

JAVA 8 - STREAMS

Stream is a new abstract layer introduced in Java 8. Using stream, you can process data in a declarative way similar to SQL statements. For example, consider the following SQL statement –

```
SELECT max(salary), employee_id, employee_name FROM Employee
```

The above SQL expression automatically returns the maximum salaried employee's details, without doing any computation on the developer's end. Using collections framework in Java, a developer has to use loops and make repeated checks. Another concern is efficiency; as multi-core processors are available at ease, a Java developer has to write parallel code processing that can be pretty error-prone.

To resolve such issues, Java 8 introduced the concept of stream that lets the developer to process data declaratively and leverage multicore architecture without the need to write any specific code for it.

What is Stream?

Stream represents a sequence of objects from a source, which supports aggregate operations. Following are the characteristics of a Stream –

- **Sequence of elements** – A stream provides a set of elements of specific type in a sequential manner. A stream gets/computes elements on demand. It never stores the elements.
- **Source** – Stream takes Collections, Arrays, or I/O resources as input source.
- **Aggregate operations** – Stream supports aggregate operations like filter, map, limit, reduce, find, match, and so on.
- **Pipelining** – Most of the stream operations return stream itself so that their result can be pipelined. These operations are called intermediate operations and their function is to take input, process them, and return output to the target. collect method is a terminal operation which is normally present at the end of the pipelining operation to mark the end of the stream.
- **Automatic iterations** – Stream operations do the iterations internally over the source elements provided, in contrast to Collections where explicit iteration is required.

Generating Streams

With Java 8, Collection interface has two methods to generate a Stream –

- **stream** – Returns a sequential stream considering collection as its source.
- **parallelStream** – Returns a parallel Stream considering collection as its source.

```
List<String> strings = Arrays.asList("abc", "", "bc", "efg", "abcd", "", "jkl");  
List<String> filtered = strings.stream().filter(string ->  
!string.isEmpty()).collect(Collectors.toList());
```

forEach

Stream has provided a new method 'forEach' to iterate each element of the stream. The following code segment shows how to print 10 random numbers using forEach.

```
Random random = new Random();  
random.ints().limit(10).forEach(System.out::println);
```

map

The 'map' method is used to map each element to its corresponding result. The following code segment prints unique squares of numbers using map.

```
List<Integer> numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);  
//get list of unique squares  
List<Integer> squaresList = numbers.stream().map( i ->  
i*i).distinct().collect(Collectors.toList());
```

filter

The 'filter' method is used to eliminate elements based on a criteria. The following code segment prints a count of empty strings using filter.

```
List<String>strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");  
//get count of empty string  
int count = strings.stream().filter(string -> string.isEmpty()).count();
```

limit

The 'limit' method is used to reduce the size of the stream. The following code segment shows how to print 10 random numbers using limit.

```
Random random = new Random();  
random.ints().limit(10).forEach(System.out::println);
```

sorted

The 'sorted' method is used to sort the stream. The following code segment shows how to print 10 random numbers in a sorted order.

```
Random random = new Random();  
random.ints().limit(10).sorted().forEach(System.out::println);
```

Parallel Processing

parallelStream is the alternative of stream for parallel processing. Take a look at the following code segment that prints a count of empty strings using parallelStream.

```
List<String> strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");  
//get count of empty string  
int count = strings.parallelStream().filter(string -> string.isEmpty()).count();
```

It is very easy to switch between sequential and parallel streams.

Collectors

Collectors are used to combine the result of processing on the elements of a stream. Collectors can be used to return a list or a string.

```
List<String>strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");  
List<String> filtered = strings.stream().filter(string ->  
!string.isEmpty()).collect(Collectors.toList());  
  
System.out.println("Filtered List: " + filtered);  
String mergedString = strings.stream().filter(string ->  
!string.isEmpty()).collect(Collectors.joining(", "));  
System.out.println("Merged String: " + mergedString);
```

Statistics

With Java 8, statistics collectors are introduced to calculate all statistics when stream processing is being done.

```
List<Integer> numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);

IntSummaryStatistics stats = integers.stream().mapToInt((x) -> x).summaryStatistics();

System.out.println("Highest number in List : " + stats.getMax());
System.out.println("Lowest number in List : " + stats.getMin());
System.out.println("Sum of all numbers : " + stats.getSum());
System.out.println("Average of all numbers : " + stats.getAverage());
```

Stream Example

Create the following Java program using any editor of your choice in, say, C:\> JAVA.

Java8Tester.java

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.IntSummaryStatistics;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;
import java.util.Map;

public class Java8Tester {
    public static void main(String args[]){
        System.out.println("Using Java 7: ");

        // Count empty strings
        List<String> strings = Arrays.asList("abc", "", "bc", "efg", "abcd", "", "jkl");
        System.out.println("List: " +strings);
        long count = getCountEmptyStringUsingJava7(strings);

        System.out.println("Empty Strings: " + count);
        count = getCountLength3UsingJava7(strings);

        System.out.println("Strings of length 3: " + count);

        //Eliminate empty string
        List<String> filtered = deleteEmptyStringsUsingJava7(strings);
        System.out.println("Filtered List: " + filtered);

        //Eliminate empty string and join using comma.
        String mergedString = getMergedStringUsingJava7(strings, ", ");
        System.out.println("Merged String: " + mergedString);
        List<Integer> numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);

        //get list of square of distinct numbers
        List<Integer> squaresList = getSquares(numbers);
        System.out.println("Squares List: " + squaresList);
        List<Integer> integers = Arrays.asList(1,2,13,4,15,6,17,8,19);

