Slicing Operations in Stream

In this lesson, we will look at some of the most common slicing methods present in the Streams API.

We'll cover the following

- 1. distinct()
- 2. limit()
- 3) skip()

The slicing operations are intermediate operations, and, as the name implies, they are used to slice a stream.

Now, we will look at some of the most common slicing methods present in Streams API.

`1. distinct()`

The first operation that we are going to discuss is `distinct()`. It returns a stream consisting of the distinct elements (according to Object.equals(Object)) of this stream.

So, if you have a stream of custom objects then your custom class should override `equals()` and `hashcode()` methods.

Let’s look at an example to understand `distinct()` better. In the below example, we have a list of countries. The list can contain duplicate elements as well. We need to print all the distinct countries.

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class StreamDemo {

    public static void main(String[] args) {
        List<String> countries = new ArrayList<>();
        countries.add("India");
        countries.add("USA");
        countries.add("China");
        countries.add("India");

        countries.add("UK");
        countries.add("China");

        countries.stream()
                .distinct()
                .forEach(System.out::println);
    }
}
```

2. limit()

This is also an intermediate function. It returns a stream consisting of the elements of this stream, truncated to be no longer than maxSize in length.

Below is the method syntax:

> **Stream<T> limit(long maxSize)**

In our example above, we used the `distinct()` method to get only the distinct countries. Now we will limit the number of countries to three.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class StreamDemo {

    public static void main(String[] args) {
        List<String> countries = new ArrayList<>();
        countries.add("India");
        countries.add("USA");
        countries.add("China");
        countries.add("India");
        countries.add("UK");
        countries.add("China");

        countries.stream()
                .distinct()
                .limit(3)
                .forEach(System.out::println);
    }
}
```

3) skip()

Like distinct() and limit(), skip() is also an intermediate method. It returns a stream consisting of the remaining elements of this stream after discarding the first n elements of the stream.

Below is the syntax of this method.

> **Stream<T> skip(long n)**

If this stream contains fewer than n elements then an empty stream will be returned.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class StreamDemo {

    public static void main(String[] args) {
        List<String> countries = new ArrayList<>();
        countries.add("India");
        countries.add("USA");
        countries.add("China");
        countries.add("India");
        countries.add("UK");
        countries.add("China");

        countries.stream()
                .distinct()
                .skip(2)
                .forEach(System.out::println);
    }
}
```

This is all we have for slicing functions. In the next lesson, we will look at matching functions.