gfg");
```



```
        // Using Stream flatMap(Function mapper)
        list.stream().flatMap(str ->
                                Stream.of(str.charAt(2))).
            forEach(System.out::println);
    }
}
```

Output :

e  
G  
e  
g

### How does flatMap() work ?

As already discussed in the post that flatMap() is the combination of a map and a flat operation i.e, it first applies map function and then flattens the result. Let us consider some examples to understand what exactly flattening a stream is.

#### Example 1 :

The list before flattening :

```
[ [2, 3, 5], [7, 11, 13], [17, 19, 23] ]
```

The list has 2 levels and consists of 3 small lists. After Flattening, it gets transformed into “one level” structure as shown :



```
[ 2, 3, 5, 7, 11, 13, 17, 19, 23 ]
```

### Example 2 :

The list before flattening :

```
[ ["G", "E", "E"], ["K", "S", "F"], ["O", "R", "G"], ["E", "E", "K", "S"] ]
```

The list has 3 levels and consists of 4 small lists. After Flattening, it gets transformed into “one level” structure as shown :

```
["G", "E", "E", "K", "S", "F", "O", "R", "G", "E", "E", "K", "S"]
```

In short, we can say that if there is a **Stream of List of <<Data Type>>** before flattening, then on applying flatMap(), **Stream of <<Data Type>>** is returned after flattening.

### Application :

```
// Java code for Stream flatMap(Function mapper)
import java.util.*;
import java.util.stream.Collectors;

class GFG
{
    // Driver code
    public static void main(String[] args)
    {
        // Creating a list of Prime Numbers
        List<Integer> PrimeNumbers = Arrays.asList(5, 7, 11, 13);

        // Creating a list of Odd Numbers
        List<Integer> OddNumbers = Arrays.asList(1, 3, 5);
```



```
// Creating a list of Even Numbers
List<Integer> EvenNumbers = Arrays.asList(2, 4, 6, 8);

List<List<Integer>> listOfListofInts =
    Arrays.asList(PrimeNumbers, OddNumbers, EvenNumbers);

System.out.println("The Structure before flattening is : " +
    listOfListofInts);

// Using flatMap for transforming and flattening.
List<Integer> listofInts = listOfListofInts.stream()
    .flatMap(list -> list.stream())
    .collect(Collectors.toList());

System.out.println("The Structure after flattening is : " +
    listofInts);
    }
}
```

Output :

The Structure before flattening is : [[5, 7, 11, 13], [1, 3, 5], [2, 4, 6, 8]]

The Structure after flattening is : [5, 7, 11, 13, 1, 3, 5, 2, 4, 6, 8]

Last Updated : 12 Mar, 2018

33

## Similar Reads



Difference Between map() And flatMap() In Java Stream



IntStream flatMap(IntFunction mapper) in Java

