Collectors: Aggregation Operations

In this lesson, we will learn about methods of the Collectors class, which are used for aggregation.

We'll cover the following

In this lesson, we will look at some of the methods of the `Collectors` class thaat help us aggregate the data in streams, e.g., sum, average, etc.

1) `counting()`

This function returns a `Collector` that counts the number of the input elements.

Suppose we have a list of employees, and we need the count of employees with an age more than 30.

In this case, we can use the `counting()` method as shown below.

```java
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class CollectorsDemo {

    public static void main(String args[]) {
        List<Employee> employeeList = new ArrayList<>();
        employeeList.add(new Employee("Alex", 23, 23000));
        employeeList.add(new Employee("Ben", 63, 25000));
        employeeList.add(new Employee("Dave", 34, 56000));
        employeeList.add(new Employee("Jodi", 43, 67000));
        employeeList.add(new Employee("Ryan", 53, 54000));

        long count = employeeList.stream()
```

```
                .filter(emp -> emp.getAge() > 30)
                .collect(Collectors.counting()); // Using the counting() method to get count of em


        System.out.println(count);
    }
}

class Employee {
    String name;
    int age;
    int salary;

    Employee(String name) {
        this.name = name;
    }

    Employee(String name, int age, int salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public int getSalary() {
        return salary;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", salary=" + salary +
                '}';
    }
}
```

2) `Collectors.summingInt(ToIntFunction<? super T> mapper)`

This method returns a Collector that produces the sum of an integer-valued function applied to the input elements.

This method takes a `ToIntFunction` as a parameter.

```
package com.collectors;

import java.util.ArrayList;
```

```java
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;

import java.util.stream.Collectors;

public class CollectorsDemo {

    public static void main(String args[]) {
        List<Employee> employeeList = new ArrayList<>();
        employeeList.add(new Employee("Alex", 23, 23000));
        employeeList.add(new Employee("Ben", 63, 25000));
        employeeList.add(new Employee("Dave", 34, 56000));
        employeeList.add(new Employee("Jodi", 43, 67000));
        employeeList.add(new Employee("Ryan", 53, 54000));

        // Using summingInt() method to get the sum of salaries of all employees.
        int count = employeeList.stream()
                .collect(Collectors.summingInt(emp -> emp.getSalary()));

        System.out.println(count);
    }
}

class Employee {
    String name;
    int age;
    int salary;

    Employee(String name) {
        this.name = name;
    }

    Employee(String name, int age, int salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public int getSalary() {
        return salary;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", salary=" + salary +
                '}';
    }
}
```

There are similar functions for long and double as well, namely `summingLong()` and `summingDouble()`, respectively.

3) `Collectors.averagingInt(ToIntFunction<? super T> mapper)`

This method returns a `Collector` that produces the arithmetic mean of an integer-valued function applied to the input elements. If no elements are present, the result is 0.

This method takes a `ToIntFunction` as a parameter.

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import java.util.stream.Collectors;

public class CollectorsDemo {

    public static void main(String args[]) {
        List<Employee> employeeList = new ArrayList<>();
        employeeList.add(new Employee("Alex", 23, 23000));
        employeeList.add(new Employee("Ben", 63, 25000));
        employeeList.add(new Employee("Dave", 34, 56000));
        employeeList.add(new Employee("Jodi", 43, 67000));
        employeeList.add(new Employee("Ryan", 53, 54000));

        // Using averagingInt() method to get the average of salaries of all employees.
        double average = employeeList.stream()
                .collect(Collectors.averagingInt(emp -> emp.getSalary()));

        System.out.println(average);
    }
}

class Employee {
    String name;
    int age;
    int salary;

    Employee(String name) {
        this.name = name;
    }

    Employee(String name, int age, int salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public int getAge() {

        return age;
    }

    public int getSalary() {
        return salary;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", salary=" + salary +
                '}';
    }
}
```

There are similar functions for long and double as well, namely `averagingLong()`, and `averagingDouble()` respectively.

3) `minBy(Comparator<? super T> comparator)`

It returns a `Collector` that returns the minimum element based on the given comparator.

The returned value is wrapped in an `Optional` instance.

```
import java.util.*;
import java.util.stream.Collectors;

public class CollectorsDemo {

    public static void main(String args[]) {
        List<Employee> employeeList = new ArrayList<>();
        employeeList.add(new Employee("Alex", 23, 23000));
        employeeList.add(new Employee("Ben", 63, 25000));
        employeeList.add(new Employee("Dave", 34, 56000));
        employeeList.add(new Employee("Jodi", 43, 67000));
        employeeList.add(new Employee("Ryan", 53, 54000));

        //Using minBy() method to get the employee with min salary.
        Optional<Employee> employee = employeeList.stream()
                .collect(Collectors.minBy(Comparator.comparing(Employee::getSalary)));

        System.out.println(employee.get().getName());
    }
}

class Employee {
    String name;
    int age;
    int salary;

    Employee(String name) {
        this.name = name;
    }

    Employee(String name, int age, int salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public int getSalary() {
        return salary;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", salary=" + salary +
                '}';
    }
}
```

4) `maxBy(Comparator<? super T> comparator)`

It returns a `Collector` that returns the minimum element based on the given comparator.

The returned value is wrapped in an `Optional` instance.

```
import java.util.*;
import java.util.stream.Collectors;

public class CollectorsDemo {

    public static void main(String args[]) {
        List<Employee> employeeList = new ArrayList<>();
        employeeList.add(new Employee("Alex", 23, 23000));
        employeeList.add(new Employee("Ben", 63, 25000));
        employeeList.add(new Employee("Dave", 34, 56000));
        employeeList.add(new Employee("Jodi", 43, 67000));
        employeeList.add(new Employee("Ryan", 53, 54000));
        //Using maxBy() method to get the employee with max salary.
        Optional<Employee> employee = employeeList.stream()

                .collect(Collectors.maxBy(Comparator.comparing(Employee::getSalary)));

        System.out.println(employee.get().getName());
    }
}

class Employee {
    String name;
    int age;
    int salary;

    Employee(String name) {
        this.name = name;
    }

    Employee(String name, int age, int salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public int getSalary() {
        return salary;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", salary=" + salary +
                '}';
    }
}
```

5) `summarizingInt(ToIntFunction<? super T> mapper)`

It returns a `Collector` that applies an int-producing mapping function to each input element and returns summary statistics for the resulting values.

```
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;
```

```
public class CollectorsDemo {

    public static void main(String args[]) {


        IntSummaryStatistics summarizingInt = Stream.of("1", "2", "3")
                .collect(Collectors.summarizingInt(Integer::parseInt));
        System.out.println(summarizingInt);
    }
}
```

6) `joining()`

It returns a `Collector` that concatenates the input elements into a `String`, in the encounter order. It also has few overloaded versions which allow us to provide delimiters and prefix and suffix strings.

```
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class CollectorsDemo {

    public static void main(String args[]) {
        // Joining all the strings.
        String joinedString = Stream.of("hello", "how", "are" , "you")
                .collect(Collectors.joining());
        System.out.println(joinedString);

        // Joining all the strings with space in between.
        joinedString = Stream.of("hello", "how", "are" , "you")
                .collect(Collectors.joining(" "));
        System.out.println(joinedString);

        // Joining all the strings with space in between and a prefix and suffix.
        joinedString = Stream.of("hello", "how", "are" , "you")
                .collect(Collectors.joining(" " , "prefix","suffix"));
        System.out.println(joinedString);
    }
}
```

In the next lesson, we will take a look at grouping operations using `Collectors`.