HowToDoInJava

Collecting Stream Items into List in Java

Last Updated: March 14, 2022 By: Lokesh Gupta Java 8 Java Stream Basics

Learn to **collect the items from a Stream into a List** using different ways in Java. We will compare these different techniques so we can decide the best way for any kind of scenario.

Table Of Contents

1. Different Ways to Collect Stream Items into List

There are primarily three ways to collect stream items into a list. Let's compare them.

1.1. Stream.toList()

- The *toList()* method has been **added in Java 16**. It is a default method that collects the stream items into an **unmodifiable List**.

- The returned list is an implementation of `Collections.unmodifiableList(new ArrayList<>(Arrays.asList(stream.toArray())))` where `stream` represents the underlying *Stream* of items.
- The order of the items in the list will be same as the order in stream, if there is any.
- As the returned *List* is unmodifiable; calls to any mutator method will always cause `UnsupportedOperationException` to be thrown.
- It is a **terminal operation**.

```
Stream<String> tokenStream = Stream.of("A", "B", "C", "D");

List<String> tokenList = tokenStream.toList();
```

1.2. Stream.collect(Collectors.toUnmodifiableList())

- This method has been **added in Java 10**. It is a *terminal operation* that collects the stream items into an **unmodifiable List**.
- The returned list is an instance of `Collections.unmodifiableList()` that is filled with stream items using JDK internal APIs able to access private methods of the JDK classes without using the reflection. In this case, the unmodifiable list is an implementation of *SharedSecrets.getJavaUtilCollectionAccess().listFromTrustedArray(list.toArray())* where the `list` is an intermediate and mutable list of stream items.
- The *List* does not allow the *null* values and the whole operation will throw the `NullPointerException` if there is a *null* value in the stream.
- The order of items in the list is the same as the order of items in the stream, if there is any.

```
Stream<String> tokenStream = Stream.of("A", "B", "C", "D");

List<String> tokenList = tokenStream.collect(Collectors.toUnmodifiabl
```

1.3. Stream.collect(Collectors.toList())

- This method has been **added in Java 8**, along with the original Stream API. It is a *terminal operation* that collects the stream items into a **mutable List**.
- The returned list is an instance of `ArrayList` class.
- Similar to other versions, the order of the items in the mutable list will be same as the order in stream, if there is any.

```
Stream<String> tokenStream = Stream.of("A", "B", "C", "D");

List<String> tokenList = tokenStream.collect(Collectors.toList());
```

2. Collecting Stream into LinkedList

Use the *Collectors.toCollection(LinkedList::new)* API along with Stream.collect() API for collecting the Stream items into a LinkedList.

```
Stream<String> tokenStream = Arrays.asList("A", "B", "C", "D").stream

List<String> tokenList = tokenStream
                .collect(Collectors.toCollection(LinkedList::new));
```

3. Filtering a Stream and Collect Items into List

Sometimes we need to find only specific items from the *Stream* and then add only those items to *List*. Here, we can use Stream.filter() method to pass a predicate that will return only those items which match the given pre-condition.

In the given example, we are filtering all employees whose salary is less than 400. Then we are collecting those employees into a List.

```
Stream<Employee> employeeStream = Stream.of(
          new Employee(1, "A", 100),
          new Employee(2, "B", 200),
          new Employee(3, "C", 300),
          new Employee(4, "D", 400),
          new Employee(5, "E", 500),
          new Employee(6, "F", 600));

List<Employee> employeeList = employeeStream
        .filter(e -> e.getSalary() < 400)
        .collect(Collectors.toList());
```

4. Collect Items from Infinite Stream into List

To convert an infinite stream into a list, we must limit the stream to a finite number of elements. Given example will work in the case of a stream of primitives.

```
IntStream infiniteNumberStream = IntStream.iterate(1, i -> i+1);

List<Integer> integerlist = infiniteNumberStream.limit(10)
        .boxed()
        .collect(Collectors.toList());
```

5. Conclusion

In this tutorial, we learned the different ways to work with streams and collect the stream items in a List.

As a general guideline, we can use *Stream.toList()* for unmodifiable lists, and use the *Stream.collect(Collectors.toList())* for modifiable lists.

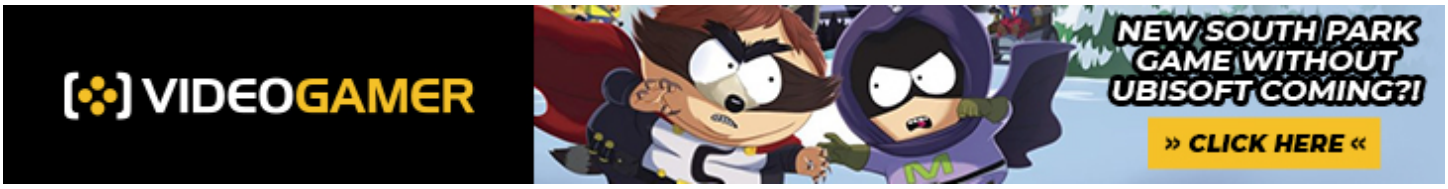
To collect items in any other List types, we must use the *Stream.collect(Collectors.toCollection(LinkedList::new))* version of the solutions.

Happy Learning !!

Sourcecode on Github

Recommended Reading:

1. Collecting Stream Items into Map in Java
2. Collecting Stream of Primitives into Collection or Array
3. Getting Distinct Stream Items by Comparing Multiple Fields
4. Append or Prepend Items to a Stream
5. Java Stream – Get Object with Max Date From a List
6. Java Stream reuse – traverse stream multiple times?
7. Finding Max and Min from List using Streams
8. Removing Items from an Array in Java
9. Python find largest N (top N) or smallest N items
0. Adding Multiple Items to ArrayList



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