        System.out.println("List: " +integers);
        System.out.println("Highest number in List : " + getMax(integers));
        System.out.println("Lowest number in List : " + getMin(integers));
        System.out.println("Sum of all numbers : " + getSum(integers));
        System.out.println("Average of all numbers : " + getAverage(integers));
        System.out.println("Random Numbers: ");

        //print ten random numbers
        Random random = new Random();

        for(int i=0; i < 10; i++){
            System.out.println(random.nextInt());
        }

        System.out.println("Using Java 8: ");
        System.out.println("List: " +strings);
```

```

count = strings.stream().filter(string->string.isEmpty()).count();
System.out.println("Empty Strings: " + count);

count = strings.stream().filter(string -> string.length() == 3).count();
System.out.println("Strings of length 3: " + count);

filtered = strings.stream().filter(string -
>!string.isEmpty()).collect(Collectors.toList());
System.out.println("Filtered List: " + filtered);

mergedString = strings.stream().filter(string -
>!string.isEmpty()).collect(Collectors.joining(", "));
System.out.println("Merged String: " + mergedString);

squaresList = numbers.stream().map( i -
>i*i).distinct().collect(Collectors.toList());
System.out.println("Squares List: " + squaresList);
System.out.println("List: " + integers);

IntSummaryStatistics stats = integers.stream().mapToInt((x) ->x).summaryStatistics();

System.out.println("Highest number in List : " + stats.getMax());
System.out.println("Lowest number in List : " + stats.getMin());
System.out.println("Sum of all numbers : " + stats.getSum());
System.out.println("Average of all numbers : " + stats.getAverage());
System.out.println("Random Numbers: ");

random.ints().limit(10).sorted().forEach(System.out::println);

//parallel processing
count = strings.parallelStream().filter(string -> string.isEmpty()).count();
System.out.println("Empty Strings: " + count);
}

private static int getCountEmptyStringUsingJava7(List<String> strings){
    int count = 0;

    for(String string: strings){

        if(string.isEmpty()){
            count++;
        }
    }
    return count;
}

private static int getCountLength3UsingJava7(List<String> strings){
    int count = 0;

    for(String string: strings){

        if(string.length() == 3){
            count++;
        }
    }
    return count;
}

private static List<String> deleteEmptyStringsUsingJava7(List<String> strings){
    List<String> filteredList = new ArrayList<String>();

    for(String string: strings){

        if(!string.isEmpty()){
            filteredList.add(string);
        }
    }
    return filteredList;
}

```

```

private static String getMergedStringUsingJava7(List<String> strings, String
separator){
    StringBuilder stringBuilder = new StringBuilder();

    for(String string: strings){

        if(!string.isEmpty()){
            stringBuilder.append(string);
            stringBuilder.append(separator);
        }
    }
    String mergedString = stringBuilder.toString();
    return mergedString.substring(0, mergedString.length()-2);
}

private static List<Integer> getSquares(List<Integer> numbers){
    List<Integer> squaresList = new ArrayList<Integer>();

    for(Integer number: numbers){
        Integer square = new Integer(number.intValue() * number.intValue());

        if(!squaresList.contains(square)){
            squaresList.add(square);
        }
    }
    return squaresList;
}

private static int getMax(List<Integer> numbers){
    int max = numbers.get(0);

    for(int i=1;i < numbers.size();i++){

        Integer number = numbers.get(i);

        if(number.intValue() > max){
            max = number.intValue();
        }
    }
    return max;
}

private static int getMin(List<Integer> numbers){
    int min = numbers.get(0);

    for(int i=1;i < numbers.size();i++){
        Integer number = numbers.get(i);

        if(number.intValue() < min){
            min = number.intValue();
        }
    }
    return min;
}

private static int getSum(List numbers){
    int sum = (int)(numbers.get(0));

    for(int i=1;i < numbers.size();i++){
        sum += (int)numbers.get(i);
    }
    return sum;
}

private static int getAverage(List<Integer> numbers){
    return getSum(numbers) / numbers.size();
}
}

```

Verify the Result

Compile the class using **javac** compiler as follows –

```
$javac Java8Tester.java
```

Now run the Java8Tester as follows –

```
$java Java8Tester
```

It should produce the following result –

```
Using Java 7:  
List: [abc, , bc, efg, abcd, , jkl]  
Empty Strings: 2  
Strings of length 3: 3  
Filtered List: [abc, bc, efg, abcd, jkl]  
Merged String: abc, bc, efg, abcd, jkl  
Squares List: [9, 4, 49, 25]  
List: [1, 2, 13, 4, 15, 6, 17, 8, 19]  
Highest number in List : 19  
Lowest number in List : 1  
Sum of all numbers : 85  
Average of all numbers : 9  
Random Numbers:  
-1279735475  
903418352  
-1133928044  
-1571118911  
628530462  
18407523  
-881538250  
-718932165  
270259229  
421676854  
Using Java 8:  
List: [abc, , bc, efg, abcd, , jkl]  
Empty Strings: 2  
Strings of length 3: 3  
Filtered List: [abc, bc, efg, abcd, jkl]  
Merged String: abc, bc, efg, abcd, jkl  
Squares List: [9, 4, 49, 25]  
List: [1, 2, 13, 4, 15, 6, 17, 8, 19]  
Highest number in List : 19  
Lowest number in List : 1  
Sum of all numbers : 85  
Average of all numbers : 9.4444444444444445  
Random Numbers:  
-1009474951  
-551240647  
-2484714  
181614550  
933444268  
1227850416  
1579250773  
1627454872  
1683033687  
1798939493  
Empty Strings: 2
```

```
Loading [MathJax]/jax/output/HTML-CSS/jax.js
```