HowToDoInJava

Creating Infinite Streams in Java

Last Updated: March 4, 2022 By: Lokesh Gupta Java 8 Java Stream Basics

Learn to generate an infinite stream of elements in Java. We will use `Stream.generate()` and `Stream.iterate()` methods to get the infinite streams.

Table Of Contents

1. Overview

This is very important to note that **Java Streams are lazy** by design. So:

- The *generate()* and *iterate()* methods are intermediate operations, so the **actual element creation doesn't begin until a terminal operation is invoked**.
- Be careful and **use the *limit()* to restrict the number of elements** in the stream, before invoking the terminal operation. Else the stream generation will continue infinitely.
- Use *iterate()* method to create **ordered** stream elements, and *generate()* method to **unordered** stream elements.

2. Infinite Streams with `iterate()` Method

2.1. Method Syntax

Syntax

```
static <T> Stream<T> iterate(T seed, UnaryOperator<T> f)
```

The *Stream.iterate()* method returns an **infinite sequential ordered stream**. The first element (index 0) in the Stream will be the provided `seed`. For `n > 0`, the element at position `n`, will be the result of applying the function `f` to the element at position `n - 1`.

2.2. Example – Sequence of Int Values

In the given example, we are creating an *infinite stream of even numbers starting from* `0`. Then we are collecting the first 10 elements from the stream to a list.

```
List<Integer> ints = IntStream.iterate(0, i -> i + 2)
            .mapToObj(Integer::valueOf)
            .limit(10)
            .collect(Collectors.toList());
```

3. Infinite Streams with `generate()` Method

3.1. Method Syntax

```
static <T> Stream<T> generate(Supplier<T> s)
```

It returns an **infinite sequential unordered stream** where each element is generated by the provided `Supplier`. This is **suitable for generating constant streams, streams of random elements**, etc.

3.2. Example – Stream of Random Numbers

The following example creates a stream of 10 random numbers between 0 and 99.

```
List<Integer> randomNumbers = Stream.generate(() -> (new Random()).nextInt(100))
                .limit(10)
                .collect(Collectors.toList());
```

3.3. Example – Stream of Custom Objects

The following example creates an infinite stream of employees and takes the first 5 employees from the stream.

```
List<Employee> employees = Stream.generate(Employee::create)
              .limit(5)
              .collect(Collectors.toList());
```

Where `Employee` class is this:

```
import java.io.Serializable;
import java.util.Random;

public class Employee implements Serializable {

    private static final long serialVersionUID = 1L;
    private static final Random r = new Random(Integer.MAX_VALUE);

    private long id;
    private String name;
    private double salary;
```

```
    //All-args constructor, getters and setters are hidden for brevity

    public static Employee create() {
        //Get the employee id in more predictable manner
        //e.g. Max id present in databse + 1

        Employee obj = new Employee(r.nextInt(), "", 0.0d);
        return obj;
    }
}
```

4. Conclusion

In this Java stream tutorial, we learned to create and operate on infinite streams. We learned to use the *generate()* and *iterate()* functions for creating *bounded infinite streams* with examples.

Happy Learning !!

Sourcecode on Github

Recommended Reading:

1. Creating Streams in Java
2. How to Detect infinite loop in LinkedList with Example
3. Boxed Streams in Java
4. Using 'if-else' Conditions with Java Streams
5. Sorting Streams in Java
6. Applying Multiple Conditions on Java Streams
7. Finding Max and Min from List using Streams
8. Java Streams API
9. Primitive Type Streams in Java
0. Creating a Temporary File in Java

1 thought on “Creating Infinite Streams in Java”

Tien Dat

May 1, 2019 at 5:43 pm

Dear,

Thanks for your useful examples regarding the infinite stream of objects.

We have a question, regarding the serialization of such infinite stream.

For instance that we generate an infinite stream of custom objects.
We would like to serialize it and pass it as and java.io.InputStream to be consumed.

The question is the only InputStream implementation we can find, which seems to be appropriate for this usecase, is the ByteArrayInputStream. However, we all know that the ByteArrayInputStream buffer the serialized objects in memory, and it clearly will lead to OOM exception once the stream grows large enough.

What is your solution for this use case?

Best
Tien Dat

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 ...